Advancing Data Curation

With Metadata and Statistical Relational Learning

vorgelegt von
Dipl.-Inf (FH) M.Ed.
Larysa Visengeriyeva

von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktorin der Ingenieurwissenschaften
-Dr.-Ing.-

genehmigte Dissertation

Promotionsausschuss:
Vorsitzender: Prof. Dr. Klaus-Robert Müller
Gutachter: Prof. Dr. Ziawasch Abedjan
Gutachter: Prof. Dr. Ulf Leser
Gutachter: Prof. Dr. Felix Naumann
Tag der wissenschaftlichen Aussprache: 14.Februar 2020

Berlin 2020

Declaration of Authorship

I, Larysa Visengeriyeva, declare that this dissertation entitled: *"Advancing Data Curation With Metadata and Statistical Relational Learning"*, and the work presented in it is my own.

I confirm that:

1. This work was done completely while in candidature for a research degree at Technische Universität Berlin.
2. Where any part of this dissertation has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated.
3. Where I have used or consulted the published work of others, this is always clearly attributed.
4. Where I have quoted from the works of others, the source is always given. With the exception of such quotations, this dissertation is entirely my own work.
5. I have acknowledged all sources used for the purpose of this work.

Date: Signature:

"Our actions may be impeded...but there can no impeding our intentions or dispositions. Because we can accommodate and adapt. The mind adapts and converts to its own purposes the obstacle to our acting.

The impediment to action advances action. What stands in the way becomes the way."

— Marcus Aurelius

Zusammenfassung

Jedes Data Science Projekt hängt von sauberen und stimmigen Daten ab, denn die Qualität der Daten bestimmt die Qualität der Machine Learning Modelle und dementsprechend korrekt sind die aus den Daten gewonnenen Erkenntnisse.

In dieser Dissertation gehen wir das Problem der Datenbereinigung an und präsentieren drei Ansätze, um Datenfehler zu erkennen und zu beheben:

(1) Wir erstellen ein Mapping, das den systematischen Zusammenhang zwischen Qualitätsproblemen von Daten und den Metadaten widerspiegelt. Wir verwenden dieses Mapping als eine generische Lösung, um Datenfehler zu erkennen und um den Prozess des Data Cleaning signifikant zu beschleunigen. (2) Wir präsentieren zwei ganzheitliche Ansätze zur effektiven Kombination verschiedener Methoden der Fehlererkennung, um die Effektivität der Fehlererkennung zu erhöhen. Unsere Methoden basieren auf State-of-the-Art Ensemble-Learning Algorithmen und integrieren die Metadaten, um die Fehlererkennung zu optimieren. (3) Wir präsentieren eine probabilistische Methode für die Verbesserung der Datenqualität. Diese Methode basiert auf Statistical Relational Learning und der probabilistischen Inferenz. Wir verwenden den Markov Logik Formalismus, um Datenqualitätsregeln deklarativ als Logiksätze erster Ordnung zu modellieren. Außerdem ermöglicht die Markov Logik die Verwendung probabilistischer Inferenz über Datenqualitäsregeln, um Datenfehler zu erkennen und eine wahrscheinliche Lösung vorzuschlagen.

Abstract

The foundation of every data science project depends on clean data because the quality of the data determines the quality of the insights derived from data by using machine learning or analytics. In this dissertation, we tackle the problem of data cleaning and provide three approaches to advance data error detection and repair: (1) We establish a mapping that reflects the connection between data quality issues and extractable dataset's metadata, and propose this mapping as a guideline for rapid prototyping of an error detection strategy; (2) We introduce two holistic approaches for effectively combining different error detection strategies to increase the efficacy of error detection. Our methods are based on state-of-the-art ensemble learning algorithms and incorporate the metadata of the dataset; and (3) We propose an approach for addressing data quality issues by formulating a set of data cleaning rules without the manual specification of the rules execution order. The concepts of statistical relational learning and probabilistic inference provide the foundation for our method. We use the Markov logic formalism, because it declaratively models data quality rules as first-order logic sentences. Markov logic allows the usage of probabilistic joint inference over data cleaning rules to detect data errors and suggest a repair.

Acknowledgements

This dissertation reveals several years of dedication, commitment and research that gave me the opportunity to explore many fields of interest to create this work. I could not have accomplished this dissertation without the number of people that supported me along this journey.

During my time at TU Berlin, I had the privilege of working with Professor Ziawasch Abedjan. He took me as his PhD student and allowed me to conduct my research in any direction. I am thankful to him for establishing a safe research environment for everyone to ask questions, define ambitious goals, and feel to be a part of a great team. Our weekly meetings were crucial in generating ideas and shaping my research mindset. I am indebted for his clear guidance and constructive feedback. His extensive scientific expertise in data integration, data cleaning, and data profiling gave me enormous support towards accomplishing my research goals and forced me into a research tempo that speeded up my dissertation progress.

My research work would never be possible without Professor Felix Naumann, who encouraged me to keep working in the *Data Cleaning* research field when it seemed like an impossible task.

I had a pleasure to meet and work with the incredible researcher Professor Sebastian Riedel at UCL London. His work and his vision of probabilistic programming languages inspired me, so I joined him in this research direction. During that time, I was introduced to a fascinating research environment with the brightest minds of the present time. I enormously enjoyed our discussions about our work on the MLN parser, continuous benchmarking for probabilistic programming, and connecting Machine Learning with Software Engineering. It was an amazing, meaningful and intense time where I learned a lot and always have been looking forward to it.

I also thank my research comrade Alan Akbik for infecting me with the excitement for science and for supporting me in all my curios adventures. Everything started with a question *"What is Markov logic?"*. Alan has become a valuable colleague and a true friend. I still believe that we should build a start-up because everything we started as a team resulted in success.

I am indebted Manohar Kaul for serving me as an invaluable mentor, teaching me scientific methods and injecting the *never give up*-mindset into my head. I also would like to thank Alexander Löser for the opportunity to work on the exiting MIA project at the beginning of my career as a researcher.

It was a pleasure to work with Sebastian Schelter. I am thankful to him for all his support and teaching me always to ask the *why*-question. I am really happy to have Sebastian as a friend.

The awesome research BigDaMa group consisting of Felix Neutatz, Mohammad Mahdavi Lahijani, Mahdi Esmailoghli, Maximilian Dohlus, Binger Chen, and Milad Abbaszade exceptionally supported me during my dissertation process. I always enjoyed our team spirit and our ability to become a *collective brain* while working on our projects.

My PhD journey would never be so well organized without our organization talent Claudia Gantzer and our server-wizard Lutz Friedel. Thank you for all your great support.

Most importantly, I would like to thank my husband Stefan, who encouraged me throughout the whole dissertation process, throughout all the depths and heights, and who served as the motivator for my work. I would not be here without you and your endless love. My awesome kids and my whole family, this dissertation is dedicated to you.

Table of Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

In this introductory chapter, we explain the impact of possessing high-quality data for institutions and enterprises, as well as describe the importance of data quality in data science workflows. Based on this analysis, we specify the research goals and give an overview of the contributions made in this dissertation.

1.1 Data Quality Importance in the Data Science Workflow

The two global trends that disrupt the economy and our daily lives are the *data-driven world*, due to the exponentially-growing amount of digitally-collected data, and the increasing importance of *data science*, which derives insights from this tremendous amount of data [221]. We refer to *data science* as an umbrella term gathering algorithms and techniques from several disciplines, such as *statistics, software engineering*, and *machine learning* [103, 29, 171, 125, 187].

The growing amount of collected data can be used for several purposes, such as data-driven discovery and innovation, massive data integration, and enhanced decision making, thus leading to the adoption of machine learning algorithms to achieve these goals. Data becomes increasingly ubiquitous to the economy and society because of the data-driven decisions in the automobile industry, retail, media, health care, agriculture, and government [200].

Acquiring predictive and prescriptive insights, which are powered by data science, demands trustworthy data. Generally, practitioners and experts report that they do not trust in data, regardless of the data source, because, frequently, data has quality issues [161, 201]. According to the recently published MIT SMR Connections report, "Data, Analytics, and AI: How Trust Delivers Value", the majority of their respondents do not trust

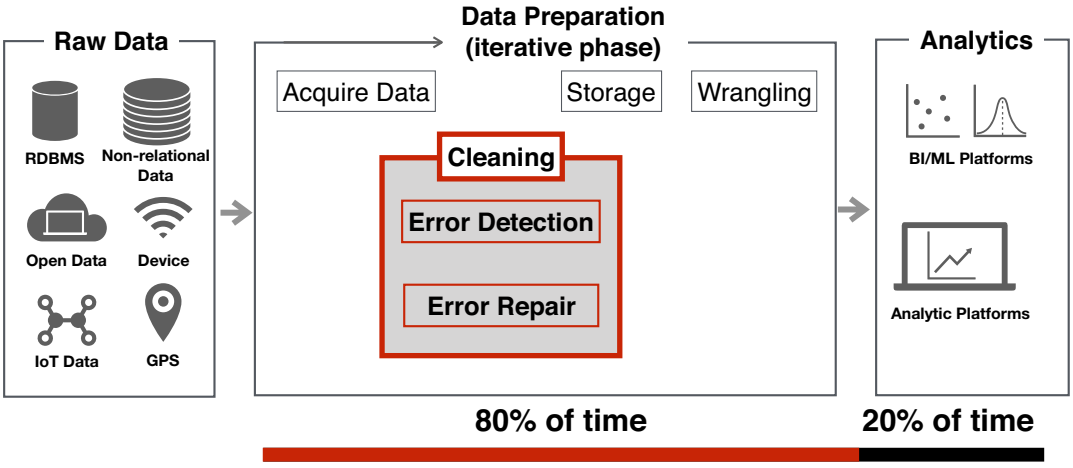

Figure 1.1: An overview of a typical data science workflow that includes the three main stages, namely data acquisition, data preparation, and data analytics.

their data judged by quality dimensions, such as *timeliness, accuracy, completeness,* and *relevance* [161]. Furthermore, poor data quality becomes the reason behind great financial losses. According to Gartner research [226], "organizations believe poor data quality to be responsible for an average of $15 million per year in losses." A study conducted by IBM estimated that poor data quality costs the US economy $3.1 trillion per year [199]. In a typical enterprise, it is expected that roughly 15% of data is either not available or wrong, and 2% of the records are getting stale on a monthly basis in a customer database [223].

Consequently, with the growing amount of data, the need for data quality management increases as well because real-word data is often *dirty*, meaning that data is *inconsistent, duplicated, stale, incomplete,* and/or *inaccurate* [82, 195, 161, 201, 200]. In order to improve the quality of data, we first consider the origin of the irregularities in data by studying an exemplified data science workflow.

The data science workflow refers to a type of "programming activity where the goal is to obtain insights from data" [103]. Roughly speaking, the typical data science workflow consists of three main phases: *data acquisition, data preparation*, and *analysis & interpreting the outputs* (e.g. visualization and storytelling) [102]. Figure 1.1 shows the core steps involved in a typical data science workflow. The initial step in any data science workflow is to acquire the data to be analyzed. Typically, data is being integrated from various resources and has different formats. This requires the second step in the data science workflow – the data preparation, which is "an iterative and agile process for exploring, combining, cleaning and transforming raw data into curated datasets for data integration, data science, data discovery and analytics/business intelligence (BI) use cases" [212]. The core of data science is the analysis phase: writing and executing machine learning algorithms to obtain insights from data. Notably, even though the preparation phase is an intermediate

phase aimed to prepare data for analysis, this phase is reported to be the most expensive in respect to resources and time [223, 167, 142]. Data preparation is a critical activity in the data science workflow because it is paramount to avoid the propagation of data errors to the next phase, data analysis, as this would result in the derivation of wrong insights from the data [201, 65].

Taking the discussion above into account, maintaining data quality is a critical aspect in data management due to the decision-making and analytics demand of accurate, consistent, complete and timely data, in order to avoid the "garbage in - garbage out" problem [223, 167]. As advanced technologies, such as machine learning, deep learning, topic modeling, text mining and sentiment analysis becomes pervasive in enterprises, the output of one predictive model will influence the next stage of data analytics. The risk is that a minor error at the initial step will propagate, and create even more errors [201, 199].

Evaluating the quality of a dataset is a critical first step in every data preparation phase of any data science and analytic workflow [186, 229], because data is usually integrated from different sources with different degrees of reliability. Therefore, the integrated dataset usually contains data points of low accuracy [153], and reveals errors, such as *outliers, duplicates, missing values*, and *inconsistencies*. The origin of these data quality problems is tracked back to various reasons, such as (1) Data integration from heterogeneous data sources of varying reliability [153, 223, 70, 106, 99]; (2) Knowledge-base construction and information extraction from Web sources [213, 225, 153]; and (3) Manually- or programmatically-inserted erroneous entries (e.g. into the Web-forms) [28, 89].

The above reasons cause the decline of the overall data quality [161]. Also, the above reasons motivated the research conducted in this dissertation.

1.2 Research Questions

As a response to the data quality issues mentioned above, research in data quality management proposed various data cleaning approaches, algorithms, and systems [62, 190, 91, 245, 54, 131].

Usually, these approaches were designed to detect and repair specific error types. Hence, the typical way to deal with data errors is to apply multiple data cleaning strategies, such as missing value imputation [245], outlier detection [8], values re-formatting [127], or deduplication [62]. Nevertheless, utilizing all possible data cleaning strategies simultaneously produces a large number of detection results. Subsequentially, with the ever-growing number of data cleaning algorithms and systems [62, 190, 91, 245, 54, 131, 239], there are emerging requirements for data cleaning systems that maximize the coverage of existing data quality issues. Moreover, these cleaning systems use miscellaneous types of metadata, such as value distributions, histograms, or derived integrity constraints, in

order to solve such problems. Consequentially, as many data cleaning systems incorporate metadata in their cleaning routines, the principal relationship between metadata and data quality problems needs to be established.

Furthermore, as data quality issues are interacting [82, 87], this interaction might not be obvious to the user. As a result, the optimal execution order of data cleaning rules or approaches is difficult to achieve during automatic data cleaning [62]. Hence, the next requirement for data cleaning systems emerges – allowing the declarative specification of data cleaning rules and customization of these rules without having to specify the rules execution order [62].

Guided by the research problems mentioned above, this dissertation considers three core research questions:

1. How can we accelerate the data curation process by using the dataset's metadata? To address this question, we created a template for practical data quality assessment based on metadata information, which can be used while establishing the data preparation tasks from scratch [233].

2. How can we effectively combine different error detection strategies using learning algorithms? We proposed a methodology for an effective data cleaning strategies aggregation for data error detection [232, 159].

3. How can we improve data quality through simultaneous treatment of data cleaning rules? To address this question, we developed a methodology for a declarative data cleaning approach without specifying the order of rule execution [234].

In the following, we describe the contributions of this dissertation.

1.3 Contributions

According to the research questions defined above, the contributions of this dissertation are divided into three categories:

Mapping Metadata to Data Quality Issues. Although various data cleaning solutions have been proposed so far, data cleaning remains a manual and iterative task that requires considerable domain and technical expertise. We study the intrinsic connection between metadata and data errors by establishing a mapping that reflects the connection between data quality issues and extractable metadata. We provide a new metadata classification, which is suitable for data quality management. Finally, we generalized the taxonomy of metadata by suggesting a closed grammar allowing the creation of new complex metadata. This research has resulted in the following publication:

- *Visengeriyeva, L. and Abedjan, Z. (2019). "Anatomy of Metadata for Data Curation". Under submission.* [233].

Error Detection. Since existing data cleaning solutions are usually adjusted towards one specific type of data errors, such as rule violations or outliers, it is reasonable to use a combination of error detection strategies to discover possible data errors in a dataset. Using all possible cleaning strategies is also misleading, as some data cleaning systems might perform poorly on a particular dataset by producing a large number of false positives. However, it is not trivial to assess the effectiveness of each strategy upfront. We provide two holistic approaches for effectively combining different error detection strategies. The proposed approaches are based on state-of-the-art ensemble learning algorithms and incorporate the dataset's metadata. The above research has resulted in the following publications:

- *Visengeriyeva, L. and Abedjan, Z. (2018). "Metadata-Driven Error Detection". In Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)* [232]
- *Mahdavi, M., Neutatz, F., Visengeriyeva, L., and Abedjan, Z. (2019). "Towards Automated Data Cleaning Workflows". In Proceedings of the LWDA 2019. [159].*

Joint Error Detection and Repair Suggestion. A common approach for addressing data quality issues is to formulate a set of data cleaning rules, which are intended to detect and repair incorrect data. To incorporate data cleaning rules within a single cleaning routine and to automate data curation, data cleaning systems must be able to treat data quality rules jointly, meaning, without the manual specification of the rules execution order. Therefore, we propose an approach to data cleaning based on statistical relational learning (SRL) and probabilistic inference. We argue that the Markov logic formalism is a natural fit for modelling data quality rules. This formalism allows the usage of probabilistic joint inference over interleaved data cleaning rules to detect data errors and suggest a repair. Furthermore, it eliminates the need to specify the order of rule execution. We demonstrate how data quality rules expressed as formulas in first-order logic directly translate into the predictive model in the suggested SRL framework. Moreover, we show that simultaneous rule execution automatically performed by our system outperforms a manually selected order. This research has resulted in the following publication:

- *Visengeriyeva, L., Akbik, A., and Kaul, M. (2016). "Improving Data Quality by Leveraging Statistical Relational Learning". In Proceedings of the 21st International Conference on Information Quality, ICIQ* [234].

1.4 Outline

This dissertation is structured as follows:

- In Chapter 2, we describe three pieces of critical background material for this dissertation: (1) The foundations of data quality management, such as the multi-dimensional aspect of data quality, a taxonomy of data quality issues, and the process of data cleaning, which consists of two phases, namely, error detection and error repair. (2) A description of data profiling as an initial phase for data preparation, and particularly in data cleaning, namely single-column and multi-column (dependencies) profiling tasks. (3) A description of Statistical Relational Learning, which includes, the concept of Markov logic formalism, and the translation of Markov logic programs into a probabilistic inference, as the foundation for a probabilistic data cleaning.

- In Chapter 3, we present a survey on several lines of related work to this dissertation. In total, we present 25 groups of related work organized along two dimensions: the first dimension captures common data quality issues and typical data cleaning tasks, which we found in the literature, and the second dimension reflects various data cleaning approaches.

- In Chapter 4, we conduct a systematic study of the application of metadata for detecting data quality problems. We achieve our goal by (1) creating a qualitatively validated mapping between metadata and data quality issues; (2) establishing a new taxonomy of profiling tasks that reflects the data quality issues; and (3) generalizing the notion of the granularity of the metadata in terms of composability.

- In Chapter 5, we develop an approach to error detection based on a combination of supervised machine learning methods and data profiling. We propose a technique that (1) effectively combines multiple error detection strategies; (2) considers the characteristics of the dataset for the error detection; and (3) selects the most effective data cleaning solutions for the particular dataset.

- In Chapter 6, we develop an approach to data cleaning based on Statistical Relational Learning (SRL) and probabilistic inference. We propose a method that (1) utilizes the probabilistic joint inference over interleaved data cleaning rules to improve data quality; (2) removes the need to specify the order of the rule execution; and (3) expresses data quality rules as a first-order logic formula, in order to directly translate into the predictive model in our SRL framework.

- In Chapter 7, we conclude our contribution as described in this dissertation, and present an overview of potential further research directions in the field of data quality management.

2

Preliminaries

In this chapter, we describe three areas of critical background knowledge for this dissertation:

1. In Section 2.1, we provide the foundations of data quality management, specifically, (1) the multi-dimensional aspect of data quality; (2) the taxonomy of data quality issues; and (3) the process of data cleaning, which consists of two phases, namely, error detection and error repair.

2. In Section 2.2, we describe *data profiling* as an initial phase for data preparation and in particular for data cleaning, namely single-column and multi-column (dependencies) profiling tasks.

3. A walk-through on what is the *statistical relational learning* is provided in Section 2.3, specifically, (1) the concept of Markov Logic programs, and (2) the translation of Markov Logic programs into probabilistic inference, as the foundation of probabilistic data cleaning.

The goal of this chapter is to set up the notations and definitions that are used consistently in the rest of this dissertation.

Please note that we restrict the subject of the research in this dissertation to structured data usually represented in a relational form. This means that images, video data, and semi-structured data formats such as XML or linked open data are outside the scope of this work.

Common notation. We introduce a common notation that is used throughout this work. We consider a database instance $\mathcal{D}$ with a relational schema $\mathcal{S}$. $\mathcal{D}$ contains a set of

relations $(\mathcal{R}_1, \ldots, \mathcal{R}_m)$, where relation $\mathcal{R}_i \in \mathcal{S}$ is defined for the set of attributes $attr(\mathcal{R}_i)$. $dom(U_i)$ denotes the domain of the i-th attribute with $U_i \in attr(\mathcal{R})$ and represents the set of values, which are allowed for the attribute U_i. When referring to relational databases, we use the terms *relation* $\mathcal{R}[U_1, \ldots, U_n]$, *tuple* $t[u_1, \ldots, u_n]$, *attribute* U_i, and *data value* $\mathcal{R}.v$ interchangeably with terms *table, row*, *column*, and *cell*.

2.1 Data Quality Management

As previously motivated in Section 1.1, data quality [82] is an important problem in data management. In this section, we consider three fundamental aspects of data quality, namely the multidimensional data quality definitions (Section 2.1.1), data quality issues (Section 2.1.2) and data cleaning tasks (Section 2.1.3).

2.1.1 Dimensions of Data Quality

In this section, we define the *Data Quality* by specifying multiple dimensions of this notion.

Data quality can be described with regard to multiple dimensions. The concept of *dimensions* describes data quality requirements and provides metrics to quantify the levels of data quality. These dimensions describe either instances (e.g., data values or tuples) or dataset schema. [18]. In this work, we focus on the data quality dimensions and metrics referred to instances. Data quality dimensions are measurable data quality properties that represent a particular aspect of the data, such as syntactic and semantic correctness of data, availability of data, the presence of conflicts in data, and the update of data values over time [18]. Even though the literature suggests several approaches to define data quality dimensions [147, 236, 172, 18], neither of them is ubiquitous. The contextual nature of the data quality causes variability in the definition of the data quality dimensions [18]. In this dissertation, we concentrate on data quality dimensions that have been the focus of existing data cleaning solutions [62, 91, 228, 55, 144, 127, 245, 193, 145, 202, 112]: *Consistency, Accuracy, Completeness, Uniqueness* and *Timeliness* [82]. In the following, we provide the definitions for each of these dimensions. Please note that these definitions were aligned along with the definitions provided by Fan W. et al. [82] and Batini C. et al. [18].

Definition 2.1.1 *The* ***Consistency*** *dimension refers to the validity and integrity of values and tuples with respect to defined inter- and intra-relational constraints that exist within either single or multiple relations.* ■

For instance, the violation of the consistency dimension captures integrity constraints such as *functional dependency* [82] and other semantic rule [228] violations.

Definition 2.1.2 ***Accuracy*** *is defined as proximity of a data value $\mathcal{R}.v$ to the correct value of the real-world entity $\mathcal{R}.v'$ The accuracy dimension identifies correct and true values of the entities presented by data.*■

Practically, accuracy is "measured as the distance between the value stored in the database and the correct one" [18].

Definition 2.1.3 *The* ***Completeness*** *dimension determines whether the database provides the complete information to correctly answer the database queries by providing all required attribute values of an entity's description.* ■

An alternative definition is provided by Redman T. et al. [200]: "*completeness* is a degree to which values are included in a data collection". Completeness is described with regard to two aspects: first, the presence and interpretation of *null* values; second, whether the relational instance is connected to the *open world assumption* or *closed world assumption* logical model. Computationally, completeness is defined as the ratio of not *null* attribute values to the total number of attribute values in the relation $\mathcal{R}$.

Definition 2.1.4 *The* ***Uniqueness*** *dimension measures whether the same real-world entity is represented by multiple data points in one or more relations.* ■

The uniqueness dimension is reflected by *duplicate detection* [82], which is also known as entity resolution, record matching, object identification, record linkage, duplicate detection, and merge-purge.

In the literature, the time-related dimension is interchangeably referred to as *Currency, Volatility*, and *Timeliness* [18, 82, 172].

Definition 2.1.5 *The* ***Timeliness*** *dimension reflects the change and update of data by identifying the most current value of an entity in a database (usually in the absence of timestamps).* ■

The opposite to the *current* or *up-to-date* values is the notion of *stale* values. We define a value $\mathcal{R}.v$ as *stale* if it is incorrect at the time τ but was in the *current* state before τ.

2.1.2 Data Quality Problems

In this section, we identify and structure the problems that disturb the data quality.

Given the core data quality dimensions, the violation of *Accuracy, Consistency, Uniqueness, Completeness* and *Timeliness* lead to *data quality issues*. Please note that we use the notions of *dirty data*, *data quality problems*, or *data quality issues* interchangeably throughout this dissertation. To improve data quality, we initially need to understand the nature of dirty data. An example of dirty data is visualized in Figure 2.1. Errors in

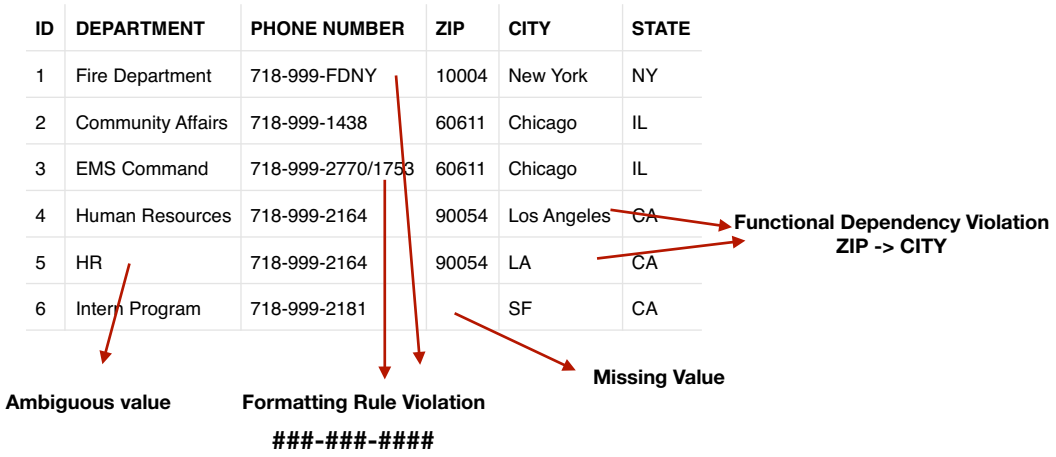

Figure 2.1: An example of multiple data errors that are present in the dataset.

the dataset might be introduced throughout any stages of the data's lifecycle, including capture, persistence, update, transmission, restore, and deletion [133]. In the following, we list most frequent reasons for data errors.

1. *Data integration* is the task of generating a unified view of data originated from heterogeneous data sources [223, 70, 106, 99]. Practically, the following data quality problems might emerge during the data integration process: (1) *instance-level conflict resolution*, that is, identifying and resolving conflicts among values that point to the same real-world entity; and (2) *quality-driven query processing*, which involves providing query results by considering the quality of the data sources [18];

2. *Information extraction*, when Web and linguistic data are transposed into the structured form [213, 225]. Typical data quality issues arising during the process of data extraction are incorrect values or duplicate entries [153];

3. Erroneous entries that are *manually* (e.g. filling the Web-forms) or *programmatically* inserted [28, 89], which result in inaccurate, missing or disguised values.

In the following, we provide a taxonomy of data quality issues, which are mapped to the violated data quality dimensions, as shown in Table 2.1. The taxonomy and the mapping are adopted from Laranjeiro N. et al. [147] and Kim W. et al. [133].

Table 2.1: Violations of data quality dimensions. Source [233].

Data Quality Issue	Description	Data Quality Dimensions				
		Accuracy	Consistency	Uniqueness	Completeness	Timeliness
Missing data [133, 170]	Refers to both missing tuples and missing values. Tuple completeness requires that all tuples are present in the table. Missing value issue consists of either *null*-values or *disguised* values [188]. Value completeness requires that all values are present in the table, while *null*-values indicate that the value is unknown or nonexistent. Additionally, the *disguised* values represent default values that are legitimate values which do not provide the complete information about the real-world entity.	✓			✓	
Incorrect data [133]	Refers to the data that differs from the values of the real-world entity (e.g., birth year "2010" instead of "2011")	✓				
Misspellings [195]	Refers to the syntactic violation of the data value (e.g., entering the last name "Smiht" instead of "Smith")	✓				
Ambiguous data [195, 133]	Refers to the data values that might be interpreted in several ways (e.g., abbreviations, cryptic values such as "NLP" or "A.")	✓	✓			
Extraneous data [147]	Refers to the presence of additional data in the attribute value (e.g., the value in the *address* column contains a person's name and address information).		✓	✓		
Outdated temporal data [133]	Refers to the temporal values that are obsolete, meaning they do not represent the real state of the entity.					✓
Misfielded values [195, 147]	Refers to values which belong to a different attribute (e.g., values from the attribute "US State" are stored in the "US City" field).	✓	✓		✓	
Incorrect references [147]	Refers to the entities that contain wrong information related to the reference relation (e.g., the employee is associated with a wrong department).	✓				
Duplicates [195, 133]	Refers to the tuples/values that represent the same real-world entity.			✓		
Structural conflicts [195]	Refers to the duplicate entities in different sources.		✓	✓		

Table 2.1 (cont.) Violations of data quality dimensions.

Data Quality Issue	Description	Data Quality Dimensions				
		Accuracy	Consistency	Uniqueness	Completeness	Timeliness
Different word orderings [133]	Refers to the values that violate the expected pattern such as first name and second name instead of the reverse ordering.		✓	✓		
Different aggregation levels [147]	Refers to the entities in multiple sources that have been produced by applying different aggregation methods (e.g., entries per quartal vs. entries per year).	✓	✓			
Temporal mismatch [133]	Refers to the values that point to different time slots	✓				✓
Different units/representations	Refers to the values from multiple sources that represent different units (e.g., the currency attributes might refer to Dollar or Euro in multiple sources).		✓			
Domain violation [195, 133]	Refers to the *illegal* values [195] that violate semantic rules defined on the particular attribute	✓				
FD violation [170, 80]	Refers to column values that violate previously specified functional dependencies	✓	✓			
Wrong data type [195]	Refers to the values that violate syntactic specification of the corresponding attribute. Alternatively, this data problem refers to the data type constraint violation.		✓			
Referential integrity violation [195, 82]	Refers to the tuple values that violate the referential integrity constraints which are defined on multiple relations (e.g., one entity has no reference in the other entity).	✓	✓		✓	
Uniqueness violation [4]	Refers to values that are either *missing* or violate the uniqueness constraint.			✓		
Use of synonyms [147]	Refers to the values in different sources that have the same semantic meaning but which are syntactically different (e.g., "lecturer" and "professor" are different representation for the same data).			✓		
Use of special characters (space, no space, dash, parenthesis) [133]	Refers to the different representations of compound data, such as social security number or phone number		✓			
Different encoding formats [133, 195]	Refers to values that have been produced by using a special algorithmic transformation (e.g., ASCII or EBCDIC).		✓			

2.1.3 Data Cleaning

Taking the data quality issues into account, in this section, we explain the constituent tasks of data cleaning. In particular, the data cleaning process consists of the error detection task (Section 2.1.3.1) as the first step in the data preparation process, and the error correction task (Section 2.1.3.2) as the second one.

Generally, data cleaning attempts to identify inconsistencies in a dataset and eventually repair such inconsistencies so that the data consistently, accurately, completely, timely and uniquely represents the entities to which they refer. Data cleaning can be seen as one component of the *data preparation* process in the data science workflow, along with data profiling and data integration [4].

2.1.3.1 Error Detection

Given that real-life data is dirty [153], the next step is to effectively localize errors in the dataset by using rules and additional error detection techniques.

Definition 2.1.6 *A **Data Quality Rule** is a data quality constraint, which declaratively expresses quality conditions that a database instance has to satisfy.* ■

A typical example of data quality rules are edits [89] or integrity constraints-based data quality rules [80, 57], such as functional dependencies [83] (see Section 2.2).

Definition 2.1.7 *Given a set of data quality rules $\Lambda = (\lambda_1 \dots \lambda_n)$ and a relation $\mathcal{R}$, **Error Detection** is the process of detecting the violation of data quality dimensions by analyzing $\mathcal{R}$ to what extend the relation $\mathcal{R}$ conforms to the existing data quality rules Λ. Records that are inconsistent with at least one of the rule $\lambda_i \in \Lambda$ are declared **erroneous**.* ■

According to Fan W. et al. [82], the error detection process is usually performed by (1) detecting irregularities in $\mathcal{R}$, i.e., identification of all tuples in $\mathcal{R}$ that violate some rule in Λ; (2) deciding whether $\mathcal{R}$ has complete information to answer queries; and (3) determining whether $\mathcal{R}$ has current information.

Generally, we distinguish several error detection techniques, such as *rule-based error detection*, *outlier detection* [8, 190], and *pattern violation detection* [127].

As we will see later in Section 2.2.2, integrity constraints define the semantics of data, and all five data quality dimensions can be specified by using these constraints [80]. Hence, data quality rules can be declared on such integrity constraints that have been discovered from the data. Another method involves specifying the set of rules that express the condition to identify an error in the dataset. Such rules originate from statistics and are called *edits* [89, 38]. An example of such *edit* rule might be the following: "there is an error if marital status is `married` and `age < 14`, and the edit that formulates the error condition,

is `(marital status = married)` $\wedge$ `(age < 14)"` [38]. The set of edits is required to be consistent and non-redundant. All tuples in $\mathcal{R}$ that are inconsistent with at least one of the edit rule are declared *erroneous*.

In addition to the rule-based error detection, we also perceive *outlier detection* [8, 190] and *pattern violation detection* [127] as error detection techniques. In particular, the outlier detection identifies data points that significantly deviate from the distribution or structure of the remaining values. For example, a negative value in the *age* attribute would be an outlier. The pattern violation detection captures errors through validation of syntactic or semantic patterns. For instance, for the *state* attribute, one can specify a regex pattern rule: $[A-Z]\{2\}$, which denotes that *"the value in the column "US State" should be an abbreviation represented by two upper case characters"*.

2.1.3.2 Error Repair

Inconsistent data in the database, meaning that data is contradicting integrity constraints or certain semantic rules, is an undesirable state of the database instance. Therefore, after performing the error detection step, the next phase in data quality is to correct the inconsistencies [13, 24].

To define the *error repair*, we first define the allowed dataset instances that represent the *corrected* or *cleaned* version of the database by specifying the notion of the *distance* between the database instances. One possible way to determine such distances is to use the partial order [13]:

Definition 2.1.8 ***Distance between database instances*** *[24]. Let $\mathcal{R}$ be a fixed instance for a relational schema S.*

(a) For two instances $\mathcal{R}_1$ and $\mathcal{R}_2$ for S, we say that $\mathcal{R}_1$ is at least as close to $\mathcal{R}$ as $\mathcal{R}_2$, denoted as $\mathcal{R}_1 \preceq_{\mathcal{R}} \mathcal{R}_2$, iff $\Delta(\mathcal{R}, \mathcal{R}_1) \subseteq \Delta(\mathcal{R}, \mathcal{R}_2)$. Here, $\Delta(\mathcal{D}_1, \mathcal{D}_2) := (\mathcal{D}_1 \setminus \mathcal{D}_2) \bigcup (\mathcal{D}_2 \setminus \mathcal{D}_1)$ is the symmetric set difference between two sets.

(b) $\mathcal{R}_1 \prec_{\mathcal{R}} \mathcal{R}_2$ holds iff $\mathcal{R}_1 \preceq_{\mathcal{R}} \mathcal{R}_2$, but not $\mathcal{R}_2 \preceq_{\mathcal{R}} \mathcal{R}_1$. ■

To perform the *data repair* task, we might need to obtain the $\preceq_{\mathcal{R}}$-minimal instances $\mathcal{R}$, which are closest to $\mathcal{R}$ and consistent with regard to either previously defined data quality rules or other quality metrics, such as statistical metrics. The elements in the set $\Delta(\mathcal{R}, \mathcal{R}')$ can be seen as a result of transformation operations such as deletions of tuples or updates of tuple values in the $\mathcal{R}$ dataset.

Definition 2.1.9 *Given a set of data quality metrics $\Lambda = (\lambda_1 \dots \lambda_n)$ and a relation $\mathcal{R}$ of schema $\mathcal{S}$,* ***Error Repair*** *is the process of the transformation of $\mathcal{R}$ into $\mathcal{R}'$, so that:*

(a) $\mathcal{R}'$ satisfies data quality metrics Λ, i.e., $\mathcal{R}' \models \Lambda$.

(b) $\mathcal{R}'$ *is* $\preceq_{\mathcal{R}}$*-minimal in the class of instances for* $\mathcal{S}$ *that satisfy* Λ*.* ■

Generally, the *minimality* means that $\mathcal{R}'$ is as close to the original dataset $\mathcal{R}$ as possible [52, 136]. Provided with two candidate sets of repairs, the one with fewer transformation operations is preferred. Hence, the repair $\mathcal{R}'$ is a new instance that is obtained from $\mathcal{R}$ and represents consistent data with respect to the data quality requirements, which are expressed as data quality rules Λ. In this way, a repair $\mathcal{R}'$ is considered as a "clean" version of the original instance $\mathcal{R}$. Informally, cleaning means identifying and correcting dataset violations while keeping the cleaned instance as close as possible to $\mathcal{R}$ [33, 24].

The notion of minimality is widely used to restrict the complexity of the search space, while deciding what $\mathcal{R}'$ instance should be selected as a *repair* instance for $\mathcal{R}$. However, recent research increasingly claims the limitation of the *minimal data repair* [68]. Concretely, the data cleaning approaches that are based on the minimality principle do not provide the likelihood of possible repairs [68, 202, 234]. We address the problem of minimality repair in Chapter 6, where we propose to utilize the probabilistic joint inference over interleaved data cleaning rules to detect errors and provide possible repairs.

2.2 Data Profiling

To begin the data quality assessment, we must have a good understanding of the data. This section provides a description of data profiling as a process of metadata discovery [4]. This process incorporates a set of tasks that creates metadata to deliver, as much as possible, a full understanding of the underlying dataset. In Section 2.2.1, we describe elementary profiling tasks performed on individual columns of the dataset. In Section 2.2.2, we provide an overview of profiling activities executed on multiple columns.

Next, we list the definitions of *metadata* and *data profiling* as provided in the literature.

Definition 2.2.1 ***Metadata*** *is "structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource" [207].* ■

Definition 2.2.2 ***Data Profiling*** *is a process for discovering metadata that includes "various experimental techniques aimed at examining the data and understanding its actual structure and dependencies" [163].* ■

In data preparation, data profiling usually precedes data cleaning activities. Even though data profiling has many use cases, including data exploration, database reverse engineering, data integration, data analysis, and query optimization, we focus on profiling data for the data quality use case. Data quality assessment literature has identified the following four types of profiling tasks to obtain metadata [163]:

1. **Attribute profiling** creates distributions and patterns associated with attributes by using descriptive statistics.
2. **Dependency profiling** identifies the inter-column relationships, e.g. (conditional) functional dependencies, matching dependencies etc.
3. **Relationship profiling** identifies entity keys and various relationships in the data model, e.g. inclusion dependencies.
4. **State-transition model profiling** determines the cycle of level- or state-dependent objects as it appears in the data. For a person's "status", the level could be "single," "married," or "divorced."

Given the set of metadata produced by profiling methods, Abedjan Z. et al. [4] provide a comprehensive taxonomy of metadata. Their classification comprises *single-column analysis* and *dataset dependencies*. Typical metadata of single columns includes cardinalities, patterns, data types, value distributions, histograms, domain classifications, summaries, and sketches. The second category of profiling tasks groups the dependency related metadata, such as (conditional) functional dependencies, (conditional) inclusion dependencies [82, 80], and unique column combinations [5]. In the following, we review *single-column analysis* and *dataset dependencies* categories.

2.2.1 Single-Column Profiling Tasks

In this section, we review elementary profiling tasks performed on individual columns of the dataset.

The essential data profiling activities provide the analysis of single columns of the given database table. Basically, single-column metadata includes counts, frequent values, basic aggregate statistics, and value distribution for each attribute. In the following, we will outline the most relevant profiling tasks that operate on individual columns [4]:
Cardinalities refers to the counts of values. This category includes:

1. *Number of rows*: the number of entities which are available in the table;
2. *Distinctness*: the number of distinct values of the single attribute;
3. *Uniqueness*: the ratio of the number of distinct values to the number of rows.

Value Distribution refers to the distribution of values on the column. This category includes:

1. *Constancy*: the ratio between the most frequent value count and the number of rows;
2. *Extreme values*: minimum and maximum values in numeric columns; shortest and longest strings in categorical, alphanumeric or text columns;

3. *Histogram*: values distribution summary on an attribute, which is constructed by grouping attribute values $\mathcal{V}$ into disjoint *buckets* (equi-depth or equi-height) and computing frequencies $\mathcal{F}$ for each bucket [122]. Histograms, considered as mathematical objects, approximate data distribution [123] and technically are pairs (v_i, f_{v_i}), where $v_i \in \mathcal{V}$ is i-th value of the value set $\mathcal{V}$ and $f_{v_i} \in \mathcal{F}$ denotes the frequency of the value v_i;

4. *Quartiles*: three points that divide numeric distribution into four equal groups;

5. *Inverse distribution*: an inverse frequency distribution (a distribution of the frequency distribution);

6. *Various distributions:* i.e., a frequency distribution of the value or cell *n-gram*, Soundex code, pattern or value length.

Data Types refers to the information about the data type. This category include:

1. *Basic type* describes the basic data type, such as numeric, alphanumeric, or date/time;

2. *Data type* includes more specific data type description that is mostly available by database management systems, for example, integer, float, boolean, varchar etc.

Patterns refers to the syntactic properties on the values of the individual column. This category includes:

1. *Lengths*, which specifies the descriptive statistics of the column value lengths, such as minimum, maximum, median and average lengths;

2. *Size*, which determines the number of digits in numeric columns;

3. *Decimals*, which determines the number of decimals in numeric columns;

4. *Patterns*, which creates the histogram of value patterns (e.g. regular expressions [197] or syntactic patterns).

Domains refers to the semantic description of the column's domain. This category includes:

1. *Data class*, which describes the generic semantic of the column, such as date or time, identifier, code or text;

2. *Domain*, which represents the semantic of the column's values, such as address information, geographical locations, phone numbers or personal information [127];

Data Completeness refers to the missing values in the column values. This category includes:

1. *Null values*, which is the count of absent values within the individual column;

2. *Default values*, which represent valid values but implicitly denote *"missing values"* [188].

The above-described single-column profiling tasks provide syntactic and semantic information about each individual column. The second category of profiling tasks comprises the dependency-related metadata, such as (conditional) functional dependencies, (conditional) inclusion dependencies [82] and unique column combinations.

2.2.2 Dependencies: Multi-Column Profiling Tasks

In this section, we give an overview of profiling activities executed on multiple columns.

Dependencies are defined as metadata that describe relationships between attributes $\mathcal{U}$ in the relation $\mathcal{R}$ [4]. There are many forms of dependencies, which are the goal of the multi-column data profiling: *unique column combinations* [5], *functional dependencies* [57, 82] and their approximate variant *conditional functional dependencies* [80], *inclusion dependencies* [44], *matching dependencies* [80], and the generalization of multi-column dependencies - *denial constraints* [24]. The above-mentioned dependencies are mainly used in the database design, for example, in to normalize schema, optimize queries, or to constrain the illegal updates [80]. Our goal is to introduce multi-column dependencies that are relevant for improving the quality of data.

Definition 2.2.3 *A **Functional Dependency** (FD) is an expression of the form $X \to Y$ where $X \subseteq \mathcal{U}$ and $Y \subseteq \mathcal{U}$ are subsets of $\mathcal{R}'s$ attributes. This FD holds if every pair of tuples of $\mathcal{R}$ that agree in each of the X attributes, also agree in the Y attributes.* ■

For example, consider the relational schema $\mathcal{U} = (X, Y, Z)$; the FD $X \to Y$ defined for this schema determines that for any two tuples $t_1[x_1, y_1, z_1], t_2[x_2, y_2, z_2]$, if $t_1[x_1] = t_2[x_2]$, then $t_1[y_1]$ and $t_2[y_2]$. In other words, the attribute X uniquely determines the attribute Y [4].

Definition 2.2.4 *A **Conditional Functional Dependency** (CFD) defined on the relational schema $\mathcal{R}$ is a pair $\mathcal{R}(X \to Y,\ T_p)$, where*

1. *$X \to Y$ is a standard FD;*

2. *T_p is a set of constraints holding on the particular subset of tuples.* ■

For example, consider the following CFD: $([x = AAA, z] \to [y])$ asserts that for any tuple in this relation, if the x-attribute value is AAA, then z uniquely determines y. This CFD uses the traditional FD $X, Z \to Y$ that satisfies the pattern $x = AAA$ [82].

Definition 2.2.5 ***Denial Constraints*** *(DC) are universally quantified first-order logic expressions of the form:*

$$\forall X_1 \ldots \forall X_{|\mathcal{S}|} \quad \neg(X_1 \wedge \cdots \wedge X_{|\mathcal{S}|} \wedge \xi(X_1 \ldots X_{|\mathcal{S}|}))$$

where $(X_1 \ldots X_{|\mathcal{S}|}) \subseteq \mathcal{U}$ *are attributes of the relational schema* $\mathcal{R}$*, and* $\xi(X_1 \ldots X_{|\mathcal{S}|})$ *is a conjunction of built-in predicates such as* $<, >, \leq, =, \neq$*.* ■

Functional dependencies are a special form of denial constraints [24]. For instance, the functional dependency $X \rightarrow Y$ on the relational schema $\mathcal{R}$ can be written in a different syntactic form, namely as a denial constraint: $\forall x \forall y \forall z \neg(\mathcal{R}(x, y) \wedge \mathcal{R}(x, z) \wedge y \neq z)$.

Definition 2.2.6 *A* ***Matching Dependency*** *(MD) for schemas* $\mathcal{R}_1$ *and* $\mathcal{R}_2$ *is syntactically defined as:*

$$\mathcal{R}_1[X_1] \approx \mathcal{R}_2[X_2] \rightarrow R_1[Y_1] \rightleftharpoons \mathcal{R}_2[Y_2]$$

where X_1 *and* X_2 *are pairwise compatible sets of attributes in* $\mathcal{R}_1$ *and* $\mathcal{R}_2$*, respectively; similarly for* Y_1 *and* Y_2*;* $\approx$ *indicates similar attributes and* $\rightleftharpoons$ *is called the matching operator.* ■

In other words, the matching operator $\rightleftharpoons$ means that for each $\mathcal{R}_1$ tuple t_1 and each $\mathcal{R}_2$ tuple t_2: $t_1[Y_1]$ and $t_2[Y_2]$ refer to the same real-world entity. Having dynamic semantics, MDs force the update of $t_1[Y_1]$ and $t_2[Y_2]$ so that they have the same values.

For example, consider the FD defined on schema $\mathcal{R}(A, B, C)$ $fd_1 : A \rightarrow B$. Also, consider two tuples $s_1(a, b_1, c_1)$ and $s_2(a, b_2, c_2)$ of this schema. These tuples obviously violate the FD fd_1. On the other hand, based on fd_1, the MD $md_1 : \mathcal{R}[A] = \mathcal{R}[A] \rightarrow \mathcal{R}[B] \rightleftharpoons \mathcal{R}[B]$ states that for any pair of tuples (s_1, s_2), if $s_1[A] = s_2[A]$, then $s_1[B]$ and $s_2[B]$ should be identified and have the same value.

To capture errors across multiple relations, we need an interrelation constraint, in the form of *inclusion dependencies* [44].

Definition 2.2.7 *An* ***Inclusion Dependency*** *(IC) over the relation schemas* $\mathcal{R}_1$ *and* $\mathcal{R}_2$ *is syntactically defined as:*

$$\mathcal{R}_1[X] \subseteq \mathcal{R}_2[Y].$$

Given two relations r_1 *and* r_2 *of schemas* $\mathcal{R}_1$ *and* $\mathcal{R}_2$*, respectively, the inclusion dependency* $r_1[X] \subseteq r_2[Y]$*, where* X *and* Y *are lists of attributes of* $\mathcal{R}_1$ *and* $\mathcal{R}_2$*, respectively, states that all values in* X *also occur in* Y*. Formally:*

$$\forall t_i \in r_1, \exists t_j \in r_2 : t_i[X] = t_j[Y].$$ ■

For example, consider the following inclusion dependency on two relations: $\mathcal{R}(x, y) \subseteq D(z, y)$, which means that for every tuple $t_i[x, y] \in \mathcal{R}$, there exists a tuple $t_j[z, y] \in D$, so that $t_i[x, y] = t_j[z, y]$.

Identifying potential key columns is an important data profiling activity. A unique column combination (UCC) describes the key dependencies of a relation [5].

Definition 2.2.8 *A **Unique Column Combination** is a set of attributes $X \subseteq \mathcal{U}$ whose projection contains no duplicate tuples. Formally,*

$$\forall t_1, t_2 \in \mathcal{R} : t_1[X] \neq t_2[X]. \blacksquare$$

Each unique column combination identifies a syntactically valid relation key. Furthermore, functional dependencies are generalizations of unique column combinations, since $X \rightarrow \mathcal{U} \setminus X$.

2.3 Statistical Relational Learning

We also propose using statistical relational learning (SRL) to advance data cleaning. This section provides the necessary background knowledge about the concepts of SRL. Being an intersection between machine learning and artificial intelligence, SRL aims to represent, reason, and learn in domains with relational and uncertain structure [93]. SRL unifies relational or first-order logic with probabilistic and statistical approaches to inference and learning [130]. The complex relational domains are represented by either first-order logic or frame-based formalisms [61]. The probabilistic semantic is usually based on probabilistic graphical models or stochastic grammars [137]. A large number of SRL approaches have been proposed, including Markov Logic [73], Relational Markov Models [11], Inductive Logic Programming [130], and Relational Dependency Networks [176].

In this dissertation, we focus on Markov logic [73] as an SRL formalism for representation of our data cleaning systems. We describe the syntactic and semantic meaning of Markov logic in Section 2.3.1. The inference functionality of Markov logic is provided in Section 2.3.2.

2.3.1 Markov Logic

We propose to use Markov logic [73] as our representation language for probabilistic data cleaning framework (see Chapter 5). Markov logic is a formalism for probabilistic extension of first-order logic [203]. The advantage of first-order logic lies in the precise specification of relations among diverse objects. A set of first-order formulae constitute a *first-order knowledge base (KB)*. The formulae in KB are conjoined. Therefore a KB is a combination of all defined formulae. Nevertheless, if one first-order logic formula in the knowledge base is *false* (unsatisfiable), the complete knowledge base becomes unsatisfiable. This is because the knowledge base is the conjunction of the first-order logic formulae, and interpreting at least one formula as *false* leads to interpreting the whole knowledge base as *false* [92]. Markov logic formalism overcomes such brittleness by adding weights to first-order logic

formulae, such that they compile to a probability distribution as an exponential (log-linear) model [175]. In the following, we provide some terms used in Markov logic.

Objects in a complex domain are represented by using a *term* that is a variable or a constant, such as `Bob, x, y, apple`. An *atomic predicate (atom)* is a predicate symbol applied to a tuple of terms and denotes a relation, such as `smoke(Bob), friends(x,y)`. First-order logic formulae are specified using atoms connected by logical connectives $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$ and logical universal and existential quantifiers: $\forall$ and $\exists$ [92].

The following two formulae [73] are common examples of Markov logic:

$$\forall \mathtt{x}\ \mathtt{smoke(x)} \ \Rightarrow\ \mathtt{cancer(x)} \tag{2.1}$$

$$\forall \mathtt{x} \forall \mathtt{y}\ \mathtt{friends(x,\ y)} \ \Rightarrow\ (\mathtt{smoke(x)} \ \Leftrightarrow\ \mathtt{smoke(y)}) \tag{2.2}$$

The first formula 2.1 denotes that smoking causes cancer. The second formula 2.2 says that if two people are friends, either both smokes or neither does.

A term containing no variables is called *a ground term*. An atom or predicate whose arguments are ground terms is called a *ground atom* or *ground predicate*, respectively. Furthermore, a ground predicate whose state is known (i.e., true or false), is called an *observed predicate*. *Hidden predicates* are ground predicates with unknown states. If a formula contains only ground atoms, then it is called *a ground formula*.

Probabilistic graphical models are popular formalisms for probabilistic modelling because they concisely represent probability distributions by exploiting conditional independence [137]. A *Markov network* [187], is an undirected graphical model that specifies a joint distribution over a set of random variables $X = (X_1, \ldots, X_m)$. The *Markov network* is given by the formula:

$$P(X = \mathtt{x}) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}}), \tag{2.3}$$

where $x_{\{k\}}$ denotes the state of the variables in the k-th clique and $\phi_k(x_{\{k\}})$ is a potential function for the k-th clique. Z is a *partition function*, so technically, it is a normalization constant given by $Z = \sum_{x \in \mathcal{X}} \prod_k \phi_k(x_{\{k\}})$.

Markov networks are often represented as *log-linear models* [175] by replacing the $x_{\{k\}}$-clique with an exponentiated weighted sum of features of the state:

$$P(X = \mathtt{x}) = \frac{1}{Z} \exp\left(\sum_j w_j f_j(x)\right), \tag{2.4}$$

where Z is the normalization constant as in Markov networks (see equation 2.3). A feature $f(x)$ may be any real-valued function. However, in this dissertation, we only focus on binary functions $f_j(x) \in \{0, 1\}$. The *weight* w_j of a feature $f_j(x)$ is the weight of the

first-order formula that originated it. Generally, formula 2.4 denotes the probability of a state x that depends on the specified weights. We have to note that by setting weights to infinite, the formula becomes a logical constraint. Formally:

Definition 2.3.1 ***Markov logic network*** *(MLN) [73] L is defined as a set of pairs (ϕ_i, w_i), where ϕ_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = c_1, c_2, \ldots, c_{|C|}$, it defines a Markov network ML, C as follows:*

1. *ML, C contains one binary node for each possible grounding of each predicate appearing in L. The value of the node is 1 if the ground predicate is true, and 0 otherwise.*

2. *ML, C contains one feature for each possible grounding of each formula ϕ_i in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the w_i associated with ϕ_i in L.* ■

The syntax of the formulae is the standard syntax of first-order logic. All unquantified variables are translated as universally quantified. Semantically, an MLN maps each possible world to a probability score [204]. Generally, an MLN is a template for the construction of *Markov networks* if a set of constraints is provided.

2.3.2 Probabilistic Inference

In this section, we lay the foundations of the probabilistic inference, as the engine for probabilistic data cleaning.

Besides being a compact representation for joint probability distributions, constructing MLNs also provides an inference functionality that is computing answers to a query (i.e., the most probable random variables assignment). In particular, *Maximum A Posteriori (MAP)* [76] inference computes the most probable state of the world, given the evidence. Marginal inference is used to compute the conditional probability for a formula to be *true*. Inference in Markov networks is $\#P$-complete [208]. Thus, a common way to perform inference is to use approximate inference approaches, such as Markov Chain Monte Carlo (MCMC) [94] for marginal inference. Another popular method for inference is *belief propagation* [248] or reducing the MAP problem to a weighted SAT problem, and solving it with an SAT-solver such as MaxWalkSAT [129]. Recently developed methods to solve MAP problems actively rely on Integer Linear Programming (ILP) [209], because of IPL's declarative nature, exactness and availability of effective ILP solvers. Generally, it is proven that any Markov Network can be translated into an integer linear program [227, 204]. Markov logic applications solve the MAP problem of determining the most probable state of hidden predicates H, given some observed ground predicates O, with $H = P \backslash O$, where P denotes the set of all predicates:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}_{H,C}} p(y|x) = \arg\max_{y \in \mathcal{Y}_{H,C}} s(y, x), \tag{2.5}$$

where $s(y, x)$ is a *scoring function* that evaluates the score of a problem solution pair (y, x)

$$s(y, x) = \sum_{(\phi, w) \in M} w \sum_{c \in C^{n_\phi}} f_c^\phi(x, y). \tag{2.6}$$

Each feature $f_c^\phi(x, y) \in 0, 1$ returns 1 if the contained ground formula ϕ is true in the possible world, otherwise it is 0.

The generic mapping from MLN to ILP replaces each feature function $f_c^\phi(x, y)$ in equation 2.6 with the binary variable λ_c^ϕ, which leads to the formulation of the optimization problem:

$$\begin{aligned} \arg\max_{y \in \mathcal{Y}_{H,C}} \quad & \sum_{(\phi, w) \in M} w \sum_{c \in C^{n_\phi}} \lambda_c^\phi \\ \text{subject to} \quad & \forall (\phi, w) \in L,\ c \in C^{n_\phi} \\ & \lambda_c^\phi = f_c^\phi(x, y) \end{aligned} \tag{2.7}$$

Given the above optimization problem, the ILP formulation is achieved through the transformation of each constraint into a set of linear constraints, by adhering to the following steps [204]:

1. Each constraint is mapped to a logical equivalence of a ground formula.
2. Observed ground atoms are replaced by their respective state value (*true* or *false*), otherwise, they are not observed and are consequently replaced by *false*.
3. Each first-order formula is replaced by its Conjunctive Normal Form (CNF). For instance, formula 2.1 is transformed as
   `smoke(x)` $\Rightarrow$ `cancer(x)` $\equiv$ `smoke(x)` $\vee$ $\neg$`cancer(x)`.
4. Each disjunction is replaced by a linear constraint [241].

The *Cutting Plane Inference* algorithm solves the above constrained optimization problem by searching for feature-weight products in equation 2.6 that do not maximally increase the overall sum given the current solution [204].

The above described Markov logic formalism is a foundation for our approach to data cleaning, which we propose in Chapter 6.

In the following chapter, we outline the related work to this dissertation from the area of error detection and repair. We present a new classification of existing data cleaning

solutions, which are categorized along with the two aspects of data cleaning: *what to clean* and *how to clean*.

3

Related Work

The related work is organized along two dimensions:

1. **What.** The first dimension captures common data quality issues and typical data cleaning tasks, which had been found in the literature.

2. **How.** The second dimension reflects differently focused data cleaning approaches.

In the following, we briefly explain the methodology behind the structure of the related work. The first dimension includes standard data cleaning tasks regarding data quality issues detection, as well as error repair. The *what*-dimension consists of the following categories that reflect the five central data quality dimension violations:

Completeness violation. This category describes research conducted in the area of missing value detection and imputation.

Accuracy violation. This category explains different techniques for the detection and repair of inaccuracies.

Uniqueness violation. This group of approaches deals with duplicate detection and resolution.

Consistency violation. This class of methods identifies integrity constraints violation and provide solutions for data repair.

Timeliness violation. This category depicts research regarding stale data detection and resolution. Due to a large number of research conducted on timestamp attributes [104], we restrict our survey on approaches that address timeliness violation with no explicit

timestamps attributes in the dataset under assessment. For instance, when values in the *salaries* attribute are constrained to be increasing, for each person, we consider the highest value in this attribute to be the most current one.

A more granular description of the data quality issues and their violation of data quality dimensions is provided in Table 2.1. Generally, existing research distinguishes between two main groups of data cleaning approaches, namely rule-based and statistical approaches [148, 78, 104, 53]. However, there are complimentary groups of data cleaning techniques, which are rarely included in the data cleaning surveys. Therefore we present a comprehensive overview of existing systems. In this chapter, we use the second *how*-dimension and classify the related work as follows:

Rule-based approaches are also known as qualitative techniques. This group of data cleaning methods is characterized by the use of data cleaning rules or integrity constraints for detection and repair of various error types in the dataset.

Statistical approaches are also referred to as quantitative techniques. This group of data cleaning systems employs a set of statistical methods to detect data errors and impute the repair. The main motivation behind using statistics-based techniques is that for some datasets (IoT, GPS, sensor reading), no integrity constraints could be declared. For this reason, the rule-based approaches for error detection and repair are not applicable.

Hybrid approaches. This group includes research prototypes, which combine the previous two groups, namely qualitative and quantitative data cleaning methods.

Probabilistic and ML-based approaches operate by using various machine learning [171] or probabilistic algorithms [31] to predict errors in data and determine the most probable repair.

Interactive Data Cleaning. This group of data cleaning methods distinguishes from other groups by including the human factor (human-in-the-loop) in the error detection and cleaning process. We review various methods that use active learning approach, crowdsourcing, or any interactivity in the data cleaning procedure.

Please note that in this chapter, we do not consider complementary areas of data cleaning, such as data fusion [30], truth discovery [154], and data diagnosis [244, 240]. Furthermore, despite the availability of numerous data cleaning surveys [148, 78, 104, 53], we collect all available techniques for data cleaning and create a comprehensive classification of existing data cleaning solutions, which are categorized along with the two aspects of data cleaning: *what to clean* and *how to clean*.

3.1 Rule-Based Approaches

The first group of data cleaning methods is characterized by using data cleaning rules or integrity constraints to detect and repair various error types in the dataset. Integrity constraints are generally used as a foundation for data quality rules [82]. These systems require the specification of denial constraints, as well as functional and matching dependencies, to formulate data quality rules [54, 91, 62]. In Chapter 6, we provide a method for defining data quality rules by using integrity constraints.

To impute missing values in the relational data, the NADEEF system and its successor [62, 132] utilize functional and matching dependencies, and enable user-defined logic for data repair. Another approach [219] employs differential dependencies to explore similarity neighbours to fill the missing cells in the dataset.

Inaccuracy detection and repair are also tackled with data cleaning rules, because accuracy violation might be detected with various integrity constraints such as functional dependencies [91, 62, 132], matching dependencies [62, 132, 54], and denial constraints, as a generalization form for both former constraints [54, 91].

The problem of duplicate record detection and resolution is pursued by adopting integrity constraints in several data cleaning approaches. In particular, the matching dependencies [81] are utilized by NADEEF [62] and used for data cleaning and query answers by Bertossi et al. [25]. Specially defined identity rules [155] are used to identify matched tuples from two different datasets.

One of the natural areas of applying integrity constraints [83, 80, 86], is the problem of integrity constraint violation and its repair. This is the largest group in the category of rule-based data cleaning approaches. The surveyed systems [54, 91, 62, 132] usually assume the presence of integrity constraints and do not provide any functionality to discover them. A new class of *fixing rules* [238] which are based on integrity constraints acts as a foundation for an automated approach to repairing data errors. To estimate data repairs for an inconsistent database with respect to a set of denial constraints, Lopatenko A. et al. [157] provided an approximation algorithm, which translates the denial constraints into an optimization problem.

To tackle the problem of timeliness violation and stale values resolution with the rule-based methods, a special class of *Currency Constraints* is proposed [82, 85, 84]. These constraints are extensions of denial constraints with additional currency information, which is derived from the semantics of data, namely the *currency order for attribute* A_i. This constraint determines the condition when the value of the attribute A_i is more current. The statement above means that the declaration of such rules implies providing the currency order manually. In contrast, Abedjan Z. et al. [1] studied the inclusion of the temporal aspect into the functional dependencies, namely the discovery and application of temporal rules for data cleaning.

In our work, we also leverage integrity constraints to specify data cleaning rules and to convert them into a probabilistic model of error detection and probable repair suggestions [234]. Furthermore, we operate on the output of the aforementioned data cleaning systems and use them as constituent systems for general error detection [232].

3.2 Statistical Approaches

This group of data cleaning prototypes uses statistical methods to detect data errors and determine the repair of corrupt values. The reason behind utilizing statistic-based techniques is that the dataset statistics, such as the values distribution, are useful for detecting errors [190]. Additionally, for some datasets, such as IoT, GPS, sensor reading, no integrity constraints could be declared. Therefore, the rule-based approaches for error detection and repair need to be extended.

Some of the ubiquitous data quality problems, when it comes to applying statistical methods, include missing values detection and imputation. The ERACER system [164] tackles the data imputation problem based on relational learning to determine the characteristics of the attribute relationships in a relational database. This technique uses knowledge about the relationships between the dataset attributes to construct a Bayesian network, which is then used to infer the missing values. The SCARE system [245] proposes the notions of the *likelihood benefit of an update* and *maximal likelihood repair* – a data repairing technique – that finds a limited amount of changes, which maximizes the likelihood of the data, given the underline data distribution. They generate predictions for tuple repairs with the corresponding prediction probabilities for each *dirty* tuple. The *DEC (Detect-Explore-Clean)* framework [22] uses statistical and other analytical techniques, such as the Fleiss' kappa measure, to compute the *glitch score*, which identifies and scores the data glitches. This score forms the base for detecting, quantifying, and correcting data quality problems. Depending on the type of missing data, such as monotone or arbitrary, regression models, respectively the Markov Chain Monte Carlo method can be applied [249]. The ERACER and SCARE systems address the problem of explicitly missing values, such as NULL values. This group together with implicitly missing values (also known as default values or disguised missing values [189]) violate the *Completeness* dimension. The FAHES system [194] collects numerical and categorical value distributions (statistical models) and uses them for the identification of disguised missing values.

The typical task for quantitative techniques is outlier detection – this is the largest group of approaches in this category. The DBOOST system [190] employs both Gaussian modelling and histogram-based methods to capture various types of outliers. Similarly, Subramaniam, S. et al. [224] propose a framework that computes an approximation of multi-dimensional data distributions, to identify distance- or density-based outliers.

Meanwhile, Chung, Y. et al. [56] provide a means of quantifying the remaining errors, namely the estimation of the number of all detectable errors in the dataset by using the *Chao92* estimation technique. Zhang, A. [250] proposes the likelihood-based repairing over sequential data. Hellerstein, J. [114] focused on a statistical properties of the data, such as histograms and correlation, and their usage for computational procedures to identify and correct errors in datasets. The Leap framework [41] tackles the problem of distance-based outliers in streaming environments by leveraging two principles, namely *"minimal probing"*, which collects the minimally needed evidence to identify outliers, and *"lifespan-aware prioritization"*, to find the most useful neighbour relationships for outlier detection. Detecting the inaccuracies in data by using statistical methods has been a target of many already described data cleaning systems [56, 250, 22, 245].

The problem of outdated materialized views on large databases due to incorrect, missing, and superfluous rows has been tackled by the Stale View Cleaning framework [143], which cleans a sample of rows from a stale materialized view and uses the clean sample to estimate aggregate query results by means of a hashing-based technique that generates an up-to-date sample view. Already mentioned data cleaning prototypes [22, 245] use statistical methods to solve multiple data cleaning problems related to missing values, outliers, inaccuracies, and integrity constraint violation. Some scientific prototypes mostly address the problems of inaccuracy detection and repair, as well as the related data cleaning issue, such as integrity constraint violation. Other systems employ metric functional dependencies to generate a minimal repair [193], or use constraints as evidence and propose a repair classifier that predicts the type of repair to resolve the inconsistency. This classifier learns from previous user repair interactions to recommend accurate repairs in the future [235].

In our work, we leverage the output of the above-described data cleaning systems. Since we consider such systems as "black boxes", these approaches can be utilized as constituent systems [232].

3.3 Probabilistic and Machine Learning-Based Approaches

Recently, there is an increasing trend of applying machine learning, artificial intelligence, and probabilistic approaches for data integration and curation tasks. An advantage of probabilistic modelling for data quality has been investigated by Naus, J. et al. [174] and Chen, K. et al. [46]. For example, the Bellman system [66] uses data mining techniques on the database structure to examine the database content, e.g., discover similar values, identify join paths, estimate potential join attributes and join sizes, and identify structures in the database. Probabilistic inference has been used in a number of tasks, including natural language processing [206], ontology alignment, data integration [179] and co-reference

resolution [192]. These research results demonstrate the advantage of joint modelling for various tasks.

The BoostClean system [141] addresses the domain value violations while cleaning training data for predictive models. They provide a library of data cleaning operations, which rely on deterministic rules (error detectors) and statistical criteria. To detect errors in categorical attributes, their framework includes an error detector based on the word embedding model – *Word2Vec* [168] – and implements a neural network architecture. BoostClean employs the classification algorithm AdaBoost for error detection in training data. In our work, we also generate error detectors based on datasets metadata. Furthermore, unlike to their techniques, we employ stacking ensemble learning to combine various error detection strategies [232].

Commonly, to detect and repair data outliers, various data cleaning systems use clustering algorithms. For instance, Song S. et al. [218] proposed to repair inaccurate data during the clustering process. ActiveClean [145] is an iterative model training framework for data cleaning that suggests a sample of data to clean based on the data's value to the model and the likelihood that the data is dirty.

To overcome the shortcomings of the CFDs- and AFDs-based data cleaning, the BayesWipe framework learns the generative model from the data and views the data cleaning problem as a statistical inference over Bayesian Networks problem. In our work, contrary to BayesWipe, we utilize undirected graphical models to generate possible repairs [234]. Although the DataXRay system [240] is a data diagnosis system, they also employ Bayesian analysis to derive the best possible error diagnosis. The HoloDetect system proposes to build a neural networks-based error detection model that requires minimal human involvement [112]. The HoloClean system [202] considers error detection as a black-box component and expects the specification of integrity constraints-aligned data quality rules to make probabilistic suggestions on how to repair erroneous data values.

Machine learning approaches have been intensively applied to address the problem of duplicate detection [12, 27, 242, 144, 222]. For instance, the Data Tamer system [222] uses a Naive Bayes classifier to obtain the probabilities for two tuples to be duplicates and then applies clustering to detect duplicates.

Often, data cleaning is viewed as a probabilistic process that produces various probable repairs. The concept of probabilistic data cleaning has been applied in the context of violations of functional dependencies [26].

In this dissertation, we applied both supervised and unsupervised machine learning approaches to achieve aggregation of various data cleaning solutions, respectively, to identify idempotent systems that provide nearly similar results [232].

3.4 Interactive Data Cleaning

Traditionally, in relational data cleaning, it is almost impossible to quantify the accuracy of an automatic data cleaning process without the ground truth. The need to include human experts into the data cleaning process is usually motivated by the unreliability of the automatic data repair [2]. Furthermore, data cleaning is considered a hard problem, a large number of approaches, which include humans for error detection and error correction. Especially with the development of crowdsourcing platforms, such as Amazon Mechanical Turk [10], humans are intensively being included into various labour tasks related to data cleaning.

The problem of missing values imputation, as well as inaccuracy detection and repair, is addressed by a number of interactive data cleaning solutions. For instance, the Wrangler system [127] introduced a *programming-by-example* concept that implies interactive data transformations by integrating a mixed-initiative user interface with an underlying declarative transformation language. The crowd of experts (oracles), besides the ubiquitous platforms, is used to decide what tuples are missing in the dataset [20]. The Potter's Wheel [197] proposed an interactive system, which uses a transformation language for data formatting and outlier detection based on extracted value patterns. The recently proposed Icarus system [196] employs a heuristic algorithm, which provides the user with small subsets of the database, as well as a set of suggested rules, to guide the human in the task of data completion.

To identify erroneous data, the Falcon framework [110] is bootstrapped with user input about the dirty data. Then, Falcon generates a set of declarative queries, which are also verified by the user. Based on user verification, the system generates data repair on the complete dataset. The Katara data cleaning system [55] is based on both the knowledge base and crowdsourcing. For a provided relational dataset, a knowledge base, and a crowd, Katara understands the table semantics, determines incorrect data, and generates top-k possible repairs for data errors.

Numerous data cleaning systems use crowdsourcing for duplicate detection and resolution [138, 222, 96, 210, 105, 237]. The task of duplicate detection is also known as entity resolution. The crowdsourcing mode of the Data Tamer system [222] allows entity resolution, which entails deciding if two entities are duplicates or not, by using experts associated with the task domain. Similarly, the interactive Magellan system [138] assists users to understand the entity matching scenario and create duplicate detection pipelines by using blocking and matching techniques. The Corleone system [96] minimizes the developer's effort to generate the entire entity resolution workflow by using crowdsourcing to accomplish a task.

The recent Dance system [16] creates a suspicious tuples graph, which enables domain experts to detect and resolve integrity constraint violations. However, they assume that

the provided integrity constraints are correct and reflect the ground truth. Similarly, the Guided Data Repair framework presumes the availability of data cleaning rules to identify the erroneous tuples in the dataset. The framework incorporates a user to verify generated updates.

Fan W. et al. [84] address the problem of determining the most current tuple, given a set of tuples related to the same entity. They studied how the currency and consistency dimensions are interleaving to resolve the timeliness violation. Their framework assumes the availability of various integrity constraints, such as common CFDs, as well as specialized *currency constraints*. The proposed framework for conflict resolution considers the mentioned integrity constraints and involves a user for confirming the truth values, which were deducted according to the provided data quality rules.

In our work, we do not directly assume any human involvement. However, the labelling effort necessary to train data cleaning systems aggregation has to be performed by a user [232].

The summary of all the discussed data cleaning approaches is provided in Table 3.1. We outline the related work along two dimensions:

1. **What-dimension**, which captures common data quality issues and typical data cleaning tasks, which we found in the literature, and
2. **How-dimension**, which reflects different methods adopted by data cleaning approaches.

Data Quality Issues	Rule-Based approaches (Qualitative techniques)	Statistical approaches (Quantitative techniques)	Probabilistic and ML-Based approaches	Interactive, Active Learning, and crowdsourcing	Approaches using profiling and metadata
Completeness violation	[62, 219, 132]	[194, 249, 22, 245, 164]	[141]	[127, 185, 20, 197, 196]	[128, 127, 141, 30]
Accuracy violation	[54, 91, 62, 132]	[190, 56, 250, 22, 245, 114, 224, 41]	[202, 218, 144, 141, 67, 240, 145]	[127, 145, 55, 20, 197, 110]	[128, 127, 141, 30, 190, 239]
Uniqueness violation	[62, 155, 25]	[22, 113, 245]	[12, 27, 242, 144, 222]	[138, 222, 96, 210, 105, 237]	[239]
Consistency violation	[83, 80, 86, 54, 50, 157, 91, 62, 132, 238]	[22, 193, 235]	[202, 26, 110]	[16, 246]	[239]
Timeliness violation	[82, 85, 1, 84]	[143]	timestamp [104]	[84]	[128]

Table 3.1: Summary of related work for different data quality issues and the corresponding type of data cleaning approaches.

4

Anatomy of Metadata for Data Quality Management

Real-world datasets often suffer from data quality problems [161, 201, 195]. Therefore, guaranteeing high-quality data is crucial for all data-driven enterprises and applications [167, 109, 223], as decision-making and analytics demand consistent, accurate, complete and timely data, to avoid the *"garbage in - garbage out"* problem [201]. As elaborated in Chapter 2, we consider data quality as a compound notion that unifies the following five dimensions: *Consistency, Uniqueness, Accuracy, Completeness*, and *Currency* [82, 18]. Real-world datasets are often *dirty*, meaning that they reveal *data quality problems*, which are a violation of at least one of the mentioned five dimensions [82, 201, 126, 142].

Data quality problems have been studied in previous research, which categorized and systematized these issues [147, 152, 195, 133]. The taxonomy, suggested by Kim W. et al. [133], consists of 33 atomic dirty data types. These errors are a hierarchical decomposition of three classes, namely *"missing data"*, *"not missing but wrong data"*, and *"not missing and not wrong but unusable"*. Kim W. et al. [133] focus on "primitive" types of dirty data and exclude dirty data that is a combination of more than one type of dirty data. Rahm E. et al. [195] classify data quality issues regarding the source of information: *single* or *multiple* with further distinctions into *schema- and instance-related* data quality problems. Rahm E. et al. [195] define *multi-source* problems as such that appear when multiple sources should be integrated, otherwise, the dirty data relates to the *single-source* problems. The *four classes* categorization for types of errors was proposed by Abedjan Z. et al. [2]. They group data quality issues into four classes, namely *outliers, duplicates, rule*

violations, and *pattern violation*. The authors also emphasize that their categorization is not exhaustive because some errors might be assigned to more than one category.

In this work, we focus on a *flat* hierarchy of data errors and focus on "primitive" types of dirty data that are described in the literature [147, 152, 195, 133].

Existing research already provided the measures of quality of a given dataset by mapping data quality dimensions to data quality problems [152, 18]. This research showed that each of the data quality issues violates at least one dimension. For instance, the existence of duplicates violates the uniqueness dimension [147]. Referential integrity violations or misfielded values disrupt four dimensions: accuracy, consistency, uniqueness, and completeness. The missing values problem violates two dimensions: accuracy and completeness.

At the same time, a large body of research already proposed various data cleaning approaches that try to address some of these data quality problems [62, 54, 202, 55, 222, 190, 91]. These systems leverage miscellaneous types of metadata, such as value distributions, histograms, or derived integrity constraints. Generally, data profiling plays an essential role in data management. In particular, data preparation intensively uses profiling results for cleaning, wrangling, integrating and maintaining data. Although single-column or structural metadata is actively used in the field of schema matching [72], the usefulness of this category of metadata for data cleaning is less well studied. In fact, data cleaning systems are partially built on metadata, by including profiling capabilities in their engines.

Depending on the *metadata category*, which is primarily used by the particular data cleaning system, we divide existing data cleaning systems into *three groups*:

Dependencies or Integrity Constraints are used as a foundation for data quality rules [82]. Integrity constraints, such as functional dependencies, inclusion dependencies, or denial constraints, form the base for rules, and data that violates these rules is potentially considered erroneous. The group of systems, mainly described as *"rule-based"* systems, leverages different integrity constraints to enable error detection and correction. These systems expect the specification of denial constraints, functional dependencies, and matching dependencies to specify data quality rules [54, 91, 62]. The dataAuditor [97] system discovers data semantic on the provided integrity constraints to analyze the data quality. Another type of systems, mainly related to as *machine learning-based* systems, also employ dependencies, such as *functional* and *matching dependencies*. For instance, the HoloClean [202] system expects the specification of integrity constraints-aligned data quality rules to make probabilistic suggestions on how to repair erroneous data values. Another group of systems employs metric functional dependencies to detect data errors and use statistical approaches to generate a minimal repair [193]. The UniDetect is a perturbation-based framework with the *"what-if"* analysis and is focused on the four types of errors: *numeric outliers,*

misspellings, uniqueness violations, and *FD violations*. UniDetect utilizes *FDs* for the FD-compliance metric.

Dataset statistics, such as value distributions or values frequencies, are included by another group of error detection techniques. The dBoost system [190] employs outlier detection methods based on value distributions and histograms. The Fahes [194] system gathers numerical and categorical value frequencies and utilizes them for the identification of disguised missing values. Other systems, such as Eracer [164] and Scare [245], produce value distributions and leverage probabilistic models to repair data. The UniDetect framework uses metadata, such as *edit distance matrix, statistical dispersion (standard deviation, MAD, and median),* and *distinctness*.

Dataset semantics, such as data types or semantic roles, are used for dataset summary visualizations to support the user in validating and cleaning data. Basically, these visualizations include histograms, area charts, and scatter plots [128]. The Wrangler [127] system uses semantic data types (e.g., zip and state codes, dates, classification codes) to clean and structure (wrangle) data. The Potter's Wheel [197] introduced a transformation language for data formatting and outlier detection based on extracted value patterns.

While some metadata has been incorporated in many data cleaning systems, a comprehensive discussion on the relationship between metadata and data quality problems is still due. To motivate the connection between metadata and dirty data, we now demonstrate how metadata is used in data quality management, by providing the following examples of heuristics and data quality rules, which is aligned to the approach suggested by Rahm E. et al. [195]:

Example 4.0.1 *A trivial relationship can be directly established between the data quality dimension completeness and the number of null values. If the percentage of null values is greater than zero, then it might be an indicator of missing or not applicable values.* ■

Example 4.0.2 *Metadata such as value pattern distributions can be transformed into a formatting rule [173]. Applying these discovered rules would expose all values that do not match this rule as errors, and therefore indicate the existence of either illegal values or domain violation issues. Some typical examples of error detection methods with formatting rules include assessing whether the values conform to an e-mail, US zip code, or a phone number pattern [127]. Figure 4.1 shows how metadata is compiled into rule discovery and error detection.* ■

Example 4.0.3 *The disguised missing values are also known as implicit default values [189, 120, 194]. Practically, disguised missing values are often values with strings of repeated*

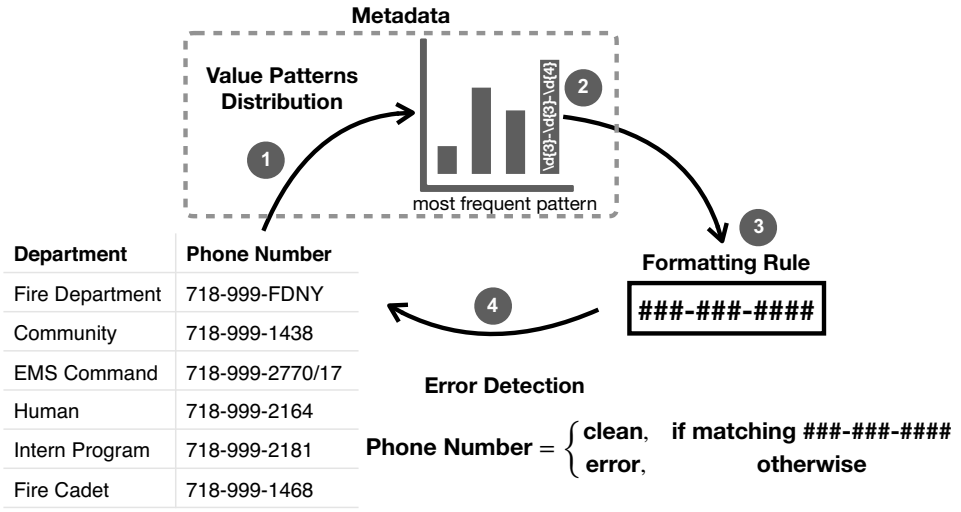

Figure 4.1: Discovery of formatting rules and error detection. The metadata-based error detection is performed by (1) analyzing the distribution of value patterns, which (2) leads to the discovery of the most frequent pattern, which is then transformed into (3) the formatting rule to (4) indicate all values that do not match this rule as errors. Source [233].

characters (e.g., "sdsdsdsd") or digits (e.g., $555-555-5555$*). Using metadata that describes the distinctness of values in a dataset, it is possible to come up with a rule that captures this type of errors. The* FAHES *system [194] leverages distinctness to compute the repeated pattern score for each value and to output the cells with the highest score as disguised missing values.* ■

The examples above show that the ***metadata*** is crucial in detecting data quality problems [4, 147, 152, 195, 133]. Moreover, these examples above expose that when using metadata for data quality rules, one typically composes simple metadata into more complex ones. In particular, metadata, such as *count of rows*, *count of nulls*, *count of value* v, and *count of value pattern* are used as an integral part of more complex metadata such as *completeness*, *distinctness*, and *value pattern distribution*. Therefore, to systematically analyze the metadata, we propose to establish a metadata categorization, which is based on the composability notion.

Provided the motivation and examples above, we now define three challenges in this chapter:

Mapping metadata to data quality issues. As data cleaning approaches tend to focus on only one or few error types, new techniques for the coverage of data errors by using metadata need to be developed. This requires knowing what metadata addresses which data error, as well as how to combine the results of such error detection process.

Metadata Taxonomy. Given a comprehensive list of data profiling, it is necessary to perform a thorough analysis of metadata regarding its usage for each particular task in data cleaning.

Metadata Composability. From the examples above, we found that using metadata for data quality rules requires some simpler metadata functions to represent the final metadata. In order to systematically analyze metadata, we must establish the concept of metadata composability.

In this chapter, we (1) create a qualitatively validated mapping between metadata and data quality issues; (2) establish a novel metadata taxonomy based on the complexity of metadata that is suited for data quality management; and (3) generalize the composability of metadata by introducing a formal description of metadata composition.

Furthermore, we conduct a systematic study of the application of metadata for detecting data quality problems. In the following, we refer to the metadata profiling task as a *metadata function*. Initially, we establish a new qualitative mapping between our new metadata taxonomy and data quality issues. Next, we organize metadata functions by establishing a new two-dimensional metadata classification matrix that reflects the different degrees of granularity. We specify how metadata is composable and formulate a closed grammar based on the Extended Backus-Naur Form [135], which offers a description for all types of metadata functions and enables generating new metadata. Finally, we demonstrate the practical application of our mapping by first, designing a case study where we address the problem of *error detection* on real-world data, second, conducting a user study to evaluate the support of our mapping for the definition of the data cleaning strategy.

This chapter is organized as follows:

1. In Section 4.1, we establish the new mapping between metadata and data quality issues.

2. In Section 4.2, we analyze metadata in the context of data quality management and provide a new two-dimensional metadata classification that reflects different degrees of metadata granularity.

3. The formalization for metadata composition is provided in Section 4.2.2.

4. Finally, we conclude this chapter with a qualitative case study to demonstrate the effectiveness of our mapping and usability study of applying our method for error detection in Section 4.3. We conducted a user study to evaluate the support of our mapping for the definition of the data cleaning strategy in Section 4.3.5

4.1 Mapping Metadata to Data Quality Issues

In this section, we establish a new mapping between the metadata taxonomy and well-known data quality issues. As previously motivated, metadata is essential for initial data quality analysis [195]. We consider the practical usefulness of particular types of the metadata for solving particular data quality issues. The methodology to create such mapping involves the connection of the **Data Errors** and **Metadata** taxonomies (see Chapter 2). In the following, we explain each of these components, as well as the connection between them in the form of a **Heuristic** statement.

Data Errors. Understanding the roots of dirty data is the most crucial aspect of data quality management. As previously mentioned, data errors are violations of data quality dimensions [82]. We use the taxonomy of dirty data, which is already established in the research and has been previously explained in Table 2.1 in Chapter 2. To connect data quality dimensions and data quality issues systematically, Laranjeiro N. et al. [147] used the categorization proposed by Rahm et al. [195] and mapped various data quality problems identified by the research community to the five central data quality dimensions [82], as shown in Table 2.1. This Table contains a detailed description of data quality issues as proposed by Laranjeiro N. et al. [147], Oliveira P. et al. [180] and Kim W. et al. [133].

Metadata. In the previous Section 2.2, we reviewed the metadata taxonomy, which is known in the literature [3]. For establishing the mapping, we include all metadata that is captured in that taxonomy.

Heuristics. One way to build a new mapping between metadata and data quality issues is to combine the taxonomies on **Data Errors** and **Metadata** by using the *weak supervision* approach. The idea of *weak supervision* or *distant supervision* originates in the *Information Extraction* field, and uses facts from existing knowledge bases or expert knowledge to generate annotations for extracted entities [169]. In the field error detection, we use *weak supervision* as a technique that heuristically matches data errors to the metadata which is used to detect these errors [198]. Thus, we define **Heuristics** as error detection rules that are declared by using metadata. Analogously to the formulation of edit rules [89], heuristics use metadata to specify the error localization logic. Concretely, we formulate first-order logic predicates by using metadata statements. The intuition behind the mapping between **Data Errors** and **Metadata** taxonomies is that, since metadata describes and explains the information resource [207], then metadata should also describe the data quality issues. Mainly, we achieve the mapping between **Data Errors** and **Metadata** taxonomies by specifying a heuristic that determines:

1. **what** data error should be identified; and

2. **how** the data error is described by metadata, and therefore how it can be detected;

Formally, we consider any heuristic as an *indicator function* $\mathcal{H} : \mathcal{M} \rightarrow [0;1]^n$ that maps the set $\mathcal{M}$ of metadata to the binary values $[0;1]$:

$$h = \mathbb{1}\left\{\phi \; is \; true\right\},$$

where the result 1 denotes that the respective dataset value is *erroneous*. Otherwise it is 0, meaning *"clean"*. The ϕ represents an *error condition* that is an either universally- or existentially-quantified logical sentence (predicate) that states how the data error is identified. Each of these logical sentences is created by using one of the metadata functions $\mathcal{M}$, a set of mathematical operators ($=$, $\approx$, $<$, $>$, $\leq$, $\geq$, $\in$, $\subset$, $\supset$, $\subseteq$, $\supseteq$) or user-defined operators (e.g., *match*), and possible thresholds.

Because each heuristic captures at least one type of data errors, we would need to combine all heuristic results to obtain the *overall error detection* results $\mathbf{e}$ for the whole dataset $\mathcal{D}$:

$$\mathbf{e} = \bigcup_{h_i \in \mathcal{H}} h_i$$

In terms of methodology, we utilize the following two approaches to identify or specify error detection heuristics:

The qualitative approach: Provided with the existing body of research, we already studied diverse methods which detect data quality violations. We identified such methods that utilize metadata to detect errors by reviewing the literature and gathering the developed heuristics. For instance, error detection can be done using formatting rules to verify values on an *e-mail address*, a *US zip code*, or a *phone number* column [127]. Many of these approaches are built based on the frequency and extreme-values statistics [114, 116], application-specific requirements [18], domain and schema knowledge [55], and outlier detection [8].

The trivial relationship approach: Often, metadata is a direct indicator of a data error. For example, the number of *null values* inside a column explicitly identifies the degree of completeness of a column. Another logically derivable relationship is between the number of *distinct values* and the number of *uniqueness violations* [5].

Provided with both taxonomies – data quality issues [147, 180, 133] and metadata [4] – we pursue to design an error detection heuristic that addresses exactly a particular data quality issue by using some particular metadata. We specified these heuristics by applying one of the above-described approaches. Importantly, each time we were able to create an

error detection heuristic rule by using the metadata, we established a mapping between the corresponding data quality issue and metadata.

The complete mapping of the data quality issues to the metadata is shown in Table 4.1. This table includes mapping of 30 metadata categories to 23 types of errors. In this research, we identified 160 heuristics from 690 possible variants. For each data quality issue from Table 2.1 and each metadata from Section 2.2, we attempted to design an error detection heuristic that addresses exactly that particular type of data error by using that particular metadata. In Table 4.1, we visualize the mapping accordingly either with the reference of the piece of literature that contains such a heuristic or with a dot (•) that represents a trivial mapping. At the top of Table 4.1, we list the data quality dimensions and represent the link between quality issues and their dimensions with ticks. Some heuristics are based on multiple references. For example, the mapping between the *Misspellings* error type and the *histogram* metadatum is supported by three references [3, 114, 190]. Although, to formulate a heuristic, one reference is sufficient, and for the sake of completeness, we provided all references that were identified during the research. Some metadata, such as the *number of rows, min, max, mean*, and *count/similarity matrix*, are not explicitly used in the mapping, because these functions are utilized by higher-order metadata. As provided in Section 4.2.3, according to the *composability* of the metadata, the *number of rows* is used to compute other metadata, such as *distinct* or *constancy*. The following metadata functions, such as *min, max,* and *mean*, are used as thresholds in the error detection rules [195].

We provide an additional Table 4.2 that visualizes the mapping of metadata to the data quality dimensions without the intermediate mapping to the data quality issues.

This mapping reveals the tendency that several error types are supported by many metadata categories, while other error types require fewer metadata. For example, the following data quality issues are mapped to 15 metadata categories on average: *missing data, incorrect values, misspelling, ambiguous values, extraneous data, misfielded values,* and *domain violation*. Furthermore, the above mapping reveals that schema-level data errors mostly require schema-related metadata to be captured. For instance, the *FD violation* issue is identified by heuristics, which requires an integrity constraint specification [82, 54, 62]; or an alternative to functional dependencies, such as *association rules* [17]. We observe the similar trend for the *wrong data type* error – the heuristics imply the specification of the *data type, data class,* and *domain* metadata [3, 128, 127]. Even though the *domain violation* error is a schema-level issue [147], this error depends on the data type, namely numeric, categorical, or temporary data. Therefore, the *domain violation* issue is backed by 14 types of metadata reflecting various data type of the attribute.

		Missing data	Incorrect data	Misspellings	Ambiguous data	Extraneous data	Outdated temp data	Misfielded values	Incorrect references	Duplicates	Structural conflicts	Wrong word orderings	Wrong aggregation level	Temporal mismatch	Wrong representations	Domain violation	FD violation	Wrong data type	Integrity violation	Uniqueness violation	Use of synonyms	Use of homonyms	Use of special characters	Different encoding formats
Data Quality Issues		Instance-Level Issues														Schema-Level Issues								
Dimensions	Accuracy	✓	✓	✓	✓			✓	✓				✓			✓	✓		✓					
	Consistency				✓	✓		✓			✓	✓	✓		✓		✓	✓	✓				✓	✓
	Uniqueness					✓				✓	✓	✓								✓	✓	✓		
	Completeness	✓						✓											✓					
	Currency						✓							✓										
Metadata	number of rows																							
	null values	▪						▪												[3]				
	value length	▪			[195]	[195]		[195]																
	first digit	[3]	[3]																					
	size (numeric)	▪		[195]	[195]								▪			[195]								
	decimal															[3, 195]								
	pattern			[3, 114]	[3]	[114]	[104, 49]	[3, 114, 116]		[3, 195]			▪	[104, 49]	[3]	[3, 114]								
	soundex			[3]		[114, 116]				[3, 195]		[3, 128]												
	n-grams		[114]	[3, 114]		[114, 116]		[114, 116]		[3, 195]		[51]				[51]								
	hash code			[3]		[114, 116]				[3, 195]														
	min/max/mean																							
	std. deviation				[17]	[17]		[17]								[17]								
	distinct			▪						▪					[195]	[195]				[3]				
	z-value			[128]	[128]		[128]																	
	quartile		[114, 116, 17]	[17]	[17]	[17]		[114, 116]																
	count/sim. matrix	[116]		[239]				[195]		[3, 195]	[195]	[3, 128]	[128]	[82, 104]							[195]	[195]	[128]	
	word embeddings		[141]	[141]				[4, 141]				[141]			[141]	[141]								
	constancy		[114]	[114]																				
	correlation			[51]	[51]				[51]			[51]				[51]								
	clustering	[116]		[128]			[82, 104]	[195]		[3, 195]	[195]	[3, 128]	[128]	[82, 104]							[195]	[195]	[128]	
	histogram	[3]	[3]	[3, 114, 190]	[114]	[114, 116]		[114, 116]								[3, 114]				▪			[114, 128]	
	summaries			[3]							[215]	[215]												
	outliers	[114, 190, 189, 17]	[8, 114, 116, 190]	[190]	[114, 116]	[114, 116, 190]	[128]	[114, 116]								[114, 116]				▪			[116, 190]	
	association rules		[17]	[17]	[17]	[17]		[17]	[3]			[17]				[17]	[17]						[17]	
	data type	[3]	[3]		[3]			[3]								[195]		[3, 195]					[3]	[3]
	data class	[3]				[3, 128]										[3]		[3]						
	domain	[228, 189]	[228]	[228]	[228]	[228]		[228]							[228]	[228]		[228]						[228]
	UCC			▪						▪ [239]										[3]		▪		
	FD/CFD		[3, 62, 82]	[82]	[202, 91]	[202, 91]	[1]	[82]	[82]	[82]MD				[124]			[82, 54, 62]							
	IND/CIND																		[3, 82]					

Table 4.1: Mapping data quality problems to metadata. The mapping between data quality dimensions and the data quality issues at the top is adopted from [147]. The mapping between data quality issues and metadata is established by designing error detection heuristics. To create these heuristics, two approaches are used: (1) - A qualitative approach, where existing methods are reviewed; and (2) - A trivial relationship approach, where the connection between data errors and metadata is trivially established (marked as •). Source [233].

		Data Quality Dimensions				
		Accuracy	Consistency	Uniqueness	Completeness	Timeliness
Metadata	number of rows					
	null values	▪ e.g., missing val.	▪ e.g., misfielded val.	[3]	▪ e.g., missing val.	
	value length	[195]	[195]	[195]	[195]	
	first digit	[3]			[3]	
	size (numeric)	[195]	[195]		▪ e.g., value size >0	
	decimal	[3, 195]				
	pattern	[3, 114, 116, 189]	[3, 114, 116]	[114]	[3, 114, 116, 189]	[104, 49]
	soundex	[3, 114, 190]	[3, 114, 116, 128]	[3, 114, 116, 128]		
	n-grams	[3, 190, 51]	[116, 51]	[3, 51]	[116]	
	hash code		[114, 116]	[114, 116]		
	min/max/mean	[17]	[17]		[17]	
	std. deviation	[17]	[17]		[17]	
	distinct	[195]	[195]	[3]		
	z-value	[128]	[128]		[128]	
	quartile	[17]	[17]	[17]	[17]	
	count/sim. matrix	[116, 195, 128]	[3, 195, 128]	[3, 195, 128]	[116, 195]	
	word embeddings	[4, 141]	[4, 141]	[141]	[4, 141]	
	constancy	[114]				
	correlation	[51]	[51]	[51]		
	clustering	[116, 195, 128]	[3, 195, 128]	[3, 195, 128]	[116, 195]	[82, 104]
	histogram	[3, 114, 116, 190]	[114, 116]	[114, 116]	[3, 114, 116]	
	summaries/sketches	[3]	[215]	[215]		
	outliers	[8, 114, 116, 190, 189]	[114, 116, 190]	[114, 116, 190]	[114, 116, 190, 189]	[128]
	association rules	[3, 17, 6]	[17]	[17]	[17, 6]	
	data type	[3, 195, 128]	[3, 82, 128]		[3, 128]	
	data class	[189]	[3, 128]	[3, 128]	[189]	
	domain/semantic role	[228, 189]	[228]	[228]	[228]	
	unique column combination	▪ e.g.,misspelling		[3]		
	FD/CFD	[82, 54, 91, 62]	[82]	[82, 54, 91, 62, 202]	[82]	[124, 1], temporal FDs
	IND/CIND	[3, 82]	[3, 82]		[3, 82]	

Table 4.2: Mapping data quality dimensions to metadata. Legend: (1) - A qualitative approach, where existing methods are reviewed; and (2) - A trivial relationship approach, where the connection between data errors and metadata is trivially established (marked as •). Source [233].

Next, we identify the most prevalent metadata by analyzing the coverage of data errors by the particular metadata. We identified two groups of metadata: the first group covers at least seven data quality issues and consists of *pattern, n-gram, count/similarity matrix, clustering, histogram, association rules, outliers, domain, data type,* and *FD/CFDs*. Therefore this group of metadata is perceived as the most effective because the above-described metadata covers seven to twelve errors. For instance, the *histogram* metadata is a foundation for heuristics to detect the following data quality issues: *missing data, incorrect data, misspellings, ambiguous data, extraneous data, misfielded values, domain violation, uniqueness violation,* and *use of special characters*. These heuristics have already been described in at least five literature references [3, 114, 190, 116, 128]. The second group, consisting of the remaining metadata functions, covers less than seven data errors.

The complete list of data quality issues and the corresponding heuristics for error detection based on metadata is provided in Appendix A. As an example, we extracted one of the error detection strategies for the *misfielded values* column from Table 4.1 into Table 4.3. This table provides all heuristics, which include metadata, to detect the *misfielded values* data errors.

In the following, we provide three representative examples, namely *misfielded values* detection, *domain violation* detection, and *functional dependency (FD) violation* detection, to demonstrate how we formulate the heuristics and therefore how we established the mapping, in full detail.

Metadata	Heuristics to identify **Misfielded values**
null values	If "neighbor"-values, $attr_i - 1$ or $attr_i + 1$ are NULL, then $attr_i$ is a potential ERROR; ▪
value length	If length(value) >max threshold, then ERROR; [195]
pattern	If the value is in the tail of the value pattern distribution, then ERROR; [114, 116]
z-value	Check z-value against a threshold; [195, 8]
quartile	Use quartiles for all histograms as threshold values;[114, 116]
clustering	Create numerical representations of values from a column range [a_i-1, a_i, a_i+1],(eg.word2vec); Cluster these vectors; Tuples with 'Misfielded values' should be captured within one cluster; [195]
histogram	If a value is in the tail of the values distribution, then ERROR; For the alphabetic data: If one of the n-grams of the value is in the tail of the n-grams frequencies distribution, then ERROR;[114, 116]
outliers	Histogram-based outlier detection: Check extreme values in the histogram; Use quartiles for all histograms as threshold values; [114, 116]
association rule	Identify data items that broke the rules and can be considered outliers (potential errors) [17]
data type	Check data type format [3]
domain	Check semantic role (zip, city, state) with regex; [228]
FD/CFD	FD $L \rightarrow R$ holds for any two rows u and v. Suspected violation: $\{u \vert u, v \in \mathcal{D}, u(L) = v(L), u(R) \neq u(R)\}$; [62, 82, 239]

Table 4.3: Defining data quality strategy by using metadata. This table is an excerpt of schematic heuristics used to detect misfielded values. All error detection rules are provided with references. A trivial relationship approach, where the connection between data errors and metadata is trivially established, is marked as • . Source [233].

Example 4.1.1 *The problem of misfielded values [195, 147] occurs when values of one attribute are placed inside the wrong column, as shown in Table 4.4. One possible reason for this error might be the attribute values shift to the left, due to mistakes in the CSV-format.*

US City	US State
San Diego	CA
Minneapolis	MN
CA	*null*
Minneapolis	NM
Santa Clara	C

Table 4.4: An example data set, which contains data issues, such as *missing value, domain violation,* and *functional dependency violation.*

Here, the "US state" value "CA" is incorrectly stored in the "US City" column. The "US City" and "US state" value distributions contain semantically different information. Considering the value length distributions of both columns, values of the "US State" attribute inside the "US City" column will emerge as extreme values in the "US City" value length distribution because state abbreviations are typically significantly shorter than city names [8].

Now we consider metadata, which might be useful to detect misfielded values. One way to identify extreme values is to use z-values [8]. Therefore, we employ the qualitative approach and answer the following question: ***how*** *are the misfielded values identified by using the z-value metadata? We formulate the following heuristic: "Flag an attribute value as misfielded, if the z-value score, computed on the value length distribution, is greater than or equal to three". That is, we identify extreme values (errors) on the value length distribution instead of just the value distribution. We define extreme values as string values that are too long or too short compared to the average value of the attribute. The primary assumption for designing such heuristic is that by computing the distribution of the length values of the attribute $\mathcal{R}.a_i$, values that are beyond the upper and lower boundaries from the middle (mean) point will be flagged as potential outliers [8]. Taking the z-value heuristic into account, the rule for outlier detection to identify misfielded values is specified as follows:*

$$h_{z-value} = \mathbb{1}\left\{z - value(length(\mathcal{R}.a_i)) \geq 3\right\} = \begin{cases} error, & if\ z - value \geq 3 \\ clean, & otherwise \end{cases} \blacksquare$$

Figure 4.2 also shows all components needed to design a *misfielded values* error detection heuristic by using the *z-value* metadatum.

Example 4.1.2 *The second example considers the domain violation data issue [147], also known as illegal values [195]. We use the relation $\mathcal{R}$ with attributes "US City" and "US State" from the previous example. By applying the qualitative approach, based on the following research [228, 127, 173], we identified a schema-based metadatum that is related*

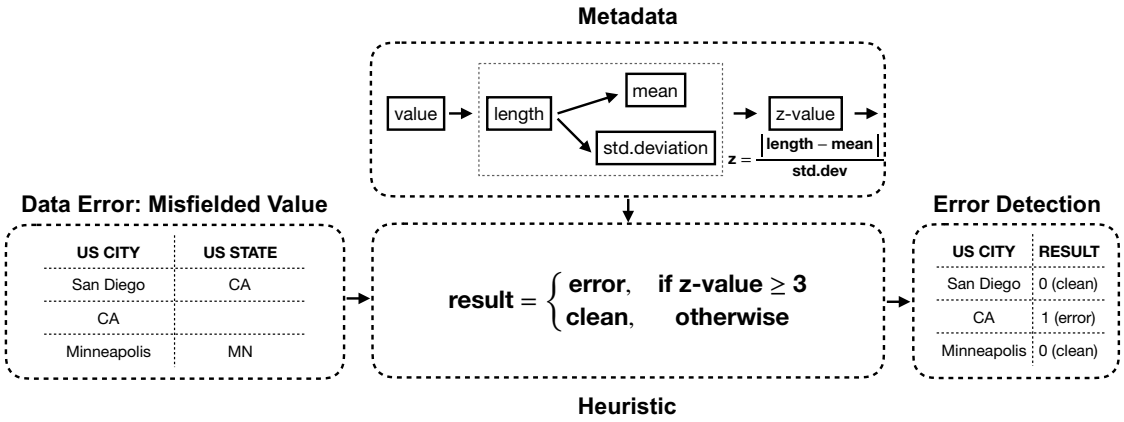

Figure 4.2: An example of mapping between the data error "misfielded value" and the metadata "z-value". To detect misfielded values in the attribute "US CITY", we specify the heuristic for error detection that is based on the z-values of the value length distribution of this attribute. Suspicious values are identified by setting a threshold for z-value scores. Source [233].

to the problem of domain violation: the domain format [228, 127, 173]. The possible domain format for the attribute "US State" is defined by using regular expression [228]:

Rule A. *Domain format for US State attribute: [A-Z]{2}*

The above rule regulates the domain format for values of the particular attribute (i.e., US State). Hence, the heuristic for domain violation detection is specified as follows:

$$h_d = \begin{cases} error, & if\ v_{i,j}\ does\ not\ match\ Rule\ A \\ clean, & otherwise \end{cases}$$

By applying this error detection rule, we can spot the illegal value in the last tuples: "US State" attribute value "C". ■

Example 4.1.3 *Our third example deals with the data error functional dependency (FD) violation [170, 82]. We use the same relation $\mathcal{R}$ with the attributes "US City" and "US State" from Table 4.4. One natural way to identify the FD violation is to use the FD compliance metric to identify erroneous rows [239]. Hence, the FD-compliance heuristic requires the specification of the functional dependency metadatum, such as $\phi : US\ City \rightarrow US\ State$. The heuristic for domain violation detection is formulated as follows:*

$$h_d = \begin{cases} error, & if\ for\ any\ two\ rows\ u\ and\ v, \{u|u,v \in \mathcal{D}, u(City) = v(City), u(State) \neq u(State)\} \\ clean, & otherwise \end{cases}$$

By applying this error detection rule, we are able to spot the FD violation in two tuples, where "US State" attribute values are "MN" and "NM". ■

We provided three representative examples, for *misfielded values* detection, *functional dependency (FD) violation*, and *domain violation* detection, to demonstrate how we formulated the heuristics and therefore how we established the mapping, provided in Table 4.1.

In particular, the advantage of using metadata for detecting a violation of a particular data quality *dimension* becomes apparent when we map metadata to the core data quality dimensions, which is shown in Table 4.2. This table includes mapping of all metadata functions to the five core data quality dimensions, which are described in Section 2.1. Table 4.2 reveals that nearly all metadata categories are useful to detect at least four data quality violations, namely *accuracy, consistency, uniqueness,* and *completeness* violation. To detect the *timeliness* violation, we identified four metadata categories to be used in heuristics, namely *pattern, clustering, outliers,* and *temporal FDs.*

4.2 Metadata Analysis for Data Quality Management

We now establish a new two-dimensional metadata classification matrix that reflects the different degrees of granularity and types of metadata.

Provided with a set of metadata produced by the profiling methods, Abedjan Z. et al. [3] established a comprehensive taxonomy of metadata. They classified profiling functions with regard to *single-* and *multi-column affiliations* and *dataset dependencies.* The *single columns* category includes the following metadata: *cardinalities, patterns, data types, value distributions, histograms,* and *domain classification.* The third category of profiling tasks comprises dependency-related metadata such as (conditional) functional dependencies, (conditional) inclusion dependencies [82], and unique column combination. Being a general-purpose data profiling classification, it is applied for a wide-range use – from database reverse engineering, data exploration, and query optimization [162] to dataset visualization recommendation [119].

Hence, for data quality management we would need to categorize metadata that is best suited for improvement of the data quality. For this reason, building on the metadata taxonomy as described above, we create a *new metadata classification for data cleaning.*

4.2.1 A Two-Dimensional Classification of Metadata

In order to use metadata for data quality management, we first categorize metadata with regard to its granularity. We adapt the categorization by Abedjan Z. et al. [3] and reduce their metadata classification to two broad categories, namely the *single-* and *multi-column* profiling tasks. Since the dependencies category covers multiple columns as well, this category is subsumed by the *multi-column* profiling. Concretely, the first *single-column* category is assigned the following metadata functions: *cardinalities, patterns, data type,*

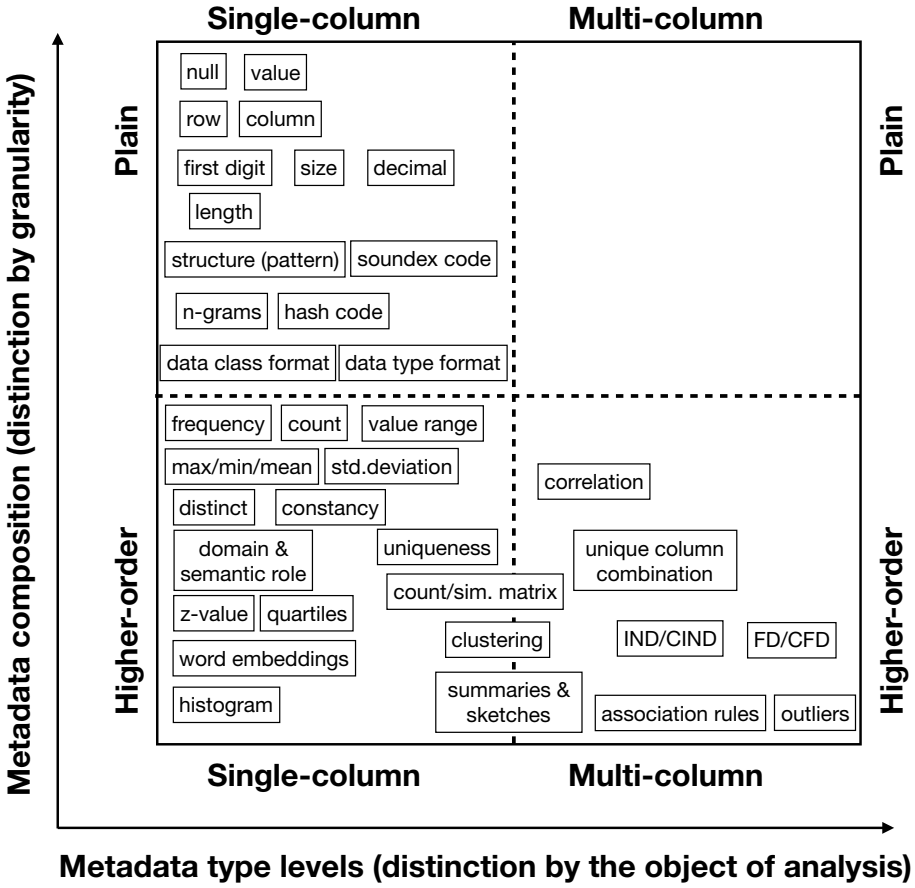

Figure 4.3: Metadata quadrants. The granularity of the metadata functions is augmented by the metadata type level that denotes instance- or schema-based metadata.

domain and semantic role, and *value distribution.* The *multi-column* metadata contains the following general groups of metadata: *correlations, clusters, outliers, summaries, sketches, uniqueness, inclusion dependencies* and *functional dependencies.* The second dimension reflects the granularity of metadata describing its composability that ranges from plain (or atomic) to more complex higher-order metadata.

We obtain our two-dimensional categorization by combining these two dimensions. The resulting categorization divides metadata into *four quadrants*, as illustrated in Figure 4.3. Next, we explain each quadrant.

The **PS-quadrant** specifies metadata categories that belong to ***P**lain* metadata and relate to the content of the dataset, namely the ***S**ingle-column metadata.* This group is populated by different data units, such as *column, row, value, null value*, or *first digit.* This quadrant also includes alternative representations of the raw values, such as *n-grams, hash code, pattern, soundex code,* or *length of the value.* Schema-related information about the values within one attribute is represented by *data class* and *type format.* This

Values	Data Units	Map or apply-to-all metadata	Fold or aggregate metadata	Higher-Order metadata	Descriptive data mining
Alphanumeric	Row Column Value Null Char	Length N-gram Numerical representation of values (word embeddings) Soundex Patterns (Structure extraction) Hash	Count Covariance Correlation Constancy Data type format Data class format	Histogram (Frequency statistic) Count matrix Similarity matrix Domain & semantic role FD (CFD) IND (CIND) Uniqueness Unique column combination	Outliers Summaries & sketches Clustering Association rules Itemset frequency
Numeric	Value Char	Length Size Decimal First digit	Count Covariance Correlation Constancy Data type / class format Min Max Mean Median Standard deviation Quartiles	Z-Value Value Intervals Histogram (Frequency statistic) Count matrix Similarity matrix Domain & semantic role FD (CFD) IND (CIND) Uniqueness Unique column combination	Outliers Summaries & sketches Clustering Association rules Itemset frequency

Table 4.5: Metadata Categorization. Metadata functions are divided into four categories to reflect the composability aspect of metadata: Group 1: Map Metadata; Group 2: Fold Metadata; Group 3: Higher-Order Metadata; Group 4: Descriptive data mining methods. The Data Units column contains the data without meta-information. Furthermore, in the provided classification, we distinguish between numeric and alphanumeric data values and identify those metadata that operate on the above value types. Source [233].

metadata includes schema type information, such as alphanumeric, numeric, varchar, or floating-point.

The **HS-quadrant** includes more complex ***H****igher-order* metadata functions, which are computed on ***S****ingle columns*. Generally, these sophisticated metadata functions require an element or a set of elements from the PS-quadrant as input. This group of metadata functions include simple aggregates, such as *min, max, mean, standard deviation, z-value, quartiles*, and more complex functions, such as *frequencies, counts, distinctness, uniqueness, constancy, histogram,* and *value ranges*.

The **HM-quadrant** mainly includes ***M****ulti-column* and *dependencies* metadata functions [3] of a dataset, such as *unique column combination, INDs, CINDs, FDs, CFDs*. Similarly, *correlation, association rules* and *outliers* are computed on several columns. This **HM**-quadrant interleaves with the **HS**-quadrant, because they include metadata that relates to both metadata type levels, namely single- and multi-column metadata functions. The main reason to place *clustering, count/similarity matrix*, and *summaries/sketches* to both quadrants is that these metadata functions can be executed either on multiple or on single columns.

By leveraging the above-described two-dimensional taxonomy, we analyze the composition of the metadata functions and make a precise classification regarding the granularity of these metadata functions.

4.2.2 Metadata Categorization

Next, we specify our metadata categorization to show how metadata is composable. As already mentioned, we consider a profiling task as a function. The intuition for our metadata categorization is that many complex metadata functions are compositions of other more basic metadata functions. Hence, the metadata function composition is a mechanism that combines plain metadata to build more sophisticated ones [162]. Analogously to the composition of functions in mathematics, the result of each metadata function is passed as the argument of the next. Refining the two-dimensional categorization as described above, we identify four metadata categories: *Map or Apply-to-All Metadata*, *Fold or Aggregate Metadata*, *Higher-order Metadata*, and *Descriptive Data Mining Methods*. This categorization separates metadata functions based on their granularity, which ranges from the most basic to more complex metadata functions. Table 4.5 shows the metadata categorization, where we distinguish between metadata, which operates on numerical data only and metadata that is computed on alphanumeric data. In the following, we provide details about each category.

Our categorization operates on data units without meta-information, including *row, column, value, null value,* and *character* (as an atomic part of the value). We also put *null values* in this group because it represents absent values in the column (see also Table 4.5).

Map or Apply-to-All Metadata. Motivated by the map paradigm from functional programming [165], this group of metadata functions gathers the set of functions that create more sophisticated metadata when applied on any element of the *plain metadata* category. For example, the metadata function *length*, applied to the plain metadata unit *values* $\mathcal{V}$ of a column $\mathcal{R}.a_i$, computes a vector $\mathbf{l} = (l_1, l_2, \ldots, l_n)$ where each element of $\mathbf{l}$ is the length of the string representation of the corresponding value: $l_j = length(v_{i,j})$. The value can be either a numerical or an alphanumerical value. Similarly, the *first digit* function is applicable to numerical values of the column $\mathcal{R}.a_i$, and produces a vector $\mathbf{f}$, where each element of $\mathbf{f} = (f_1, f_2, \ldots, f_n)$ is the first digit of the corresponding numerical value: $f_j = first - digit(v_{i,j})$.

Alternative representations of the nominal values in a given column, such as *n-grams, numerical vectors representation, (e.g. word2vec [168])*, *soundex*, and *hash code* are the transformations of the original column values into a different format (see also Table 4.5 Group 2). The *Map Metadata* is usually necessary for further processing of data mining or natural language processing tasks [168]. Because the *Map Metadata* contains suitable items to be counted and aggregated by the following group of metadata functions, this metadata is an intermediate representation of the actual data under assessment.

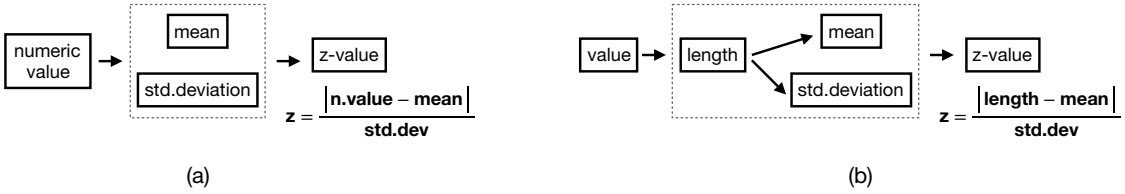

Figure 4.4: Z-value computation. This figure demonstrates the metadata to compute z-values on (a) the distribution of numerical values, and (b) the value length distribution. Source [233].

Fold or Aggregate Metadata. This category of metadata functions is inspired by the fold paradigm [165]. The *fold* function is used to combine all the elements of a column or row and produce "aggregate" results from either raw data or the intermediate representation of the data taken from the *Map*-group. The *fold* or *aggregate* function takes two arguments: the input sequence and the *fold*-function, such as *min, max, mean, median, quartile, covariance, correlation,* and *constancy*. The result of the "folding" operation depends on the applied *fold* function. For metadata such as *min, max, mean, median, quartile, covariance, correlation,* and *constancy*, the result of the folding function is a single real value number $n \in \mathbb{R}$. For instance, the result of the *count* metadatum is a natural number $n \in \mathbb{N}$. We also assigned the *data type* and *data class format* to the *Fold Metadata* group, because they summarize the values in each column.

Higher-Order Metadata. This group of profiling tasks relates to functions that compute compound metadata by taking the raw data or its intermediate representation and metadata from either the *Map* or the *Fold* categories. The *Higher-order Metadata* group includes *frequency statistics, histogram, z-values, uniqueness, count matrix,* and *similarity matrix*. Their results are mathematical objects, such as a set of tuples or numbers (see also Table 4.5 Group 4). The following examples provide the composability of the *histogram* and *z-value* metadata. While some of the mentioned metadata, such as *histogram*, is perceived as a visualization [128], they can also serve as mathematical objects that approximate data distributions [123]. A histogram on an attribute $\mathcal{R}.a_i$ is constructed by grouping its values $\mathcal{V}$ into disjoint *buckets* and counting the frequency for each bucket [122]. Technically, histograms are pairs (v_i, f_{v_i}), where $v_i \in \mathcal{V}$ is the i-the value of the value set $\mathcal{V}$ and $f_{v_i} = count(v_i)$ denotes the frequency of the value v_i. Furthermore, the categorical data is aggregated using a *"group by"* function [123]. Each of the pair components belongs to the *Plain, Map*, respectively *Fold* metadata categories. For instance, a histogram might be constructed as one of the following kind of pairs: (*value*, *count*), (*value length*, *count*), or (*value pattern*, *count*), where the *value* is the *Plain* metadata, *value length* and *value pattern* belong to the *Map*-category, while *count* is assigned to the *Fold*-metadata category.

Example 4.2.1 *To demonstrate the metadata from the Higher-order Metadata group, we provide an example of the z-values [8] (see also Figure 4.4 (a)). Statistically, z-value denotes j-th observation of a given random variable. We compute z-values on the $\mathcal{R}.a_i$ attribute values as follows:*

$$z_{i,j} = \frac{|l_j - \mu(length(\mathcal{R}.a_i))|}{\sigma(length(\mathcal{R}.a_i))},$$

where $\mu(length(\mathcal{R}.a_i))$ is the mean and $\sigma(length(\mathcal{R}.a_i))$ is the standard deviation of all observations "$length(\mathcal{R}.a_i)$"; where $length(\mathcal{R}.a_i) = (l_1 \dots l_n)$, and $l_j = length(v_{i,j})$, $j \in [1, n]$ denotes a length of the corresponding value $v_{i,j}$. Hence, to compute z-values, we would need to pre-compute metadata from the three former categories: value from the Plain metadata, length from the Map metadata and mean and standard deviation from the Fold metadata category. The schematic computation of the z-value metadata is provided in Figure 4.4 (b). ■

Descriptive Data Mining Methods. We also consider the importance of gathering new insights about the dataset. Therefore, we use unsupervised data mining techniques for data profiling [195, 3]. To reflect this kind of metadata functions, we selected another group of methods related to the data mining algorithms. The goal of these methods is to obtain interesting characteristics of the dataset [3]. The following data mining methods belong to this group: *outlier detection, summaries and sketches generation, clustering, association rules mining*, and *itemset frequency computation.* Group 5 shows this group of metadata. The following example aims to demonstrate the composability of the outlier metadata.

Example 4.2.2 *Continuing example 4.2.1 about the composability of the $z-value$ metadata from the Higher-order Metadata group, one straightforward approach for outlier detection is based on the z-values. This outlier detection rule is specified as follows:*

$$outlier = \begin{cases} yes, & if\ z_{i,j} \geq 3 \\ no, & otherwise \end{cases}$$

Note that the threshold score is set to 3, which is a simple heuristic [8] to denote the distance from the distribution mean. Usually, such heuristics are context-specific. Alternatively, reviewing the frequency of values and identifying values that have unusually high frequencies provide a mechanism to identify outliers [114]. ■

To summarize, we demonstrated the composability of the metadata functions, which is the foundation for the metadata categorization for data quality management. Next, we provide a generalization of metadata composability with the aim to construct new metadata.

4.2.3 Formal Description of Metadata Composition

In the previous section, we provided a classification of metadata functions according to their composability. As metadata can be aggregated and categorized, we can formally describe the metadata functions to provide a generalization for constructing more complex metadata structures by using simple metadata functions. Technically, the functional decomposition allows us to determine the most re-usable parts of the metadata functions far earlier in the development cycle.

We specify a semantic description of metadata composition in Extended Backus-Naur form (EBNF) [135], by introducing two conceptual groups of rules:

Group 1. Metadata Classification Rules:

⟨data-unit⟩ ⊨ *row* | *column* | *value* | *null-value* | *char*

⟨map⟩ ⊨ *length* | *n-gram* | *word embeddings* | *soundex* | *patterns* | *hash code*

⟨map-numeric⟩ ⊨ ⟨map⟩ | *length* | *size* | *decimal* | *first digit*

⟨fold⟩ ⊨ *count* | *covariance* | *correlation* | *constancy* | *data type format* | *data class format*

⟨fold-numeric⟩ ⊨ ⟨fold⟩ | *count* | *min* | *max* | *mean* | *trimmed mean* | *winsorized mean* | *std.deviation* | *trimmed std.deviation* | *winsorized std.deviation* | *median absolute deviation* | *quartiles* | *covariance* | *correlation* | *constancy*

⟨h⟩ ⊨ *histogram* | *frequency stat.* | *z values* | *count matrix* | *similarity matrix* | *domain and semantic role* | *FD* | *CFD* | *IND* | *CIND* | *uniqueness* | *unique column combination*

⟨h-numeric⟩ ⊨ ⟨h⟩ | *z values* | *value intervals*

⟨d⟩ ⊨ *outliers* | *clustering* | *summaries-sketches* | *association rules* | *itemset frequency*

Group 2. Metadata Composability Rules:

⟨separator⟩ ⊨ ,

⟨f⟩ ⊨ ⟨map⟩ | ⟨fold⟩

⟨f-numeric⟩ ⊨ ⟨map-numeric⟩ | ⟨fold-numeric⟩

⟨arg⟩ ⊨ ⟨data-unit⟩ | ⟨f⟩(⟨arg⟩) | ⟨f-numeric⟩(⟨arg⟩)

⟨args⟩ ⊨ ⟨arg⟩ {⟨separator⟩ ⟨arg⟩}

⟨ext args⟩ ⊨ ⟨h⟩(⟨args⟩) | ⟨h-numeric⟩(⟨args⟩) | ⟨f⟩(⟨args⟩) | ⟨f-numeric⟩(⟨args⟩)

⟨metadata⟩ ⊨ ⟨data-unit⟩ | ⟨ext args⟩ | ⟨d⟩(⟨ext args⟩)

The above EBNF-rules describe the metadata composition. We defined two groups: the **first group** contains EBNF rules that reflect our metadata classification as provided in Table 4.5. The $\langle data-unit \rangle$ rule is defined as one of the alternative functions, which lists data units. The $\langle map \rangle$ and $\langle fold \rangle$ rules are defined to describe metadata functions from the *Map* and the *Fold* metadata categories, respectively. Additionally, the $\langle map-numeric \rangle$ and $\langle fold-numeric \rangle$ rules are the *Map* and the *Fold* rules that operate only on numeric data. The $\langle h \rangle$ rule defines all metadata functions from the *Higher-Order* metadata group. Whereas the $\langle h-numeric \rangle$ rule defines metadata functions that are exclusively applicable on numeric data. The $\langle d \rangle$ rule comprises all metadata functions belonging to the *Descriptive Data Mining Methods*.

The **second group** consists of the EBNF rules that define the composability of metadata functions. The $\langle metadata \rangle$ rule is specified either as a $\langle data-unit \rangle$ rule, as a $\langle ext-args \rangle$ rule, or as the *Data Mining Methods* $\langle d \rangle$ rule with the $\langle ext-args \rangle$ arguments. The $\langle ext-args \rangle$ rule is formulated either as the sequence of *Higher-Order* $\langle h \rangle$ or $\langle h-numeric \rangle$ rules with $\langle args \rangle$ arguments. The $\langle ext-args \rangle$ rule might be defined as either $\langle f \rangle$ or $\langle f-numeric \rangle$ rules with $\langle args \rangle$. These arguments $\langle args \rangle$ are a sequence of one or more $\langle arg \rangle$ components, while the $\langle arg \rangle$ components correspond to either $\langle data-unit \rangle$, $\langle f \rangle$, or $\langle f-numeric \rangle$, which might take $\langle arg \rangle$ arguments as well. Semantically, $\langle ext-args \rangle$, $\langle args \rangle$ and $\langle arg \rangle$ describe the metadata's compound arguments. Finally, the $\langle f \rangle$ rule is specified as one of two alternative rules, either $\langle map \rangle$ or $\langle fold \rangle$. Similarly, the $\langle f-numeric \rangle$ rule is defined as one of two alternative rules, either $\langle map-numeric \rangle$ or $\langle fold-numeric \rangle$.

For instance, the straightforward metadatum, such as *count of rows*, is composed by applying the following sequence of rules:

1. <metadata>
2. <f>(<args>)
3. <fold>(<arg>)
4. count(<plain>)
5. count(row)

To demonstrate the formal composition methodology, we continue on the z-value-based outlier detection from the Example 4.2.2 and demonstrate four constituents of the *z-value* metadatum.

Example 4.2.3 *The z-values measure is a compound function with four-valued arguments and specified as following :*

$$z = z - value(\mathcal{R}.a_i,\ length(\mathcal{R}.a_i),\ \mu(length(\mathcal{R}.a_i)),\ \sigma(length(\mathcal{R}.a_i))),$$

where the last three arguments are functions. We specify the second argument as the function $length : \mathcal{V} \to \mathbb{N}$ *that maps the set* $\mathcal{V}$ *of string values of the attribute* $\mathcal{R}.a_i$ *to a set of natural numbers* $\mathbb{N}^n$ *that represents the length of the corresponding element* $v_{i,j}$*. The*

last two arguments of the z-value function are mean and standard deviation. They are also compound functions and operate on the distribution values produced by the length function $length(\mathcal{R}.a_i) = (l_1 \dots l_n)$*. For given* $\mathcal{R}.a_i$ *attribute values, the computation of the mean and standard deviation is accomplished as follows:*

$$\mu(length(\mathcal{R}.a_i)) = \frac{1}{n}\sum_{j=1}^{n} length(\mathcal{R}.a_i)$$

$$\sigma(length(\mathcal{R}.a_i)) = \sqrt{\frac{1}{n-1}\sum_{j=1}^{n}(length(\mathcal{R}.a_i) - \mu(length(\mathcal{R}.a_i)))^2} \quad \blacksquare$$

The formal demonstration of the composability of the *outlier*-function, which is based on the *z-value* function, is provided in Table 4.6. This table shows that the *outlier* metadatum adhered to the formal EBNF rules, which were previously formulated.

To summarize, in this section, we analyzed metadata in the context of data quality management. Initially, we built a new mapping between metadata and well-known data quality issues by analyzing error detection methods, which use metadata functions. Based on our mapping, we categorized the metadata functions with regard to the composability of each function. Finally, we introduced EBNF rules that describe the metadata functions, generalize the metadata composability and, more importantly, generate new types of metadata functions.

In the following section, we evaluate the effectiveness of our mapping between data quality issues and metadata. We apply the EBNF rules to construct new metadata for outlier detection heuristics. Finally, we provide a usability study to demonstrate how our mapping and metadata composability rules can support data scientists in developing data cleaning strategy.

4.3 Case Study

This section provides an empirical evaluation of the mapping between data errors and extractable metadata by assessing the accuracy of the metadata-based heuristics. We demonstrate that the mapping between metadata and data quality issues, as shown in Table 4.1, is a useful guideline for rapid prototyping of an error detection strategy. Furthermore, we expose that the generated error detection rules might be used in conjunction with other data cleaning approaches or to be integrated within a more general data cleaning or data integration framework.

Status	Reason
<metadata>	Given
<d>(<ext-args>)	Replace <metadata> by its RHS
outlier(<ext-args>)	Replace <d> by its RHS in <metadata> rule
outlier(z-value(<args>))	Replace <ext-args> by its RHS in <ext-args> rule
outlier(z-value(<arg> {<separator> <arg>}))	Replace <args> by its RHS in <args> rule
outlier(z-value(<plain> { <separator> <arg>}))	Replace first <arg> by its RHS in <arg> rule
outlier(z-value(value {<separator> <arg>}))	Replace <plain> by its RHS in <plain> rule
outlier(z-value(value, <f>(<arg>) {<separator> <arg>}))	Replace <separator> <arg> pair by its RHS in <separator> and <arg> rules
outlier(z-value(value, <map>(<plain>){<separator> <arg>}))	Replace <f> by its RHS <map> resolution and <arg> by its RHS in <arg> rule
outlier(z-value(value, length(column){<separator> <arg>}))	Replace <map> by its RHS in <map> rule and <plain> by its RHS in <plain> rule
outlier(z-value(value, length(column), <f>(<arg>) {<separator> <arg>}))	Replace <separator> <arg> pair by its RHS in <separator> and <arg> rules
outlier(z-value(value, length(column), <fold>(length(column)) {<separator> <arg>}))	Replace <f> by its RHS in <f> and <arg> by length(column)
outlier(z-value(value, length(column), mean(length(column)) {<separator> <arg>}))	Replace <fold> by its RHS in <fold> rule
outlier(z-value(value, length(column), mean(length(column)), <f>(<arg>)))	Replace <separator> <arg> pair by its RHS in <separator> and <arg> rules
outlier(z-value(value, length(column), mean(length(column)), <fold>(length(column))))	Replace <f> by its RHS in <f> and <arg> by length(column)
outlier(z-value(value, length(column), mean(length(column)), std.dev(length(column))))	Replace <fold> by its RHS in <fold> rule

Table 4.6: Formal demonstration that shows that outlier composition matches the formal EBNF description, which is provided in Section 4.2.3. Source [233].

We describe the experimental setup in Section 4.3.1. To showcase the mapping between *data quality issues* and *metadata*, we applied the following strategy: for every dataset, we first determine a set of data quality issues and extractable metadata (Section 4.2), then we create error detection heuristics based on extracted metadata. The implementation of our system is provided in Section 4.3.3. Since real-world data contains multiple error types [2], we evaluate the combination of the error detection heuristics to asses the joint accuracy of all metadata-based heuristics in Section 4.3.4. The user study to assess the usability of our mapping and EBNF-rules for developing data cleaning strategy is provided in Section 4.3.5

4.3.1 Evaluation Metric

To assess the accuracy of the metadata-based error detection rules, we use *Precision (P)*, *Recall (R)*, and *F-measure (F_1)*. *Precision (P)* is the ratio of the correctly identified errors (*tp* - true positives) to the total amount of identified errors (*tp* + *fp*, where *fp* denotes false positives). *Recall (R)* expresses the ratio of the correctly identified errors (*tp*) to the total amount of existing errors (*tp* + *tn*, where *tn* denotes erroneous attribute values which were not found). The harmonic mean of *Precision* and *Recall* is the F_1 score, which is calculated as follows: $F_1 = \frac{2PR}{P+R}$

Implementation Details. We implemented the error detection pipeline using the programming languages Scala and Python. The Scala programming language is used to implement all data pipelines for profiling, heuristics generation, the evaluation methods, and the two combination algorithms, namely *Majority Wins* [252] and *UnionAll* [2]. The Python programming language is used to implement the *eigenvalue-based* technique for heuristic combination [63]. We ran all experiments on a single machine with 2.3 GHz Intel Core i7 processor and 16 GB RAM.

4.3.2 Datasets and Known Data Quality Issues

In order to conduct the case study, we use four real-world datasets. All datasets are summarized in Table 4.7.

The MUSEUM [166] dataset includes information on more than 420,000 artworks from the New York Metropolitan Museum of Art[1]. This dataset contains 48 columns, and the content is related to different data types, such as string or date/time. Known issues in MUSEUM include *missing values, inconsistent information (extraneous data), ambiguous values*, and *wrong word ordering*. To obtain the ground truth, we manually cleaned a subset of 2189 data points and compared it with the dirty version. The MUSEUM dataset contains 52.2% erroneous values.

[1] `https://github.com/metmuseum/openaccess`

	MUSEUM	BEERS	FLIGHTS	ADDRESS
# columns	43	10	9	12
# rows	2189	1265	74k	94k
ground truth	2189	1265	74k	94k
real errors	52.2%	9.2%	61.85%	36.9%
# FDs	3	4	4	1
Memory (dataset and metadata)	2003 MB	942 MB	5357 MB	5787 MB
Metadata computation runtime	2470 ms	1600 ms	3918 ms	6377 ms
Heuristic rules execution runtime	700 ms	520 ms	645 ms	210 ms

Table 4.7: Experimental datasets summary. This summary shows the structure of each dataset, e.g. the number of columns and rows, as well as the ground truth size and the percentage of dirty values, which are contained in each dataset. Additionally, the bottom part includes the number of integrity constraints considered in the experiments, memory allocation, and the runtime in milliseconds for metadata calculation and rules execution on each dataset. We measured the runtime for the execution of all rules. Source [233].

The BEERS [117] dataset was initially taken from the Kaggle platform. It contains data about 1265 US craft beers and 510 US breweries. The BEERS dataset was provided in its clean version. We performed "reverse engineering" of the cleaning process based on the instructions provided by Kaggle. Given these "cleaning" steps, we created BEERS initial "dirty" version. This dataset contains ten attributes which encode breweries and beer identifiers and their names, as well as numerical information about beers and the geographical location of the breweries. Known issues in BEERS include *misfielded values, missing values, wrong data type*, and *missing disguised values*. The BEERS dataset contains 9.2% erroneous values.

The FLIGHTS dataset was created by integrating 38 web sources with the aid of fusion techniques [153] and contains mainly the *date* and *time* data types, and categorical attributes that describe information about gates. FLIGHTS is accompanied by gold standard, also created by the authors [153]. For our case study, we selected a 74k data points that have corresponding ground truth values. Known issues in FLIGHTS are *missing values* and *missing disguised values*. The FLIGHTS dataset contains 61.85% erroneous values.

ADDRESS is a proprietary dataset contains anonymized address data. ADDRESS contains 12 attributes that represent textual and categorical information, such as names, social security numbers, and addresses. The dirty version of ADDRESS contains several error types, such as *missing values, wrong formatting (domain violation)*, and *misfielded values*. In total, the ADDRESS dataset contains 36.9% erroneous values.

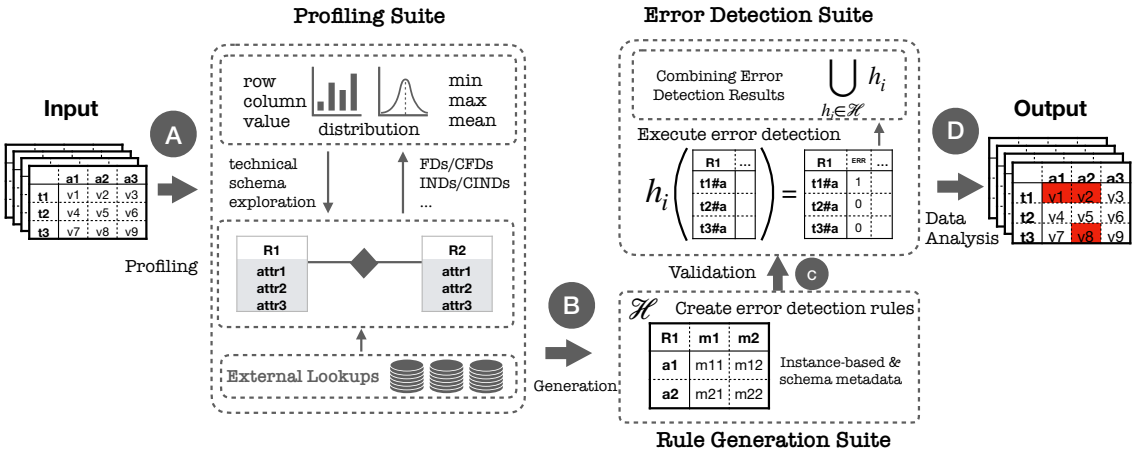

Figure 4.5: Error-Detection Pipeline Implementation. The complete pipeline includes: (A) The profiling step: an input data is analyzed by running different profiling tasks, and complete instance- and schema-related metadata is collected. (B) The rules generation step: gathered metadata is used to construct error detection heuristics. (C) The validation step: The multiple error detection results are combined. (D) The data analysis step produces the aggregated dataset analysis. Source [233]

4.3.3 Error-Detection Pipeline Implementation

In this section, we describe the implementation of the error detection pipeline[2]. Its core architecture consists of two main components, namely (1) the Profiling Suite and (2) the Error Detection Suite. Figure 4.5 schematically depicts the system implementation details. In the following, we explain each of the main components:

Profiling Suite. Initially, we extract the dataset metadata by using dataset profiling functions, such as *value length, null values, histogram,* and *value distributions*. The statistical profiles and schema-based metadata are computed for every attribute. Table 4.8 provides the extracted metadata for all datasets. In our empirical evaluation, we only describe metadata that will appear in the designed heuristics, which should cover the dataset errors from the experimental datasets.

The Profiling Suite has been developed by using the combination of the Metanome [182] framework and custom Spark SQL scripts [14]. We use Metanome as the foundation for data profiling because it integrates data profiling algorithms for computation of statistical metadata and integrity constraints into a common framework. Furthermore, we provide another custom profiling component, based on Spark SQL framework to compute values distributions and descriptive dataset statistics. As shown in Table 4.7, the runtime of the metadata computation is directly proportional to the number of cells in the dataset. Our profiling component computes the following descriptive statistics for each dataset:

[2]The implementation is provided online `http://bit.ly/er-on-metadata`

number of rows, ten most frequent values and their frequencies, percentage of distinct values, number of null values, and *percentage of null values.* Furthermore, several of Metanome's pluggable algorithms to discover the dependencies are used to estimate *normalized functional dependencies.*

Based on the EBNF rules, we generated the following metadata profiling functions, such as *trimmed/winsorized value length distributions, MAD, mean*, and *median.* This generated metadata is implemented by using the Spark SQL framework.

In order to apply the *distant supervision* approach for heuristics creation [198], we use existing knowledge bases and dictionaries to align with the appropriate attributes. For instance, by identifying the *domain* metadata, such as geography-related data types, we incorporate external master data to verify the corresponding attribute values. We use the *master data* for geography-related attributes such as *city, zip code, state, country.*

The *domain/semantic role* metadata includes a set of regular expressions describing attributes such as *e-mail, gender, credit card, zip.* Moreover, we curate a dictionary of default values for different data types, such as *"Not Available", "-", "–", "N/A", "unknown", "null", "1970-01-01T00:00:00Z", "January 1st, 1900".*

Importantly, in this case study, our focus lies outside *integrity constraints*, including *FDs, CFDs*, and *INDs.* This is because, firstly, dependencies-based data cleaning solutions have been already extensively studied by rule-based data cleaning systems, and its effectiveness has already been shown [54, 91, 62, 202], and secondly, in Chapter 6, we describe the specification of the data cleaning rules based on integrity constraints and provide an approach to probabilistic data cleaning by using statistical relational learning.

Rule Generation Suite This component generates error detection rules by using extracted and new metadata. We leverage the mapping between metadata and data errors as a template for heuristics generation.

Error Detection Suite This component executes single heuristics on the appropriate attribute values (see Section 4.3.4). The runtime of the heuristic rule execution is shown in Table 4.7, which demonstrates that the runtime depends on the amount of attributes for which the cardinality-based heuristic is performed because it includes the duplicate values identification sub-routine. Finally, it combines all error detection results. We use unsupervised techniques to combine error detection rules, which we explain later in Section 4.3.4.

4.3.4 Metadata-Based Error Detection Heuristics

In the following, we establish a mapping between known data quality issues and extracted metadata by using the previously-inspected datasets, as shown in Table 4.8. Then, we

		Known Data Quality Issues							
		missing values	misfielded values	wrong data type	default values	domain violation	extraneous data	ambiguous values	wrong word ordering
Extracted Metadata	null values	✓	✓						
	value length distribution		✓					✓	✓
	decimal							✓	
	pattern							✓	
	distinct								
	constancy								
	histogram (frequency distribution)		✓		✓		✓	✓	✓
	data type			✓	✓	✓			
	data class			✓	✓			✓	
	domain/semantic role					✓		✓	
	master data				✓			✓	
	unique column combination							✓	
	functional dependencies	✓	✓			✓		✓	✓
Generated Metadata	trimmed value length distribution		✓					✓	✓
	winsorized value length distribution		✓					✓	✓
	min/max/mean/median on value length distribution		✓					✓	✓
	median absolute deviation (MAD) on value length distribution		✓					✓	✓
	standard deviation on value length distribution		✓					✓	✓
	z-value on value length distribution		✓				✓	✓	✓
	quartile on value length distribution		✓		✓			✓	✓

Table 4.8: Empirical mapping between extracted and generated metadata and data quality issues. Generating new metadata has been performed by using EBNF rules. This mapping reflects concrete datasets: MUSEUM, BEERS, FLIGHTS and ADDRESS. Source [233].

evaluate metadata-based error detection rules and analyze their results for each experimental dataset.

Error Detection Heuristics Provided the guidelines for designing error detection heuristics (see Section 4.1), we demonstrate concrete error detection rules, which are created for all experimental datasets. Table 4.9 provides a summary of the designed heuristics for all experimental datasets. Generally, each heuristic h_i returns a set of error detection results $(r_1, \ldots, r_m)$, where m is the number of all inspected cells in the dataset. Informally, r_i produces one of the following results: {"*error*", "*clean*", "*does not apply*"}. Concretelly, each error detection rule is assigned one value $r_i \in \{1, 0, -1\}$, where 0 indicates that the particular rule does not cover the respective attribute. For example, if one rule is formulated to detect irregularities in numerical data, then on the categorical attributes, this rule produces 0 as a result. The results -1 and 1 denote *"clean"*, respectively *"error"*

for the particular cell v_i. We do not consider 0 cells in the analysis, and only evaluate the error detection heuristics on the cells that are labelled as -1 *"clean"* and 1 *"error"*.

We distinguish between heuristics based on traditional metadata [3] and heuristics that incorporate newly generated metadata by applying our EBNF rules for metadata composition. The first group contains the rule for capturing *illegal values/domain violations*, namely the *valid-data-type* rule, is designed in accordance with the description in Example 4.1.2. Rules, such as *missing-value* and *default-value* are intended to identify either *null* values or *implicit missing* values. One of the common heuristic to detect outliers is to examine the distribution of values and check whether the value is in the tail of the distribution [190]. This heuristic is implemented by the *top-value* rule and it uses the *frequency statistics* metadatum. For the sake of completeness, we also include the heuristics based on *integrity constraints*, such as *functional dependencies*. Generally, this metadata has been extensively studied by rule-based data cleaning systems, and its effectiveness has already been shown in the related work [54, 91, 62, 202]. The violation of the functional dependencies might reveal the consistency and accuracy violations, such as *missing values, misfielded values, domain violation, ambiguous values,* and *wrong word ordering* issues. We designed the *fd-compliance* heuristic by using the *FD compliance* metric to identify values in rows that violate an FD [239]. The *fd-compliance* rule unites multiple functional dependencies for each dataset.

Table 4.9 provides a summary of the designed heuristics. It shows the semantics of error detection rules, along with the applied approach to designing a heuristic, the metadata that is involved in each rule, and possible thresholds. The second group consists of error detection rules, which are based on the generated metadata. The composability of metadata and our proposed EBNF rules will allow designing various heuristics by re-using metadata of different granularity levels as well as by generating new metadata. In particular, the *value-len-z-test* rule is formulated to detect *misfielded values*. This heuristic is described in Example 4.1.1, Section 4.1. To capture *illegal values/domain violations*, we created the *valid-data-type* rule. Its full description is shown in Example 4.1.2, Section 4.1. More importantly, the composability of metadata (see Section 4.2.3) allows us to construct heuristics by re-using metadata of different granularity levels. It is mostly the lower-level or more basic metadata, such as *Map*, or *Fold*, that is suitable for being re-used. For example, the *value-len-evt* rule is created by using the extreme value theory [116]. This rule utilizes metadata such as *value length distribution, std.deviation*, and *mean*. At the same time, the *value-len-z-test* rule is created by using the *value length distribution* and *z-value* metadata. According to the composability of metadata, we re-use previously computed *std.deviation* and *mean* to compute the *z-value* metadata.

Heuristic	Description (Error Condition)	Mapping approach	Metadata	Threshold
missing-value	validating value for $NULL$	▪	null value	-
default-value	check that value is default	[194]	data type	-
top-value	check that value is in the tail of the value distribution	[127, 190]	histogram	from the 11-th frequent value
valid-number	check that the numeric value doesn't match the data type format	[127, 228]	data type format	-
cardinality-vio	check that the attribute $\in$ PK and uniqueness $<$ 100%	▪	PK, uniqueness	-
lookup-attr	check value against the lookup table	[55, 2]	attribute values	-
valid-data-type	check that value doesn't match the domain format (e.g. SSN, zip)	[127, 228]	domain format	-
unused-column	check that attribute values are $NULL$ or are the same	▪	constancy, null value	-
fd-compliance	check that the FD $L \rightarrow R$ holds for any two rows u and v. Suspected violation: $\{u \vert u, v \in \mathcal{D}, u(L) = v(L), u(R) \neq u(R)\}$	[239]	functional dependenies	-
value-len	check that the value length is outside the inter-quartile range	[190]	value length distribution quartiles	1-st and 3-rd quartiles
value-len-Hampelx84	check that length value is an outlier according to Hampel X84 [114]	[114, 150]	length value distribution median MAD	1.4826 [114, 150] (for lower- and upper bounds $bound = median \pm (MAD * 1.4826)$)
value-len-evt	check value length against "extreme values theory" [116]	[116]	value length distribution mean std.deviation	$P > t$ (where t set by user) $P = exp(-exp((-1) * (\frac{(val-mean)}{std.dev}))$
value-len-1-5-IQR	check that value length is outside the 1.5*inter-quartile range	[114]	value length distribution quartiles	1-st and 3-rd quartiles
value-len-z-test	performs the z-value test on value length distribution	[8]	length value distribution z-values	3, described in [8]
value-len-trimmed	check that value length is outside the range $[t.mean - 2 * t.stdDev; t.mean + 2 * t.stdDev]$	[114]	trimmed value length distribution trimmed mean trimmed std.deviation	2 [114]
value-len-winsorized	check that value length is outside the range $[w.mean - 2 * w.stdDev; w.mean + 2 * w.stdDev]$	[114]	winsorized value length distribution winsorized mean winsorized std.deviation	2 [114]

Table 4.9: Experimental heuristics. This table describes the error detection heuristics. Furthermore, for every heuristic, we provide the involved metadata and thresholds. The column *Mapping approach* provides the method used to design the error detection heuristics, which also explains the mapping between data quality issues and the particular metadata. To recap, we used two approaches: the related work approach, where existing methods are reviewed, and the trivial relationship approach (marked as •), where the connection between data errors and metadata is trivially established. Threshold scores, which are taken from the related work, are provided with references. Source [233].

	MUSEUM			BEERS			FLIGHTS			ADDRESS		
Metadata-based heuristics	P	R	F1	P	R	F1	P	R	F1	P	R	F1
missing-value	0.9982	0.8974	0.9451	0.1261	0.0517	0.0733	0.8704	0.3889	0.5376	0.0029	0.002	0.0023
default-value	1.0	0.0101	0.02	1.0	0.0824	0.1522	0.9514	0.0139	0.0273	-	-	-
top-value	0.0116	0.0628	0.0196	0.0652	0.5242	0.1159	0.4519	0.9883	0.6202	0.4414	0.8786	0.5876
valid-number	0.1909	1.0	0.3206	1.0	1.0	1.0	-	-	-	-	-	-
cardinality-vio	-	-	-	0.0986	1.0	0.1795	-	-	-	-	-	-
lookup-attr	-	-	-	0.7257	1.0	0.8411	-	-	-	0.9681	0.6817	0.8001
valid-data-type	-	-	-	-	-	-	-	-	-	0.9979	0.8032	0.89
unused-column	0.9156	1.0	0.956	-	-	-	-	-	-	-	-	-
fd-compliance	0.7257	1.0	0.841	0.2089	1.0	0.3456	0.7899	1.0	0.8826	0.4999	1.0	0.6666
Value length based outlier detection rules												
value-len	0.0553	1.0	0.1048	0.1709	1.0	0.292	0.5279	1.0	0.691	0.6193	1.0	0.7649
value-len-Hampelx84	0.0475	1.0	0.0906	0.1247	1.0	0.2218	0.4978	1.0	0.6648	0.6385	1.0	0.7794
value-len-evt	0.2262	1.0	0.3689	0.0485	1.0	0.0926	0.5146	1.0	0.6795	0.7043	1.0	0.8265
value-len-1-5-IQR	0.1161	1.0	0.208	0.068	1.0	0.1273	0.3966	1.0	0.5679	0.3716	1.0	0.5418
value-len-z-test	0.3012	1.0	0.4629	0.0536	1.0	0.1017	0.231	1.0	0.3753	0.6858	1.0	0.8136
value-len-trimmed	0.1821	1.0	0.3081	0.0946	1.0	0.1729	0.3246	1.0	0.4901	0.589	1.0	0.7413
value-len-winsorized	0.1386	1.0	0.2435	0.2506	0.6526	0.3621	0.4615	1.0	0.6316	0.7096	1.0	0.8302

Table 4.10: Precision, recall, and F_1-measure of single heuristics. The evaluation is performed only on relevant data, meaning that if the attribute is not covered by the error detection rule, then the values of this attribute are excluded from the evaluation. The dashes "-" denote that the rule does not provide any valid results. Source [233].

Evaluating Error Detection Heuristics. Real-world datasets enclose diverse error types. Hence, providing different heuristics is essential for capturing as many error types as possible [2]. We first study heuristics that are designed to capture at least one error type. Next, we will consider the group of heuristics for identifying outlier because an *outlier* might subsume different errors, such as *misfielded values, ambiguous and extraneous data, wrong word order* or *illegal values*. The results of all individual heuristics are shown in Table 4.10.

First, we study the following heuristic rules: *missing-value, default-value, top-value, valid-number, cardinality-vio, lookup-attr, valid-data-type, unused-column*. Each of these rules is designed to identify at least one error type. Common issues in the MUSEUM and FLIGHTS datasets are *missing values*. Hence, the corresponding rules, such as *missing-value* and *unused-column*, mainly use *null value* metadata. The evaluation of the above two rules provides the best results compared to the remaining rules, by producing 0.95 as an F_1-score for the MUSEUM dataset. To identify outliers, we created a rule that analyzes the FLIGHTS histogram of values and their frequencies. This rule produced an F1-score of 0.62 – the highest among all datasets. The primary issue of the BEERS dataset is the *wrong data type*, meaning that numeric columns include non-numeric values. Hence, we formulated the *valid-number* rule to analyze numeric columns by using the *data type format* metadata. The above rule results in a 1.0 F_1-score. The second best result – 0.84 F_1-score for capturing errors in a geographical location - is achieved by the *lookup-att* rule. This rule employs both *data type* metadata and the *master data*, which is used for comparing values in the corresponding geographic columns. As the ADDRESS dataset contains information about people and their addresses, two rules, *lookup-att* and

the *valid-data-type*, achieves the best results of all rules. The *lookup-att* provides a 0.8 F_1-score, and the *valid-data-type* rule achieves a 0.89 F_1-score. The *fd-compliance* rule identifies multiple data quality issues, such as *missing values, misfielded values, domain violation, ambiguous values,* and *wrong word ordering*. This rule achieves the best result - 0.88 F_1-score on the FLIGHTS dataset among all datasets.

Next, we consider heuristics that are formulated to spot *outliers*. Because *outliers* are often an indication of irregularities in our data [17, 8]. We consider the problem of outlier detection meant to identify exceptions such as *misfielded values, ambiguous and extraneous data, wrong word order* or *illegal values* in our datasets. Therefore, we also created a group of outlier detection heuristics on value length distributions for all dataset columns. All of these rules contain the *error condition* to detect an *outlier* by producing 1 as a result; otherwise, every rule produces 0, which means that the rule cannot decide whether the corresponding cell is an *error* or not. By analyzing both tails of the distribution, these rules are intended to flag irregularities in data. To specify the range for the *outliers*, we used metadata, such as *quartiles, median, MAD, mean,* and *standard deviation*, which were computed on the numerical representations of the attribute values (e.g. *value length*). All outlier detection rules for all experimental datasets show the same behaviour: these rules produce high recall and low precision. This means that although they are able to spot all errors, they also falsely select clean values as *errors*. Outliers are only one type of errors that appear in the datasets. Hence, even if we execute different outlier detection rules, we still cannot cover all possible errors in the experimental datasets.

Data Error Coverage. To evaluate the efficacy of each error detection heuristic, we analyzed two datasets – MUSEUM and BEERS – and studied *how heuristics cover each particular error type*. We can distinguish particular data errors and use these labels to analyze each heuristic because we either manually cleaned or inserted errors in the above datasets. The coverage numbers in percentage are shown in Table 4.11. These numbers confirm our hypothesis that whenever a data error can be considered as an extreme value or outlier, then outlier detection heuristics will capture the majority of such errors.

To capture *misfielded values* in the BEERS dataset, we implemented heuristics by using schema-based metadata such as *domain/semantic role*. These heuristics outperform *outlier-based* heuristics. In particular, the *misfielded values* in BEERS appear on geographic attributes and therefore are best covered by the *lookup-attr* rule, which uses the *domain* and *semantic role* metadata.

For example, in the MUSEUM dataset, we consider *ambiguous, extraneous data, misfielded values*, and *wrong word order* as outliers, and the heuristics capture extreme values in the value length distribution. *Missing values* are captured mainly by using the heuristics that include the *null values* metadata, and the results of the *missing-value* rule (precision, recall, F_1, as well as errors coverage) are confirming this claim.

	MUSEUM						BEERS			
	ambiguous data	misfielded values	missing values	extraneous data	default values	wrong word order	misfielded val.	wrong data type	default values	illegal values
missing-value	-	-	100.0	-	-	-	50.0	-	-	-
default-value	-	-	-	-	35.71	-	-	-	100.0	-
top-value	85.23	100.0	-	4.29	35.71	12.9	50.0	90.62	100.0	20.85
valid-number	93.25	-	-	-	64.29	-	-	100.0	100.0	100.0
cardinality-vio	-	-	-	-	-	-	11.02	10.39	5.73	7.13
lookup-attr	-	-	-	-	-	-	100.0	-	-	-
unused-column	-	-	85.69	-	-	-	-	-	-	-
fd-compliance	6.33	-	8.96	10.0	-	-	97.7	-	-	-
value-len	100.0	100.0	-	100.0	35.71	100.0	22.05	20.49	100.0	41.24
value-len-Hampelx84	100.0	100.0	-	100.0	35.71	100.0	22.05	20.49	100.0	30.4
value-len-evt	16.46	72.0	-	80.95	-	-	3.94	-	-	-
value-len-1-5-IQR	100.0	72.0	-	100.0	100.0	100.0	21.26	100.0	100.0	30.4
value-len-z-test	10.13	72.0	-	76.19	-	-	2.36	-	-	-
value-len-trimmed-range	100.0	96.0	-	98.1	-	100.0	6.69	17.17	100.0	-
value-len-winsorized-range	16.46	72.0	-	81.9	-	100.0	3.94	-	-	-

Table 4.11: Error coverage by heuristics in percent. Every issue for the MUSEUM and BEERS datasets has been distinguished, and we determined how each of the heuristics covers the particular data error. Please note that in this study, we consider only datasets, where the categorization of data errors was possible. "-" dashes denote that the heuristic has no coverage for the particular error type. Source [233].

As the problem of *misfielded values* is usually caused by shifting attribute values, the *null value* metadata might be a reliable indicator for this data quality problem. The 50%-coverage of the *missing-value* rule is the support for the above claim. To identify the *wrong data type* error, either the *valid-number* rule or the *top-value* rule can be used. The former rule uses the *data type* metadatum, whereas the latter uses the *histogram*. Both heuristics deliver good results for the identification of *wrong data type* errors. Similarly, the *FD-compliance* rule with 97.7% reliably covers this type of errors because the RHS parts of functional dependencies $brewery - id- > state$, $city- > state$ cover the above mentioned geographic attributes.

Since *default* or *illegal value* data errors appear in numeric attributes in the BEERS dataset and contain alphanumeric instead of numeric characters, these errors are best captured by the *valid-number* rule, which is formulated by using the *data type format* metadatum. Because the percentage of such errors is low (less than 2%), they are fully identified by the outlier-based heuristics.

Generally, the data errors coverage empirically shows that the mapping between metadata and data quality issues, as shown in Table 4.8, is a useful guideline for rapid prototyping of an error detection strategy. The proposed EBNF grammar can be used to

generate any other possible metadata that has not been explicitly mentioned by Abedjan et al. [3]. In our case study, we applied the proposed EBNF grammar to generate metadata for outlier detection heuristics. Hence we demonstrated the practical applicability of the composability concept to generate new metadata. Furthermore, while our heuristics generate many false positives, they are simple to formulate and can be used in conjunction with other heuristics and data cleaning approaches.

Aggregating Heuristics Because datasets contain various types of data errors [232, 2, 82], a variety of error detection methods need to be used in conjunction [223, 160] to support the definition and construction of more general data cleaning frameworks. The previously-defined heuristics capture a particular type of error in data; therefore, the results of these rules need to be aggregated to a final output. To embrace all possible data error types in a dataset, we apply several error detection rules, and then combine their results in an unsupervised way. We propose using unsupervised combining approaches because training sets with predefined labels are not always available, and the creation of the labels is usually an expensive labour process [232, 198]. Please note that combining error detection strategies by using supervised approaches is provided in Chapter 5.

Every combination approach takes a matrix $\mathbf{H}$ as input, where each column $h_i = (r_1, \ldots, r_m)$ contains the error detection results of the respective heuristic h_i for all dataset values. We focus on three representative approaches for the unsupervised combination of multiple results:

Majority vote. [184, 191] This is the combination method that marks a dataset value as an *"error"* if the majority of the values from the tuple $(h_1 \ldots h_n)$ are set to *"error"*.

UnionAll. [2] This method flags a dataset value as an *"error"* if at least one of the applied error detection heuristic has indicated the value v_i as *"error"*, otherwise the value is labelled as *"clean"*.

Eigenvalue-based technique. Dalvi N. et al. [63] proposed two approaches based on spectral analysis algorithm which takes the sparsity of the labeling matrix in both algorithm design and theoretical analysis into account. This method originates from the area of crowdsourcing and addresses the problem of how to combine labels from multiple annotators with various levels of expertise. Our experimental settings are similar to the aggregating crowdsourced binary labels. Therefore we applied this method by assuming our error detection heuristics as "crowd workers". In particular, we re-implemented an algorithm[3] that estimates a dataset cell as either *erroneous* or *clean* for an arbitrary *heuristic-dataset cell* assignment matrix, which is created by

[3]The implementation is provided as a Jupyter notebook `http://bit.ly/eigen-val-aggregation`

	MUSEUM			BEERS			FLIGHTS			ADDRESS		
Combined metadata-based heuristics	P	R	F1	P	R	F1	P	R	F1	P	R	F1
UnionAll	0.5344	1.0	0.6965	0.0946	1.0	0.1729	0.525	1.0	0.6885	0.3385	1.0	0.5058
Majority wins	0.1145	9.0E-4	0.0018	0.25	0.0069	0.0135	0.3089	0.0053	0.0103	0.8755	0.0277	0.0537
Eigenvector-based technique: Alg 1 [63]	0.5035	0.9087	0.6480	0.0931	0.9597	0.1698	0.5082	0.9490	0.6619	0.3471	0.9750	0.5119
Eigenvector-based technique: Alg 2 [63]	0.5035	0.9188	0.6506	0.2023	0.3503	0.2564	0.0	0.0	0.0	0.3434	0.9365	0.5025

Table 4.12: Precision, recall, and F_1-score of a combination of error detection rules. *UnionAll* is expected to perform best because of the different error detection rules coverage: each error detection rule is responsible for one specific irregularity in the data. Source [233].

the concatenation of all error detection rules. The "expertise" of each heuristic varies, hence a heuristic provides a result for a subset of dataset cells.

Results of Heuristic Combinations. Table 4.12 provides the results of all combination techniques. The *UnionAll* and both algorithms from the *Eigenvalue-based technique* deliver almost identical results, with high recall scores – 1.0 for all datasets. By applying the above combination techniques, we achieved an F_1-score of 0.69 for MUSEUM, 0.17 for BEERS, 0.68 for FLIGHTS and 0.51 for ADDRESS. Despite the high recall, the precision for some datasets is still low, roughly achieving 0.5%, which causes a decrease in the F_1-score. Because error detection rules are created to capture one type of data error, these rules reveal their "expertise" only for a subset of the dataset cells. Therefore, the *UnionAll* and *Eigenvalue-based* techniques are appropriate to combine heuristics. Although both methods deliver similar results, the *Eigenvalue-based technique* is algorithmically more complex [63], compared to the *UnionAll* method. Following the *no free lunch theorem* [35], there are no generally best models and the preference should be given to the least complex models; consequently, we prefer the latest and most straightforward combination technique. The *Majority Vote* approach provides insufficient results because it takes the class label that receives the most significant number of votes as the final result. This functionality contradicts our setting where just a few heuristics can detect a particular error type.

To conclude, while our heuristics generate many false positives, they are straightforward and might be rapidly implemented. Furthermore, they can be used in conjunction with other heuristics and data cleaning approaches, to be integrated within a more general data cleaning or data integration framework.

4.3.5 Usability of Metadata Mapping and EBNF Grammar

In this dissertation, we also interested in understanding whether our mapping and metadata composability can help data scientists in developing data cleaning strategy because data science projects are unique and it is challenging to find one data cleaning system to *"fit-it-all"* [142, 223]. Primary, our mapping targets anyone, with little to no expertise,

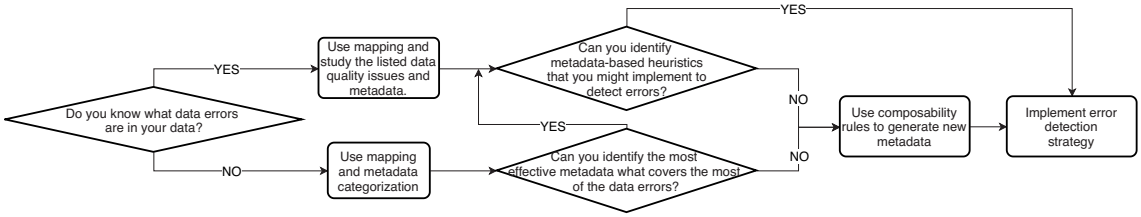

Figure 4.6: A flowchart of defining data cleaning strategy. This flowchart shows how to use the proposed mapping between data errors and metadata, categorization, and composability rules for new metadata generation. Source [233].

to perform data quality assessment when provided with a template that based on our mapping between metadata and data quality issues, and metadata composability rules.

We conducted an interview study[4] of 8 data scientists and data engineers. The interview consists of 14 questions to be completed within 20 minutes. The initial four questions inspect the expertise of each participant, which revealed that that 7 interviewees are performing data preparation as their daily task. The next 9 questions are designed as tasks, where our mapping and composability rules should be used to develop a data cleaning strategy. We analyzed the correctness of responses, which is an indicator for the usefulness of the mapping. Finally, we asked a general question about the practical value of our mapping. Mainly, our interview study results show that:

1. Provided with the mapping between metadata and data errors, data scientists were able to write a program/function based on one of the heuristics provided in the mapping to identify particular data errors. Particularly, 5 out of 9 tasks were correctly solved by *all* participants and the remaining four tasks were correctly solved by at least four participants.

2. Provided with the composability rules, data scientists were able to create new metadata to identify outliers in a given dataset. For example, 7 of 8 responses constructed an outlier detection method by composing *z-values* from the distribution of value length; and

3. Provided with the mapping between metadata and data errors, *all* data scientists report a speed up the process of defining the data cleaning strategy, because they use the above mapping as a template and adapt it for their needs.

Each interview was started with a short tutorial about how to apply our mapping, categorization, and composability rules to create a data cleaning strategy, as summarized in Figure 4.6.

Since data cleaning is an *ad-hoc* process, we designed the interview questions to observe how data scientists address data quality problems if they are equipped with our

[4]The interview questions are available online: `http://bit.ly/dc-user-study`

categorization of metadata and the mapping between metadata and data quality issues. For example, the responses to the question *"provided with the mapping between data quality issues and metadata, what heuristic would you implement as first to detect "domain violation" data error in the dataset in Table 4.4?"* All of the interviewees were able to identify the data quality issue in our mapping and select one metadata-based heuristic (*domain and semantic role*) to implement.

Considering that real data errors might take a different form, it is crucial to identify a category of the error or at least what data quality dimension such error violates. In response to the interview question *"what data quality issue from the mapping would you select to design your solution for the "default values detection?"* four of 8 of responses have correctly identified the related error type, such as *"missing values"*. Hence, they would be able to implement the appropriate errors handling by selecting the *"missing values"* column in our mapping (see Table 4.1).

Next, we provided users with a dirty dataset, as shown in Table 4.4, and asked to specify whether they can write a script for error detection based on one of the heuristics provided in the mapping to identify *"misfielded value"* and how many of them can they implement? As a response, all interviewees were able to implement the above heuristics, and the amount of the heuristics ranges from one to ten, where 14 error detection rules are initially specified in the mapping. This result shows that if data scientists are aware of error types, they were able to implement a data cleaning strategy for this particular error with the help of our mapping.

Often, data preparation includes processing data of unknown quality [2], and it is crucial to be able to define data cleaning routine for any datasets. To examine, how users would proceed in the above setting, we asked them to answer the following question *"Provided with the mapping between data quality issues and metadata, what is your data quality assessment strategy by using that mapping?"* All of the interviewees were able to specify such a data quality assessment strategy. The responses include the following strategies:

- *"Use outliers and clustering to detect most of the issues"*,
- *"Use a histogram to check for outliers"*, or
- *"I will pick metadata that cover more errors first"*.

More experienced users make use of the mapping data errors to the data quality dimensions and direct their data cleaning as *"applying all strategies noted in relation to data errors that are related to completeness and consistency"*.

To get the overall impression about the usability of our mapping, we asked a final question about *"how would the categorization of metadata and the mapping between metadata and data quality issues help or assist data scientists in their daily job for data quality*

assessment?". In the following, we provide all received answers to the above interview question, except for one participant, who missed this question:

1. *"It could help to have better insight about the data and that's help a lot."*
2. *"It makes finding the right tool and heuristic much easier for data scientists to clean the data without struggling much with the data error types and cleaning solutions."*
3. *"This categorization and the mapping is a great literature survey across all current data cleaning approaches and a great help to newcomers in the data cleaning field."*
4. *"It is very useful. It can help in most general problems. But we have to analyse our specific problem when using the mapping as reference."*
5. *"It would be very useful and helpful as well as time saving as the mapping provides a quick and easy way to get strategies for different errors based on the metadata. This is especially useful for people who are newer to data curation and don't know the best test for different cases."*
6. *"It's certainly a good overview over current state error detection and what methods to apply. On a purely theoretical basis, this would be quite helpful, especially for data scientist not familiar with the intricacies of data cleaning.... there are instances where a high value dataset needs to be imported and cleaned and then a overview as presented could be extremely helpful to make sure, no aspect is forgotten and to make sure, the best methods are applied (correctly)."*
7. *"Technically it would help a lot for having a good data quality in daily job."*

Hence, from analyzing the user study responses, we claim that our mapping is a useful tool for defining the data cleaning strategy. In the following, we outline the usability aspects of applying our work while writing custom scripts for data quality assessment [142].

As data scientists report, data assessment routine includes the development of ad-hoc scripts [142]. According to our survey, Python, Scala/Spark, SQL, and R are popular scripting languages to accomplish the above tasks. As elaborated in Section 4.3.3, we implemented the data quality assessment pipeline by using Apache Spark SQL framework [14]. In the beginning, we use our mapping as a template and generate a set of error detection heuristics as *User Defined Functions (UDF)*. Each UDF takes data unit and metadata as parameters, and returns $\{-1, 0, 1\}$ integer values, which semantically denote $\{$"*clean*","*not applicable*","*error*"$\}$ as a result of applying the error detection heuristic on data. It is important to note that implemented UDFs are generated once and used multiple times for all datasets. Given the above set of error detection UDFs, we create an application- and dataset-specific data validation routine. That means, if we aware of

one kind of errors in the dataset, we target these errors with specific UDFs from the pool of heuristics.

In this section, we analyzed the usability of design and implementation of metadata-based tools for assessing data quality.

4.4 Summary

In this chapter, we studied the intrinsic connection between metadata and data errors. Furthermore, we established a mapping that reflects the connection between data quality issues and extractable metadata, using qualitative and quantitative techniques. Additionally, we presented a new metadata classification and taxonomy based on a closed grammar that can also maintain all complex metadata. We provided a systematic study for using metadata in data quality management by showing the *composition* of the metadata functions. We created a *mapping* between metadata and data quality issues, such that it can provide a basis for understanding the connection between dirty data and extractable metadata. Then, we *categorized* metadata according to the *composability* notion. Based on the above categorization, we *generalized* the metadata composability as a set of EBNF rules, which enable data scientists to generate new metadata. Now, provided with our new mapping, new effective techniques for the coverage of data errors by using metadata can be developed. Practically, as our usability study demonstrated, these heuristics are straightforward yet efficient and can be used in conjunction with other error detection strategies for rapid prototyping of data cleaning solutions. We also provided an empirical study to demonstrate the effectiveness and usability of our mapping by assessing metadata-based heuristics for error detection on real-world datasets.

In the following Chapter 5, we use the elaborated connection between data quality issues and metadata and provide an approach for error detection based on ensemble learning methods, which are enhanced with data profiling information.

5

Supervised Error Detection with Metadata

Evaluating the quality of a dataset is a critical first step in every data science and analytic workflow [186, 229]. Data is usually integrated from different sources with different degrees of reliability [75, 74]. Hence, the final dataset often suffers from low accuracy [153], revealing errors such as *outliers, duplicates, missing values*, and *inconsistencies*. The origin of these data quality problems might also go back to several reasons, such as:

- Information extraction from Web sources [213, 225, 153];
- Data integration from heterogeneous data sources of varying reliability [153, 223, 70, 106, 99];
- Manually or programmatically inserted erroneous entries (e.g. into the Web-forms) [28, 89].

With an increasing amount of digitally collected data, the above reasons might lead to the decline of the overall data quality [161]. To address this, in the last two decades, research in data quality management provided many data cleaning approaches, algorithms, and systems [62, 190, 91, 245, 54, 131]. These algorithms were designed to detect and repair specific error types. One possible way to deal with data errors is to apply multiple data cleaning strategies, such as outlier detection [8], missing value imputation [245], or formulating data cleaning rules [62, 91]. However, utilizing all possible data cleaning strategies simultaneously produces a large number of detection results. These results also include many false positives, because not all error detection strategies are equally suitable for each dataset. Example 5.0.1 demonstrates the above problem on real-world data.

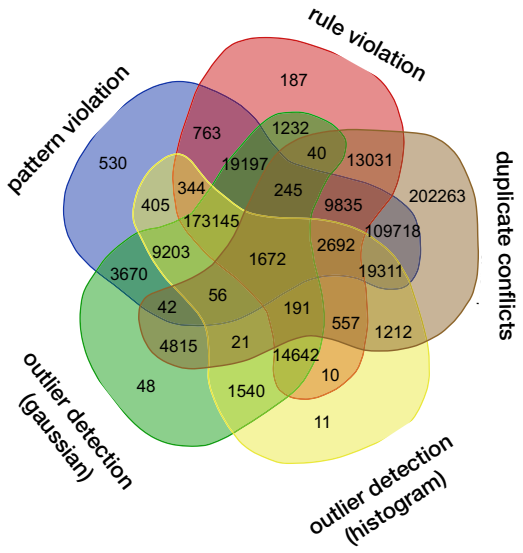

Figure 5.1: Result overlap of four different error detection strategies on the FLIGHTS dataset. Each value represents the number of correctly detected errors by the set of overlapping systems. Source [232].

Example 5.0.1 *We applied different error detection strategies, implemented by several systems, on the real-world dataset* FLIGHTS *[153] and visualized their correct error detection results in Figure 5.1. Each value represents the number of correctly detected errors by either each system or the set of overlapping systems. This visualization shows that these strategies overlap in their results, and none of them fully covers all possible errors in a dataset. Additionally, as each algorithm contributes to the overall results, it is not trivial to optimally combine their results.* ■

Triggered by the motivation and the example above, the research challenges addressed in this chapter are the following:

Data Quality Issues: Datasets reveal multiple data errors and these data quality issues influence each other. The problem arising here is how to cover all possible errors in data.

Error Detection Strategies: Running all available data cleaning solutions might produce a large number of system errors - *false positives*.

Optimal Ensemble Methods: Abedjan Z. et al. [2] proposed an approach that states that *"by assessing the overlap of errors detected by the various tools, it is possible to order the application of these tools to minimize false positives"*. This approach assumes human-in-the-loop because ordering strategy must be dataset-specific. Therefore, the next challenge is to determine the optimal combination method for error detection that is specific for each dataset.

Augmenting Error Detection with Metadata: The effectiveness of the application of each error detection technique is dataset-specific [2]. As shown in Chapter 3, metadata delivers structured information that describes the information resource. To enable the dataset-specific error detection, we propose to augment the combining methods with metadata.

To address these challenges, we developed a holistic error detecting framework which unifies the results of multiple error detection algorithms and uses the dataset's metadata. The *goal* is to combine the set of error detection systems that maximizes the overall recall and precision, aggregated by the harmonic mean F_1. To achieve this goal, we propose two learning-based methods, built on ensemble learning approaches [252]. Our first approach combines the results of all existing error detection solutions. As filtering the most effective combination of such approaches can enhance the overall error detection outcome, the second method selects relevant error detection approaches before combining them.

In this chapter, we present an approach for error detection based on a combination of supervised machine learning methods and data profiling. In particular, we propose a technique that (1) effectively combines multiple error detection strategies; (2) considers the metadata for the error detection; and (3) selects the most effective data cleaning solutions for the particular dataset. This chapter is organized as follows:

1. The implementation of our framework for error detection is described in Section 5.1.
2. In Section 5.2, we propose ensemble strategies using state-of-the-art learning approaches. These dataset-specific strategies aggregate results from different error detection systems.
3. A methodology to select the most effective systems and the accumulation of their results by using learning approaches is provided in Section 5.2.6.
4. In Section 5.3, we provide the holistic aggregation of the error detection algorithms that is enhanced by instance- and schema-based metadata and present the set of classification features for this purpose.
5. In Section 5.4, we describe experimental results of our aggregation strategies and demonstrates the efficacy of the accumulated results from the error detection systems.
6. Finally, we summarize our approach to combining error detection strategies in Section 5.5.

5.1 Error Detection Framework

In this section, we outline the implementation of our approaches. Figure 5.2 illustrates the overall architecture of our system. The proposed system aggregates constituent error

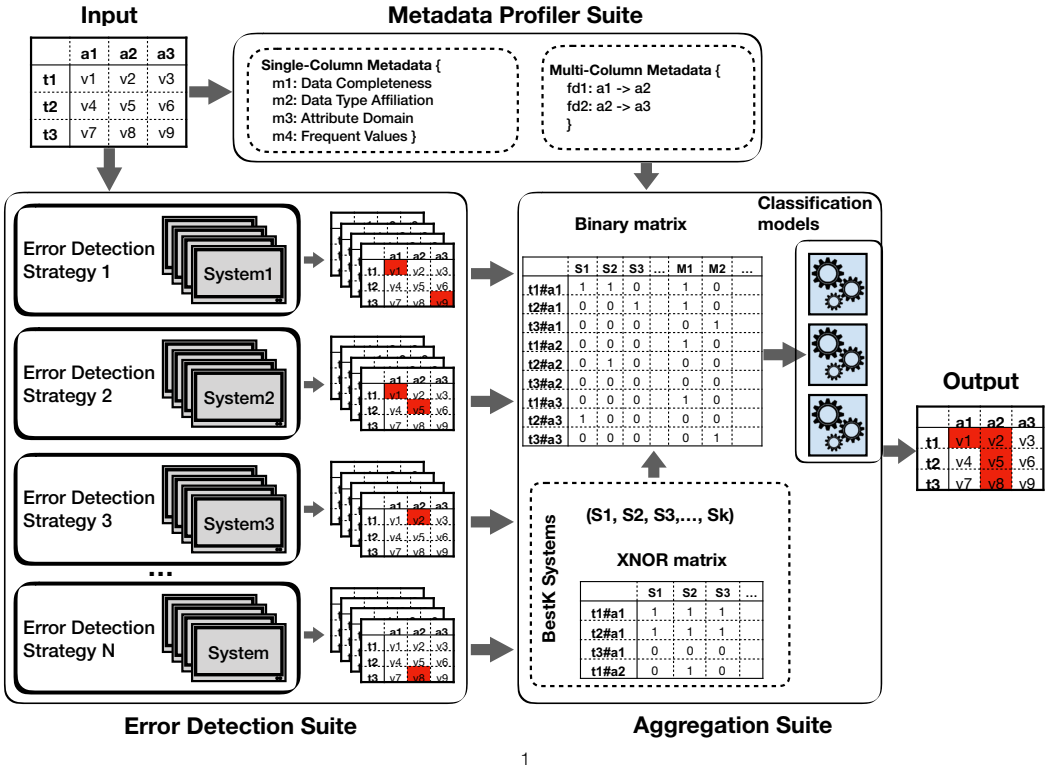

Figure 5.2: The architecture of the system for aggregation of the error detection algorithms. Source [232].

detection systems and consists of three components: 1) an **Error Detection Suite**, which includes pluggable error detection systems that function as black boxes to our system. 2) a **Metadata Profiler Suite**, which extracts various metadata categories, and 3) an **Aggregation Suite**, which combines the output of the error detection suite and the profiler. In the following, we describe each of the components.

Error Detection Suite. The error detection suite includes various error detection systems that can be independently plugged in. Each of these cleaning systems is considered a black box to the enclosing aggregation system. The error detection suite assumes data cleaning algorithms for detecting various data quality issues, such as pattern violation detection, rules violation detection, duplicate-based error detection, or outliers detection. Generally, the entire system is designed to accept an arbitrary algorithm.

Specifically, we include the *pattern violation detection strategy*, which is responsible for the identification of syntactic pattern violations, misspelled values, formatting rule violations, wrong semantic data type affiliation, or misfielded values. This component

focuses on single-source attribute-level error detection [195] and its primary goal is to discover violations of the syntactic and semantic constraints.

The *rule violation detection strategy* is responsible for capturing inter-attribute inconsistencies. Usually, a number of integrity constraints need to be provided to define data cleaning rules [1, 62, 91]. Hence, the initial step in the rule violation detection component is the specification of the set of integrity constraints. These constraints are either known to the data owner or can be extracted by existing metadata profiling systems [183, 182, 23].

The *duplicate conflicts resolution* detects conflicting attributes in duplicates. In our framework, we include the duplicate detection algorithm of the NADEEF [62] system. The duplicate-based system records the exposed data conflicts as potential errors.

The *outlier detection strategy* captures various data quality issues, which can be categorized as outliers. For example, data quality issues, such as *misspellings, extraneous data, incorrect data*, or *wrong word order* might be considered as *outliers*. We utilize the DBOOST system [190], which provides two general outlier detection methods based on *Gaussian* distribution and *Histograms*.

The Metadata Profiler Suite. This auxiliary component acquires general metadata through a data profiling step [2]. The profiler is applied to each dataset independently to the error detection suite. We extract the full set of metadata categories, including instance- and schema-based metadata (see Section 2.2). This suite generates a metadata matrix M with the tuples, which is used by the Aggregation Suite to augment the error detection strategies combination.

The Results Aggregation Suite. Finally, we aggregate the results from each error detection strategy by augmenting it with the metadata information. Concretely, each cleaning subsystem outputs one binary vector $\mathbf{e}^{(j)}$ indicating whether or not a cell is an error by the particular data cleaning strategy j. All yielded vectors constitute the binary matrix E, which is then horizontally concatenated with the metadata matrix M, which is an input for the Result Aggregator Suite. The output of the combination is also a binary vector $\mathbf{e}^{out}$. This suite implements the aggregation strategies of the data cleaning methods. The algorithms behind these aggregation methods are presented in the next section.

5.2 Error Detection as a Classification Task

In this section, we present our approach of combining the results of various data cleaning systems. Initially, we formalize the problem of the error detection combination. Then we explain error classification algorithms, and propose holistic strategies to combine existing data cleaning systems. Finally, we develop an approach to select and aggregate the most effective subset of data cleaning solutions.

5.2.1 Error Detection Formalization

We begin with the formalization of the problem of combining error detection strategies. We consider a dataset instance $\mathcal{D}$ with a schema $\mathcal{S}$. Let $attr(\mathcal{S})$ be the set of attributes in $\mathcal{S}$. Each tuple $t \in \mathcal{D}$ consists of cells. We define the a cell of the tuple t of the attribute $i \in attr(\mathcal{S})$ as $t[i]$. The value of each cell is $v_{t,i}$. We specify $\tilde{v}_{t,i}$ being the true (or clean) value of the cell at the t and i coordinates. A cell $v_{t,i}$ in the dataset $\mathcal{D}$ is determined as an *error* if $v_{t,i} \neq \tilde{v}_{t,i}$.

Let $\mathcal{T}$ be the set of error detection approaches that are independently applied on $\mathcal{D}$. The $s_j \in \mathcal{T}$ is the j-th error detection system with $j \in \{1 \dots |\mathcal{T}|\}$. The result of the j-th error detection system is the set of cells $\mathcal{E}_{s_j}$. We define $e_{t,i}^{(j)} \in \mathcal{E}_{s_j}$ as the output of the j-th system on the i-th cell of the tuple t of the attribute $i \in attr(\mathcal{S})$. Hence, each cell in $\mathcal{D}$ is linked to an output of each error detection system, and the complete output of error detection systems is denoted as a tuple $(e_{t,i}^{(1)}, e_{t,i}^{(2)}, ...e_{t,i}^{(|\mathcal{T}|)})$. The values of $e_{t,i}^{(j)}$ are assigned as follows:

$$e_{t,i}^{(j)} = \begin{cases} 1 & \text{if the j-th system identified } (t,i)\text{-th cell as an error} \\ 0 & \text{otherwise} \end{cases}$$

To holistically combine results of multiple error detection strategies and label each value of the tuple as "clean" or "error", we suggest to build a prediction model that is based on the performance of each data cleaning system. One way to combine the results of various error detection methods is to perceive this problem as a classification task. The result of the classification task will be the combination of applied error detection systems. The general goal is to learn a mapping from the inputs $(\mathcal{E}_{s_1}, \mathcal{E}_{s_2} \dots \mathcal{E}_{s_{|\mathcal{T}|}})$, $s_j \in \mathcal{T}$ to the output $\mathbf{e}^{out} \in \{0, 1\}$. We suggest to train an error classifier that takes error detection results from all systems as features, and classifies each cell $v_{t,i}$ in the dataset $\mathcal{D}$ into two classes $\mathcal{Y} \in \{0, 1\}$, where $y = 1$ denotes an error and $y = 0$ indicates a "clean" value.

5.2.2 Error Classification Algorithms

In the following, we describe the error classification algorithms, which are used in this chapter, namely **Naive Bayes, Neural Network,** and **Decision Tree**. The first is the **Naive Bayes Classifier**, which predicts the probability of an error given the input tuple from the data cleaning systems. Given the corresponding labels $\mathcal{L} \in \{0, 1\}$, the *Naive Bayes* model predicts that $\mathbf{e}$ belongs to the class of $\{0, 1\}$ having the highest posterior

probability conditioned on $\mathcal{E}$:

$$\mathbf{e}_{nb} = \arg \max_{\mathcal{L}_i \in \{0,1\}} P(\mathcal{L}_i|\mathcal{E}), \text{ where}$$
$$P(\mathcal{L}|\mathcal{E}) = \prod_{i=1}^{|\mathcal{T}|} P(\mathcal{L}|e^{(i)})$$

The predicted label from the given feature vector $\mathbf{e}$ is therefore the class $\mathbf{e}_{nb} \in \{0, 1\}$ for which the $P(\mathcal{L}|\mathcal{E})$ is maximal.

The next classifier is the **Neural Network Classifier** and consists of K multiple layers of nodes, where each node in the input layer is a value of the feature vector $\mathbf{e} = (e^{(1)}, e^{(2)}, ...e^{(|\mathcal{T}|)})$.

$$f_{input}(\mathbf{w}_{init}^T \mathbf{e} + \mathbf{b}_{init}) = \frac{1}{1 + \exp(-\mathbf{w}_{init}^T \mathbf{e} + \mathbf{b}_{init})}$$

The K-1 intermediate layers of the network map inputs to the outputs by applying the linear function of the inputs, the corresponding node weights $\mathbf{w}_i$ and bias values $\mathbf{b}_j,\ j \in K$, for each layer, and applying an activation function (see the definition above):

$$y_{\mathbf{e}} = f_k(\dots f_2(\mathbf{w}^T f_{input}(\mathbf{w}_{init}^T \mathbf{e} + \mathbf{b}_{init}) + \mathbf{b}_2) \cdots + \mathbf{b}_k)$$

The number of nodes of the output layer corresponds to the two classes: $\mathbf{e}_{nn} \in \{0, 1\}$ and are computed by using the softmax [29] function:

$$\mathbf{e}_{nn} = \arg \max_{\mathcal{C}_{nn} \in \{0,1\}} P(\mathcal{C}_{nn}|y_{\mathbf{e}}), \text{ where}$$
$$P(\mathcal{C}_{nn} = i|y_{\mathbf{e}}) = \frac{\exp(\mathbf{w}_i^T f_{k-1} + \mathbf{b}_k)}{\exp(\mathbf{w}_0^T f_{k-1} + \mathbf{b}_k) + \exp(\mathbf{w}_1^T f_{k-1} + \mathbf{b}_k)}$$

where $k \in K$ denotes the k-th layer of the network and $i \in \{0, 1\}$.

The **Decision Tree Classifier** algorithm requires the following parameters: 1) Dataset $\mathcal{E}^{train}$ and their associated labels $\mathcal{L}^{train}$; and 2) *Attribute selection* method to partitioning $\mathcal{E}^{train}$ data points into *error* or *clean* classes. The *Decision Tree* is constructed by iteratively splitting the features vector $\mathcal{E}^{train}$ into two branches: *errors* and *non-errors*. The splitting criterion depends on the notion of *Information Gain* [107] $IG(\mathcal{E}^{train}, s)$, which is used as measure to partition $\mathcal{E}^{train}$ into two classes by using the best tree branching split: $s^* = \arg\max_s IG(\mathcal{E}^{train}, s)$. Each value of the leaf node represents a class label y for the predicted *error* or *clean* classes.

In the following section, we motivate and explain the combination of error detection methods by utilizing ensemble learning algorithms.

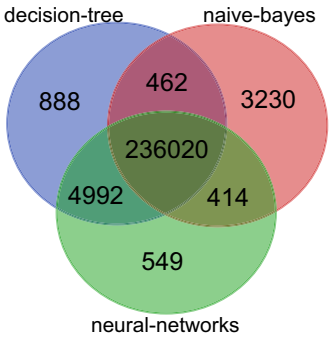

Figure 5.3: Venn diagram showing joint statistics of three classifiers: Neural Network, Decision Tree and Naive Bayes algorithms on the same dataset. Source [232].

5.2.3 Combining Error Detection Methods With Ensemble Learning

The main reason to implement ensemble learning classification models is the fact that every machine learning algorithm performs differently. We visualized the performance of the three used learners in Figure 5.3. This figure shows that although the intersection of these models is 95%, there are still errors which are identified only by one or two classifiers. Therefore, to train an error classification model, we adopt *ensemble learning*, which combines single (base) classifiers. Ensemble methods have been proven to produce better performance and robustness on classification tasks [71]. The independence between the base learners leads to the reduction of the overall classification error after the combination of the above learners [252]. Instead of trying to determine the best single base learner, ensemble methods utilize a combination of these, to achieve a better generalization ability. The advantage of this combination is motivated by the following fundamental issues of machine learning, as formulated by Dietterich T. [71]:

Statistical issue: The learning algorithms generate several different prediction models giving the same accuracy on the training data. There is a risk that the final prediction data may offer a slightly inaccurate prognosis of the future data. By averaging the prediction models, we can achieve a good approximation to the true hypothesis and, thus, the risk of selecting a wrong model can be reduced.

Computational issue: Given the fact that the majority of learning algorithms use local search, defining the final model may end up in local optima. For instance, the training of both neural networks and decision trees is NP-hard [32]. Therefore, the aggregation might achieve a better approximation to the true unknown hypothesis, by executing the local search from many different starting points. Hence, the combination of the different learning models diminishes the risk of choosing a wrong local minimum.

Representational issue: In most applications of machine learning, the optimal hypothesis cannot be represented by any of the hypotheses search space. By establishing a

weighted sum of hypotheses, it may be feasible to expand the space of hypotheses, and hence the learning algorithm is able to learn a more accurate approximation to the true unknown hypothesis function.

These fundamental issues are the most important reasons why existing learning algorithms sometimes perform poorly [71, 252]. Therefore, ensemble methods are designed to diminish these three shortcomings of learning algorithms.

According to the *no free lunch theorem* [35], there are no generally best models, and the common way to evaluate different classification methods is to empirically assess each of them [171]. In the following, we present two variants of the ensemble methods: **Stacking** and **Bagging**. While **Stacking** is arguably the more advanced approach and yields better results in our experiments, we also include the discussion on **Bagging** as one of the best and most straightforward methods according to the empirical comparison of supervised learning algorithms conducted in [42]. In general, 1) **Stacking** combines multiple (different) error detection models into one model, which is trained on the output of these models [243]. The first-layer learners are trained by using the same training data. 2) **Bagging** combines learning models of the same machine learning algorithm, while the base learners are trained on different subsamples of the training dataset [37].

5.2.4 Combining Error Detection Methods With Stacking

Our first approach for error detection strategies aggregation is *Stacking*. It combines different base error-detection classifiers into one meta-classifier [243]. All used models, **Neural Network**, **Decision Tree**, and **Naive Bayes** [29], produce various results for error classification. The numbers in each area denote the total count of correctly classified errors of the overlapping models. Therefore, the goal of using the stacking strategy is to aggregate these results and achieve an improved accuracy compared to a single learned model. Although the intersection of all three results is 95%, there are still errors which are identified only by one or two classifiers. Hence, stacking utilizes the combination to achieve a better generalization of the meta-learner function.

A general flow of stacking is demonstrated in Figure 5.4. The pseudo-code of error detection that is based on the stacking combination method is summarized in Algorithm 5.1. The input of this algorithm consists of two parameters. First parameter is the training dataset $\mathcal{E}^{train}$. The second parameter is the dataset $\mathcal{D}$, where the data cells need to be classified as *"clean"* or *"error"*. The output of the Algorithm 5.1 is a vector $\mathbf{e}^{out} \in \{0, 1\}$ that contains the predicted labels for each data value of the dataset $\mathcal{D}$.

Initially, in the Algorithm 5.1 code line 1, we create the feature vector $\mathcal{F}_{train}$, which is based on the labeled training dataset $\mathcal{E}^{train}$. We follow the ensemble setting provided by Caruana R. et al., [42] and train three classifiers based on the following algorithms: $\mathcal{C}_{dt}$ - *Decision Tree*, $\mathcal{C}_{nb}$ - *Naive Bayes* and $\mathcal{C}_{nn}$ - *Neural Network* (Algorithm 5.1 code

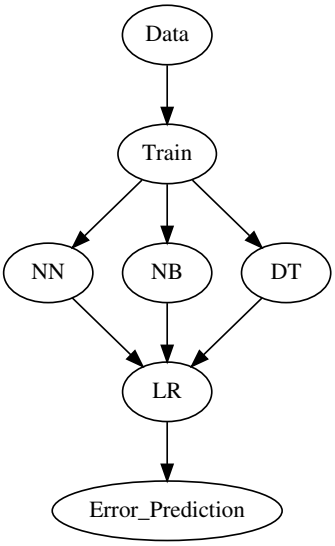

Figure 5.4: Stacking Algorithm. Node notations: NN - Neural Network, DT - Decision Tree, NB - Naive Bayes and LR - Logistic Regression. Training data is denoted by the node *Train*. Source [232].

lines 3-5). The outputs of the base-learner functions are processed by the meta-predictor - **Logistic Regression Classifier** [29], those training is shown in the code line 6. This meta-classifier, takes the output values of three base-classifiers $\mathbf{c} = (\mathbf{e}_{nb}, \mathbf{e}_{nn}, \mathbf{e}_{dt})$ and forms a linear combination of these values given the corresponding labels $\mathcal{L} \in \{0, 1\}$. The highest value of combination will be assigned to the particular class $\mathbf{e}^{out} \in \{0, 1\}$. Formally, this binary classifier can be represented as a Bernoulli distribution of the form:

$$\mathbf{e}^{out} = P(\mathcal{L}|\mathbf{c}, \mathbf{w}) = \text{Ber}(\mathcal{L}|\text{sigm}(\mathbf{w}^T\mathbf{c})), \text{ where}$$
$$\text{sigm}(\mathbf{w}^T\mathbf{c}) = \frac{1}{1 + \exp(-\mathbf{w}^T\mathbf{c})}$$

where $\mathbf{w}^T\mathbf{c}$ represents the inner scalar product between the model's weight vector $\mathbf{w}$ and the feature vector $\mathbf{c}$. Classifying data errors on $\mathcal{D}$ by using base classifiers and processing their outputs by the meta-learner are shown in the Algorithm 5.1 code lines 8-11.

5.2.5 Combining Error Detection Methods With Bagging

The next error prediction technique uses the *Bagging* [37] ensemble method. We use the *parallel ensemble methods* paradigm where the base learners are generated independently from each other. The main reason for the use of parallel ensemble methods is to expoloit the independence between the base learners. As the error can be reduced considerably by combining independent base learners [37, 252]. The essential steps for the bagging strategy are provided in Algorithm 5.2 and Figure 5.5. The input of this Algorithms are

Algorithm 5.1 Algorithm for learning the error classification model that is based on stacking ensemble learning.

function ErrorDetectionWithStacking($\mathcal{E}^{train}, \mathcal{D}$)
  $\mathcal{F}_{train} \leftarrow$ CreateFeatures($\mathcal{E}^{train}$)
  $\mathcal{C}_{nb} \leftarrow$ NaiveBayes($\mathcal{F}_{train}$) ▷ Learning the Naive Bayes base model
  $\mathcal{C}_{dt} \leftarrow$ DecisionTree($\mathcal{F}_{train}$) ▷ Learning the Decision Tree base model
  $\mathcal{C}_{nn} \leftarrow$ NeuralNetwork($\mathcal{F}_{train}$) ▷ Learning the Neural Network base model
  $\mathcal{C}_{lr} \leftarrow$ LogisticRegression($\mathcal{C}_{nb}, \mathcal{C}_{dt}, \mathcal{C}_{nn}$) ▷ Learning the Logistic Regression meta-model on the outputs of the base models
  $\mathcal{D}_f \leftarrow$ CreateFeatures($\mathcal{D}$)
  $e_{nb} \leftarrow \mathcal{C}_{nb}(\mathcal{D}_f)$ ▷ Classifying data errors with the Naive Bayes base model
  $e_{dt} \leftarrow \mathcal{C}_{dt}(\mathcal{D}_f)$ ▷ Classifying data errors with the Decision Tree base model
  $e_{nn} \leftarrow \mathcal{C}_{nn}(\mathcal{D}_f)$ ▷ Classifying data errors with the Neural Network base model
  $e^{out} \leftarrow \mathcal{C}_{lr}(e_{nb}, e_{dt}, e_{nn})$ ▷ Classifying data errors with the Logistic Regression meta-model on the outputs of the base models
  return e^{out}
end function

Algorithm 5.2 Algorithm for learning the error classification model that is based on bagging ensemble learning.

function ErrorDetectionWithBagging($\mathcal{E}^{train}, k, \mathcal{D}$)
  $DT \leftarrow []$
  $T \leftarrow []$
  $\mathcal{F}_{train} \leftarrow$ CreateFeatures($\mathcal{E}^{train}$)
  for $i \leftarrow k$ **do**
    $\mathcal{B}_i \leftarrow$ RandomSample($\mathcal{F}^{train}$) ▷ use $\mathcal{B}_i$ to learn model DT_i
    $\mathcal{C}_i^{dt} \leftarrow$ DecisionTree($\mathcal{B}_i$) ▷ Learning the Decision Tree base model
    $DT \leftarrow DT + \mathcal{C}_i^{dt}$
  end for
  $\mathcal{D}_f \leftarrow$ CreateFeatures($\mathcal{D}$)
  for $\mathcal{C}_i^{dt} \leftarrow DT$ **do**
    $e_i \leftarrow \mathcal{C}_i^{dt}(\mathcal{D}_f)$ ▷ Classifying data errors with the Decision Tree model on the dataset $\mathcal{D}$
    $T \leftarrow T + e_i$
  end for
  $e^{out} \leftarrow$ MajorityWins(T) ▷ Final classifying data errors with the MajorityWins meta-combiner
  return e_{out}
end function

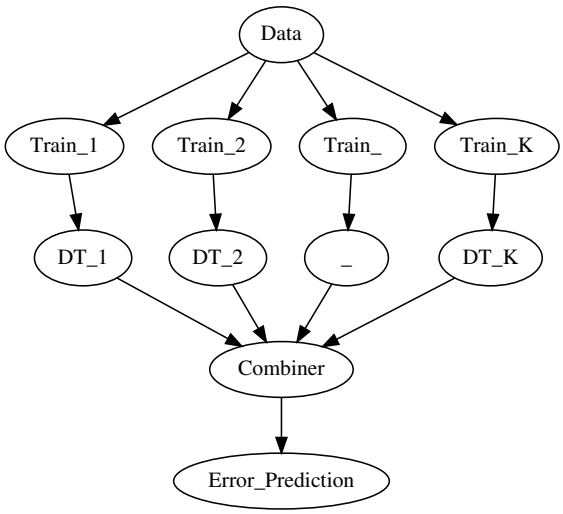

Figure 5.5: Bagging Algorithm. Node notations: DT_i - Decision Tree. Training data is denoted by the node $Train_i$. Source [232].

three parameters, first, the training dataset $\mathcal{E}^{train}$, second, the parameter k denotes the number of base-learners, which will be trained on k- different subsets, and the third is dataset $\mathcal{D}$, where the data cells need to be classified as *"clean"* or *"error"*. The output of the Algorithm 5.2 is a vector $\mathbf{e}^{out} \in \{0, 1\}$ that contains the predicted labels for each data value of the dataset $\mathcal{D}$.

Bagging algorithms take advantage of the bootstrap distribution for generating different base learners. Technically, it uses bootstrap sampling [77] to obtain the data subsets for training the base learners. Basically, given a training dataset $\mathcal{E}^{train}$ containing n training examples, a sample of m_i training examples will be produced by sampling with replacement. This leads to some original examples appearing more than once, and some examples not being present in the sample. By repeating the procedure K times, we obtain K samples of m training examples. Finally, from each sample $t_i, i \leq |K|$ a base learner can be trained by applying the base learning algorithm (Algorithm 5.2 code lines 5-9). In our approach, the bagging ensemble learning algorithm utilizes the *Decision Tree* algorithm as a base learner. A *Decision Tree* is a popular and effective classifier [171]. This algorithm is a greedy method to binary partition the feature space.

Given a training dataset $\mathcal{E}^{train}$, the bagged classifier $\mathcal{C}_{dt}$ collects the output of each previously trained classifier DT_i and returns a final prediction of each cell in $\mathcal{D}$ being erroneous or not.

To predict an instance, Bagging feeds this instance to its base classifiers, collects all of their outputs (Algorithm 5.2 code lines 11-14), then applies Majority Voting to the labels and takes the winner label as the output class label of the ensemble (Algorithm 5.2 code line 15). Combining decision trees is a competitive baseline compared to other classifiers [42].

Therefore we used this ensemble method to compare against other ensemble learning methods.

5.2.6 Eliminating Redundant Error Detection Strategies

As motivated at the beginning of this chapter, we might run idempotent error detection strategies, meaning they might follow the same strategy to identify errors. For instance, sometimes rule- and pattern-based approaches use similar histogram-based heuristics to find errors [127, 190]. These approaches might detect very similar set of errors. In other words, these systems are idempotent. As a result, this will skew the error classification learning approach in favour of these two algorithms, which means that we would need to identify representative data cleaning systems of such overlapping groups. To address this problem, we extend the classification strategy with an initial phase to eliminate redundant error detection systems by selecting the representative solutions from the $\mathcal{T}$ approaches. In the remainder of this section, we refer to this representative subset as a *Best-K data cleaning solutions*, where $K \leq |\mathcal{T}|$.

Our method requires two execution stages:

1) Best-K *Strategies Selection*, where we determine the subset $\mathcal{T}^*$ of all $\mathcal{T}$ strategies.

2) Best-K *Result Aggregation*, where we apply the Classification Strategy (described in Sections 5.2.4 and 5.2.5) to the subset $\mathcal{T}^*$ to classify the input dataset cells into *error* or *not error* classes.

The general method for the proposed strategy of eliminating idempotent error detection algorithms is provided in Algorithm 5.3 code lines 19-28. The input of this algorithms are three parameters, first, the training dataset $\mathcal{E}^{train}$, second, $\mathcal{L}^{train}$ labels of the training data, and third, the parameter k that denotes the size of subset $\mathcal{T}^*$ of all $\mathcal{T}$ strategies. The output of the Algorithm 5.3 is a set B of non-overlapping systems. In the following, we describe the 5 steps of our algorithm to select and aggregate $\mathcal{T}^*$ systems:

Step 1: Execution of Error Detection Approaches. Initially, we run available data quality approaches independently of each other on the dataset $\mathcal{D}$. The result contains all errors detected by each system, represented as a matrix $\mathcal{E}$, as described in Section 5.2.1. Furthermore, by using the training subset $\mathcal{E}^{train}$ where the labels $\mathcal{L}^{train}$ are available, we can calculate precision - the percentage of correctly identified errors for each system output.

Step 2: The Truth Matrix Creation. Given a sample $\mathcal{E}^{train}$ and the respective labels $\mathcal{L}^{train}$, which reflect whether a data cell is an error or not, we create a truth matrix $\mathcal{C}$ by applying the **XNOR**-operator [19] on each column in $\mathcal{E}^{train}$ and label vector $\mathcal{L}^{train}$. The

Algorithm 5.3 Algorithm for eliminating redundant error detection strategies.

function createTruthMatrix($\mathcal{E}, \mathcal{L}$) ▷ Encoding labels into the error detection results
    $T \leftarrow \text{XNOR}(\mathcal{E}, \mathcal{L})$
    return T
end function
function clusterSystems($\mathcal{T}, k$)
    $C \leftarrow$ kMeansClustering($\mathcal{T}, k$) ▷ Applying the k-Means clustering algorithm to cluster similar data cleaning systems.
    return C
end function
function bestSystemInCluster(C) ▷ C is a set of error detection systems in one cluster
    $P \leftarrow []$
    for $i \leftarrow C$ **do**
        $P_i \leftarrow$ getPrecisionOfSystem(i) ▷ Determining the precision score of each system in cluster
        $P \leftarrow P + P_i$
    end for
    $P_{max} \leftarrow \text{MAX}(P)$
    $T \leftarrow$ getSystemWithMaxPrecision(P_{max}, C) ▷ Determining the system with the highest precision.
    return T
end function
function BestKSystemsSelection($\mathcal{E}^{train}, \mathcal{L}^{train}, k$)
    $B \leftarrow []$
    $T \leftarrow$ createTruthMatrix($\mathcal{E}^{train}, \mathcal{L}^{train}$)
    $C \leftarrow$ clusterSystems(T, k)
    for $c \leftarrow C$ **do**
        $B_c \leftarrow$ bestSystemInCluster(c)
        $B \leftarrow B + B_c$
    end for
    return B
end function

values of this matrix are generated as follows:

$$c_{t,i} = e_{t,i}^{(j)} \textbf{ XNOR } l_{t,i}^{(j)}, \text{ that is } c_{t,i} = \begin{cases} 1 & \text{if } e_{t,i}^{(j)} = l_{t,i}^{(j)} \\ 0 & \text{otherwise} \end{cases}$$

The semantics of the **XNOR**-operator is the following:

$$\mathcal{E}^{train} \textbf{ XNOR } \mathcal{L}^{train} \equiv \neg(\mathcal{E}^{train} \textbf{ XOR } \mathcal{L}^{train})$$

The operation **XNOR** is also called *identity*, which means that the result is *true* if the values of the operands $\mathcal{E}^{train}$ and $\mathcal{L}^{train}$ are the same [19]. Hence, by applying the **XNOR**-operator, we created a truth matrix $\mathcal{C}$ that reflects whether the particular error detection system delivered a correct result. The above *Truth Matrix Creation* step is provided as pseudo-code in Algorithm 5.3 code lines 1-4.

Step 3: Clustering the Error Detection Results. Provided with the truth matrix $\mathcal{C}$ of $n = |\mathcal{T}|$ error detection results on $\mathcal{D}$, we partition similar system results into k groups ($k \leq n$). We apply k-means clustering [108], where each partition represents a cluster of identical systems. We consider two error detection strategies t_i and t_j, $(i \neq j)$ as similar if they produce similar error detection results on the same dataset. This similarity can be determined by using proper metrics for measuring the overlap between two finite sets $e^{(i)}$ and $e^{(j)}$. Suitable similarity metrics are, for example, the Szymkiewicz–Simpson coefficient, the Euclidean distance, or the Jaccard similarity coefficient [231, 78]. The result of the clustering is the distribution of the error detection strategies into k clusters, $C_1, C_2, \ldots, C_k$, that is, $C_i \subset \mathcal{T}$ and $C_i \cap C_j \in \varnothing (1 \leq i, j \leq k)$. The above *clustering* step is provided as pseudo-code in Algorithm 5.3 code lines 5-8.

Step 4: Precision-Based Best-K Strategies Selection. The output of the clustering step is a set of system groups. Further, for each cluster, we select the error detection strategy b_j based on the system results with the highest precision score. This step is provided as pseudo-code in Algorithm 5.3 code lines 9-18.

Step 5: Best-K Strategies Combination. After determining a subset of the k most effective approaches in our system as a feature vector $\mathbf{b} = (b_1, b_2, \ldots, b_k)$, we apply the combination (based on classification) strategy for error detection, which has been presented in Sections 5.2.4 and 5.2.5.

In Chapter 4, we conducted a systematic study for the intrinsic connection between metadata and data quality issues and developed effective methods for error detection based on extracted metadata. In the following, we utilize these findings and augment the feature vector with metadata to perform the error classification task.

5.3 Metadata-Augmented Error Classification

Abedjan Z. et al. [2] conducted an empirical evaluation of data cleaning systems on multiple types of real-world data. This study yielded insights into the differences between datasets in regard to the distribution of error types. To support the dataset-specific aggregation of error detection systems, we augment the learning approaches with profiling information. In this section, we provide an approach of designing features for metadata-augmented error classification by modelling the instance- and schema-based metadata features.

Metadata has been extensively studied in Chapter 4, where we already provided a systematic analysis of metadata functions and its connection to data quality management. We established a mapping between data quality problems and metadata (see Section 4.1). In this way, we demonstrated that exploiting metadata will address data quality issues [195, 233]. Our goal is to include the characteristics of the dataset into the error classification method. To design features for metadata-augmented error classification, we suggest the qualitative categorization of metadata. This taxonomy should reflect the dataset characteristic and be included in the classification methods.

We propose five broad feature groups which are based on metadata. To support error detection, we propose to encode instance- and schema-based metadata for each attribute. For this purpose, we introduce an additional feature vector $\mathcal{M}$ that contains all metadata information for each cell in $\mathcal{D}$. Each attribute value $v_{t,i}$ of t-th tuple and i-th attribute, $i \in attr(\mathcal{S})$, is associated with a metadata tuple $(m_{t,i}^{(1)}, m_{t,i}^{(2)}, ...m_{t,i}^{(|\mathcal{M}|)})$, where $|\mathcal{M}|$ is the number of generated metadata features.

In the following, we outline each of the metadata-based features:

Attribute Completeness Features. As provided in Chapter 4, the *Completeness* data quality dimension is reflected by the following metadata: *number of rows, null values, size (numeric), outliers (for default values)*, and *attribute type/class/domain defaults*. For example, we use the information about *null* values, which expresses whether the value in a particular data cell is absent in the input dataset. Below, we show the metadata-features modelling: the values of $m^{(completeness)}$ are assigned as follows:

$$m_{t,i}^{(completeness)} = \begin{cases} 1 & \text{if the value in the } (t,i)\text{-th cell is absent;} \\ 0 & \text{otherwise;} \end{cases}$$

Alternatively, to reflect the presence of default values in the attribute, we model this feature as follows:

$$m_{t,i}^{(default)} = \begin{cases} 1 & \text{if the value in the } (t,i)\text{-th cell belongs to the default values of the data type;} \\ 0 & \text{otherwise;} \end{cases}$$

Attribute Type Affiliation Features. The generic data types determine whether the data type on a particular column belongs to one of the following data types and structures: *boolean, date&time, string, integer,* or *decimal*. As we already have shown in Chapter 4, knowledge about the column type is crucial in the *pattern violation detection* to identify non-permissible values for this attribute [229].

Attribute Domain Features. As we already discussed in Chapter 4, the *domain* and *semantic role* metadata is crucial for the identification of data quality issues, such as, *incorrect data, misspellings, ambiguous data, misfielded values*, and *domain violation*. We model the attribute domain information as a binary variable indicating whether the attribute value belongs to one of the following domains: *IP-Address, URL, HttpCode, social security number, phone number, e-mail, credit card number, gender, zip, US state*, or *geographical data*. In this way, we target *outlier detection* or *pattern violation detection* algorithms [229, 190]. Concretely, we model these features as follows:

$$m_{t,i}^{(dom:e-mail)} = \begin{cases} 1 & \text{if the value in the } (t,i)\text{-th cell matches the format of the } \textit{e-mail}; \\ 0 & \text{otherwise;} \end{cases}$$

Attribute Value Frequency Features. This class of metadata represents the distribution of either the attribute values or the alternative representation of attribute values such as patterns, n-grams, or embeddings of each attribute. As detailed in the previous Chapter 4, we follow the same hypothesis as *extreme values analysis* [116] and *histogram-based* techniques in outlier detection [190, 8] and use the *histogram* metadatum to identify the following data quality issues: *incorrect data, misspellings, ambiguous data, misfielded values*, and *use of special characters*. In particular, the values of $m^{(top-values)}$ are assigned according to the following rule:

$$m_{t,i}^{(top-values)} = \begin{cases} 1 & \text{if the value in the } (t,i)\text{-th cell belongs to the TOP10 attribute values;} \\ 0 & \text{otherwise;} \end{cases}$$

Multi-attribute Dependency Features. According to our research, conducted in Chapter 4, *dataset dependencies* metadata is used to identify the violation of all data quality dimensions. We encode the relationships between attributes as an additional set of features. For instance, in our approach to supervised error detection, we use *Functional Dependencies* of the form $\phi : A \rightarrow B$, which expresses that the attribute A functionally determines another attribute B. Since the error detection systems utilize FDs as rules for duplicate detection and rule violation detection [62], the FDs mostly include attributes, which might be relevant in the error detection process. To include more signals about the relevant attributes A and B, we propose the following straightforward modelling: for each

available FD, one binary feature vector is generated. Each vector expresses whether the attribute value $v_{t,i}$ appears in one of the involved columns of the dependency or not. For a functional dependency ϕ_k, the feature vector $m^{(\phi_k)}$ is created as follows:

$$m_{t,i}^{(\phi_k)} = \begin{cases} 1 & \text{if the } i\text{-attribute is in } \phi_k; \\ 0 & \text{otherwise;} \end{cases}$$

By generating metadata-based features, the initial feature vector that reflects the results of the error detection systems $\mathbf{e}^{train} = (e_{t,i}^{(1)}, e_{t,i}^{(2)}, ...e_{t,i}^{(|\mathcal{T}|)})$ is extended by the metadata vector $\mathbf{m} = (m_{t,i}^{(1)}, m_{t,i}^{(2)}, ...m_{t,i}^{(|\mathcal{M}|)})$. Hence, given the extended feature space $\mathcal{F} = [E_{n\times|\mathcal{T}|}M_{n\times|\mathcal{M}|}]$, the error classification method is considered as the learning of the mapping function $y = f(\mathcal{F}^{train}) = f(e_{t,i}^{(1)}, e_{t,i}^{(2)}, ...e_{t,i}^{(|\mathcal{T}|)}, m_{t,i}^{(1)}, m_{t,i}^{(2)}, ...m_{t,i}^{(|\mathcal{M}|)})$.

Table 5.1 summarizes the complete list of the above described metadata-based features. To aggregate the newly selected set of error detection systems, both algorithms, either *Stacking* or *Bagging*, can be applied on the enhanced feature space in the same way, as described in Sections 5.2.4, 5.2.5, and 5.2.6.

5.4 Experiments

In this section, we provide experimental results of supervised and metadata-driven error detection. We first describe the experimental setup in Section 5.4.1. The performance of error detection systems is evaluated in Section 5.4.2. Next, we explain the classification algorithms setup and the optimization steps in Section 5.4.3. All experiments are compared to multiple baselines, as described in Section 5.4.4. Finally, we provide an evaluation of our aggregation approach in Section 5.4.5, and the method of combining the most effective error detection systems is evaluated in Section 5.4.6.

5.4.1 Experimental Setup

Datasets. The experiments are performed on four different datasets, as summarized in Table 5.2. The HOSP dataset is available on the website of the US Department of Health and Human Services[1]. This dataset consists of 10k records with 18 attributes, which are mainly addresses, ZIP codes, state codes, and hospital names. This dataset is a ground truth. To produce a dirty version of the HOSP dataset, we applied the BART system [15]. We configured BART to insert functional dependency violations by changing values in data fields. The inserted error percentage is 9.2%.

[1] http://www.medicare.gov/hospitalcompare/Data/Data-Download.html

Feature Category	Dataset Metadata	Feature Description
Attribute Completeness	Number of rows Null values Size (numeric) Outliers (for default values) Attribute type/class/domain defaults	A numeric value reflecting the number of rows or the size of the numeric value. A binary value expressing whether the value $v_{t,i}$ is absent in the input dataset $\mathcal{D}$. A binary value expressing whether the value $v_{t,i}$ is a default value for a given attribute type/class/domain.
Attribute Type Affiliation	Data type/class affiliation, such as boolean, string, integer, decimal, date/time	A binary value expressing whether the value $v_{t,i}$ is of the mentioned attribute type/class/domain.
Attribute Domain Affiliation	Domain/semantic role format, such as IP-Address, URL, HttpCode, Social Security Number, Phone Number, E-Mail, Credit Card Number, Gender, Zip, US States, Geographical data.	Binary variables indicating whether the value $v_{t,i}$ belongs to one of the mentioned domains.
Attribute Value Frequencies	Histogram of values, Histogram of numerical representations, Histogram of patterns, Histogram of n-grams, Frequency distributions	A binary variable indicating whether the value $v_{t,i}$ belongs to the $\textsc{Top}10$ values of the column. A binary variable indicating whether the value $v_{t,i}$ belongs to the $\textsc{tail}$ of the distribution.
Multi-attribute Dependencies	FDs, CFDs, Relaxed FDs [43]	For each available integrity constraints ϕ one binary feature vector is created to encode whether the value is in ϕ.

Table 5.1: Metadata-based features used for enhancing error detection strategies.

	ADDRESS	HOSP	SALARIES	FLIGHTS
# columns	12	18	13	9
# rows	94k	10k	75k	74k
ground truth	94k	10k	75k	74k
real errors	36.9 %	-	-	61.85 %
generated errors	-	9.2 %	2.33 %	-
Features number				
SYSTEM	5	5	5	5
META	13	16	12	13

Table 5.2: Experimental datasets.

Another real-world dataset - SALARIES - contains the names, job titles, and salaries of San Francisco city employees on an annual basis from 2011 to 2014[2]. This dataset comprises 75k tuples. The attributes of this dataset are mainly numerical values. To produce the dirty version of the SALARIES dataset, we used the BART system and introduced 2.33% errors. We configured BART to produce numerical outliers spread on several numerical attributes, such as *base pay, overtime pay, other pay, benefits, total pay, total pay benefits*.

Please note that another two datasets, namely ADDRESSES and FLIGHTS, which are used in the current experiments have already been described in Chapter 3. To guarantee reproducibility, our system includes openly available error detection frameworks and operates on freely available datasets[3], and our code-base is provided online[4].

Evaluation Metrics. To evaluate the baselines and aggregation methods, we use *Precision (P)*, *Recall (R)* and their harmonic mean - *F-measure (F_1)*. To compute these scores, we use the ground truth of each dataset to determine tp - the correctly selected values as errors; fp - values that are falsely determined as errors and fn - erroneous values that are not marked as such. The above evaluation scores are computed as follows:

$$P = \frac{tp}{tp + fp} \; ; \; R = \frac{tp}{tp + fn} \; ; \; F_1 = \frac{2PR}{P + R}$$

Implementation Details All experiments were performed on a single machine with 2.3 GHz Intel Core i7 processor and 16 GB RAM. The implementation language is Scala.

5.4.2 Performance of Error Detection Systems

We begin with the performance analysis of the error detection suite. The system contains four major components for detecting outliers, duplicates, rule violations, and pattern

[2]https://www.kaggle.com/kaggle/sf-salaries
[3]`http://bit.ly/datasets-for-aggregation`
[4]`http://bit.ly/systems-aggregation`

Data Error Detection Strategy	Constituent System	Note
Pattern violations detection	WRANGLER	
Rule violations detection	NADEEF (FD)	
Outliers detection	dBOOST(Hist)	Categorical data
	dBOOST(Gauss)	Numerical data
Duplicate Conflicts	NADEEF (D)	

Table 5.3: Representative for each error detection strategy. Source [232].

	ADDRESS			HOSP			SALARIES			FLIGHTS		
Constituent System	P	R	F-1	P	R	F-1	P	R	F-1	P	R	F-1
WRANGLER	0.4145	0.1398	0.2091	0.9448	0.2016	0.3322	0.0024	0.0199	0.0043	0.8894	0.3888	0.5411
NADEEF(D)	0.3154	0.1137	0.1672	0.0904	0.8748	0.1638	0.0989	8.0E-4	0.0016	0.6189	0.9916	**0.7621**
dBOOST(Hist)	0.2843	0.0087	0.017	0.2954	0.2985	**0.2969**	0.0326	0.1337	0.0524	0.4895	0.066	0.1163
dBOOST(Gauss)	0.3188	0.1778	0.2282	0.2169	0.0263	0.0469	0.1513	0.1073	**0.1256**	0.8141	0.0131	0.0258
NADEEF(FD)	0.5235	0.1776	**0.2652**	0.2622	0.9845	0.4141	0.1313	0.0036	0.007	0.6433	0.0719	0.1293

Table 5.4: Performance of each constituent system. The evaluation was performed on the complete dataset. The best results are marked as **bold**. Source [232].

violations. For each error type, we identified one representative error detection system, as shown in Table 5.3. The performance of each data cleaning system is provided in Table 5.4. Note that due to differences between datasets, the results of each constituent data cleaning system are different.

The *pattern violation* discovery is represented by the Wrangler system [127]. This system covers a wide range of transformations and pattern violations in data. Provided with pattern rules, Wrangler captures misfielded, mismatched, invalid or empty values. This component achieves either the highest or the second-highest result on all three datasets: ADDRESS, HOSP and FLIGHTS. Pattern violation includes missing values and misspelled values. Since the major error types in the SALARIES dataset is mostly numerical outliers, Wrangler was not able to capture any such values, except for negative numbers. The *outlier detection* component includes the dBOOST [190] framework. For our experiments, we use two algorithms of the outlier detection method: *Gaussian-based* - for detecting outliers in numerical attributes and *Histogram-based* - to identify outliers in categorical attributes. The SALARIES dataset contains mostly numerical outliers in the six payment attributes. Therefore, dBOOST is the only system that is able to capture these errors in SALARIES. Furthermore, dBOOST also identifies missing values as outliers. To identify conflicting values in *duplicates*, we use the NADEEF (D) [62] system with its ability to integrate data cleaning rules, which are based on matching and functional dependencies. As the FLIGHTS dataset largely contains duplicates, NADEEF(D) demonstrated to be the most effective system to capture these errors. The SALARIES dataset contains almost unique data values for each person. Hence, the duplicate-based approach on this dataset is the least effective. The *rule violation* discovery component is also depicted by the Nadeef(FD) [62] system, which has comparably good performance in terms of

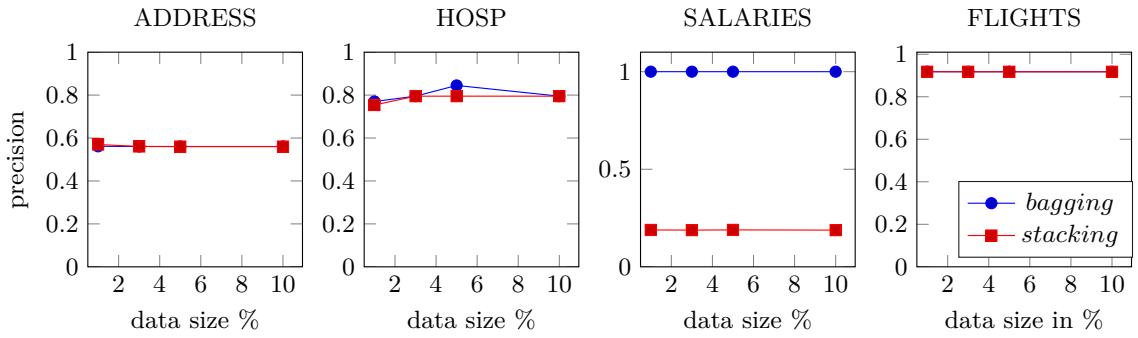

Figure 5.6: Precision scores of the classification model performance for different sizes of training data. Source [232].

F_1-measure measure on the HOSP and ADDRESS datasets. However, on SALARIES, NADEEF(FD) provides low F_1-measure and low recall compared to other error detection systems, as SALARIES-specific outliers cannot be captured by the data quality rules. For all datasets, the set of rules is provided in the form of functional dependencies. To specify such dependencies, we used the METANOME [182] framework.

Additionally, running each system in parallel generates a large number of detection results - including many false positives. To overcome this problem, we suggest combining error detection strategies that would cover all types of data errors.

5.4.3 Classification Algorithms Setup

Applying machine learning methods requires optimization such as labeling of the training data, model selection, and using effective features. The most common problem in machine learning-assisted applications is *overfitting*, meaning insufficient generalization of trained classifiers, which will thus perform poor on unseen data. A group of learning models will prevent overfitting, hence perform better than the best individual model [37], which shaped our decision to apply ensemble learning.

An important initial question for nearly every supervised machine learning method is the amount of labelled data for model learning. As this is an expensive process, it requires either a human annotator or a clean version of data to create labels. [56, 198]. We have estimated empirically that 1% of the dataset is sufficient to successfully train data error classifiers. To estimate the sufficient amount of training data, we conducted several experiments for training classifiers on training data with different sizes ranging from 10% to 1%. Practically, for each dataset, we randomly sampled 1% of the results of all error detection systems and used this subset $\mathcal{E}^{train}$ to train our error classification models in all experiments described below. Figure 5.6 shows that for all datasets, a small training data size, such as 1% of the dataset, is sufficient to train the error classifier. The labeling of

the training and testing subsets is performed automatically because we are provided the ground truth for the complete dataset.

In the training phase of bagging, we train $2k$ *Decision Tree* models by using randomly sampled training subsets with replacement $\mathcal{B}_1, \mathcal{B}_2 \dots \mathcal{B}_{2k}$ from the initial $\mathcal{E}^{train}$. Please note that in our experiments $k = 6$ is an empirically chosen number. The complementary subset to each sample $\mathcal{E}^{train} \setminus \mathcal{B}_i$ is used to test the trained model. After training and testing all $2k$ models, the system selects the best k models. Initially, we trained 12 *Decision Tree* models by using 12 randomly sampled training subsets with replacement from the initial $\mathcal{E}^{train}$. The complementary subset to each training set is applied to test the trained model. After training and testing all models, the system selects 6 models with the top F_1 scores. The *Decision Tree* model was trained with the following hyperparameters: the split criteria is based on the calculation of the information gain according to the "Gini"-impurity criteria. The default tree width is equal to the number of combining systems.

In the training phase of stacking, we generate a new dataset from the first-level classifiers. The risk of overfitting becomes higher if the exact data that is used to train the base-learner is also used to generate the new dataset for training the meta-learner [252]. In detail, we use *k-fold* cross-validation. The original training dataset $\mathcal{E}^{train}$ is randomly split into k parts $\mathcal{B}_1, \mathcal{B}_2 \dots \mathcal{B}_k$. We specify a $\mathcal{B}_i$ test set and $\mathcal{B}_{-i} = \mathcal{E}^{train} \setminus \mathcal{B}_i$ training sets for the i-th fold. Given T learning algorithms, a base-learner $h(-i)$ is obtained by invoking the learning algorithm on $\mathcal{B}_{-i}$. For each i-th fold test set $\mathcal{B}_i$, we denote b as the output of the learner $h(-i)$. At the end of the cross-validation procedure, the new dataset is produced from the T individual learning algorithms as $B' = \{(b_{i,1} \dots b_{i,n}, l_i)\}_{i=1}^{n}$, on which the meta learning algorithm will be applied, and the resulting learner h' is a function of $(b_1, ..., b_T)$ for the label y. We provide the setup of each particular algorithm: *Decision Tree*, *Neural Network*, *Naive Bayes* and *Logistic Regression*. For the *Decision Tree* model selection, we randomly subsampled five training sets from the training data, we trained five models and selected the most effective one as a base-model. For the *Neural Network*, we used the multilayer perceptron classifier based on the feedforward *Neural Network*. For instance, we used a single-hidden-layer perceptron network with the sigmoid (logistic) activation function. The input layer size is set to the sum the number of aggregated systems and the number of the available metadata features. The output layer size is two, denoting "error" and "clean" classes. The initial weights are randomly generated by using the seed value. For the *Naive Bayes* classifier, we used the Bernoulli model, which reflects the presence or absence of the error identification by the existing algorithms. The final meta-model selection for the *Logistic Regression* classifier is performed by greedy search on the 32 threshold parameters ranging from 0.01 to 0.6: the model with the largest F_1 score is selected. For instance, the system selects the best threshold value for the *Logistic Regression* function by assessing the

Features	ADDRESS	HOSP	SALARIES	FLIGHTS
missingValue	0.1329	0.0015	0.0049	0.0672
isFrequentValue	0.0027	0.0168	0.0052	0.0333
isString	0.021	0.0131	0.011	0.2617
isDateTime	0.0439	-	0.0019	0.2617
isZipCode	8.0E-4	0.0043	-	-
isBoolean	-	0.006	-	-
isAddress	0.0015	-	-	-
isDecimal	-	-	0.0178	-
isInteger	-	-	0.0019	-
isPhoneNumber	-	0.0208	-	-
isSSN	0.0033	-	-	-

Table 5.5: The features importance. Information gain measures for adding various metadata features on all datasets. The dashes '-' denote that the particular feature is not available in the dataset. Source [232].

model performance for each possible value in a greed. The above-described classification algorithm settings are the same for all datasets used in the experiments.

To support our claim that metadata features encode important signals in error detection (see Chapter 4), we computed the information gain for every applicable metadata feature for all datasets. Information gain measures in bits how much information each particular metadata column provides about the *error class*. Basically, this measure interprets the importance of the features. Unrelated features should get *zero information gain*. Table 5.5 shows that metadata features provide relevant information about errors in data.

5.4.4 Baselines

To evaluate our proposed approach to combine data cleaning systems, we compare our classification strategy to several baselines from the related work: *UnionAll* [2], *Min-k* [2], and *Majority wins* [252]. *UnionAll* takes the union of the errors yielded by all error detection systems. *Min-K* is a voting strategy, which accounts an error when at least K systems detected a cell as an error. Since *UnionAll* and *Min-1* are the same baselines, we restrict K as follows: $2 \leq K \leq |\mathcal{T}|$, where $|\mathcal{T}|$ is the number of the constituent systems. *Majority Wins* combines error detection systems by deciding what majority of the systems computed result for a cell $v_{t,i}$. The value of the system combination on (t, i)-th cell by using this strategy is estimated as follows:

$$m_{t,i} = \begin{cases} 1 & \text{if } \sum_{j=1}^{|\mathcal{T}|} e_{t,i}^{(j)} \geq \frac{|\mathcal{T}|}{2} \\ 0 & \text{otherwise} \end{cases}$$

To evaluate our Best-K system combination approach, we compare it against the *Precision Based Ordering (PBO)* method [2], which is an iterative approach to estimate the

	ADDRESS			HOSP			SALARIES			FLIGHTS		
Baselines	P	R	F-1	P	R	F-1	P	R	F-1	P	R	F-1
Majority Wins	0.5878	0.0157	0.0306	0.6554	0.3009	**0.4124**	0.8244	0.0167	0.0327	0.8166	0.0881	0.159
UnionAll	0.3698	0.4739	**0.4154**	0.0983	0.9994	0.179	0.0111	0.1399	0.0206	0.6187	0.9949	**0.763**
Min-2	0.3888	0.1248	0.1889	0.2271	0.8663	0.3599	0.1577	0.108	**0.1282**	0.763	0.4446	0.5618
Min-3	0.5981	0.0169	0.0328	0.6617	0.2866	0.4	0.7674	0.0148	0.0291	0.823	0.0913	0.1644
Min-4	0.9773	0.0011	0.0022	0.9279	0.1811	0.303	0.0	0.0	0.0	0.9759	0.0077	0.0154
Min-5	0.0	0.0	0.0	0.6458	0.0198	0.0385	0.0	0.0	0.0	0.0	0.0	0.0
Aggregation Strategy												
BAGGING	0.5599	0.1295	0.2103	0.9462	0.2023	0.3334	1.0	0.0161	0.0317	0.6434	0.9951	0.7815
STACKING	0.3691	1.0	0.5392	0.2655	0.9908	0.4188	0.1551	0.1081	0.1274	0.6439	0.99	0.7802
BAGGING+META	0.5615	0.886	0.6874	0.7571	0.3886	0.5136	1.0	0.0161	0.0317	0.9176	0.9769	**0.9463**
STACKING + META	0.5602	0.9127	**0.6943**	0.4637	0.6846	**0.5529**	0.1889	0.1114	**0.1401**	0.917	0.977	**0.9463**

Table 5.6: Performance of baselines compared to the results of error detection algorithms aggregation strategies. Best results are provided in **bold**. Source [232].

performance of error detection systems by assessing their performance on samples [2]. To execute this baseline, we generated a sample by randomly selecting 1% of systems output. The algorithm greedily picks the system with the highest precision after each iteration by ignoring other systems whose precision is smaller than a provided threshold value.

5.4.5 Systems Aggregation Results

In the first series of experiments, we consider the complete set of error detection systems. The corresponding baselines for this aggregation strategy are *UnionAll*, *Min-K*, and *Majority Wins*, as described in Section 5.4.4.

Bagging. By comparing bagging to the baselines, we recognize an improvement in F_1 only for the FLIGHTS dataset. We explain this improvement through the improved recall. While analyzing the SALARIES dataset, we see that the *Decision Tree* classifiers improve the precision, by decreasing false positives. The reason can be found in the original results of the error detection systems, which were not able to capture outliers and therefore these data points cannot act as positive examples during the training classifiers. Table 5.6 shows the bagging results in row $BAGGING$.

Stacking. By applying this ensemble method, we observe an improvement of error classification over the *bagging* method on SALARIES, HOSP and ADDRESS datasets. Generally, the *stacking* method improves the recall measures for all datasets by increasing error coverage, which is expressed by the true positives score. Table 5.6 shows the results of the stacking approach in row $STACKING$.

Bagging with metadata-based features. Using metadata enhanced features results in a considerable improvement of the error classification. One exception is the SALARIES

dataset. To understand this behavior, we analyzed the single column metadata on the *payment* attributes, which comprises about 6 attributes: *base pay, overtime pay, other pay, benefits, total pay, total pay benefits*. We observed that the percentage of distinct values metadatum for the *payment* attributes range from 44% to 100% per attribute. The percentage of nulls for the same attributes is almost zero. This denotes that the metadata feature matrix is very sparse, and there is very little correlation between core and metadata features. In contrast to the SALARIES dataset, executing bagging showed an improved recall on the ADDRESS and HOSP datasets. The precision increases in all three datasets: ADDRESS, HOSP and FLIGHTS, which influenced the overall improvement on the F_1 score. The dataset with the most missing values is FLIGHTS. Adding the metadata feature vector increases the F_1 score of the bagging strategy by about 18%, which is achieved by reducing false positives. Table 5.6 shows the result of extending the feature vector by metadata in row $BAGGING + META$.

Stacking with metadata-based features. As demonstrated in Table 5.6, row $STACKING + META$, adding the metadata-based feature vector achieves the most significant improvement in the F_1 measures, compared to all previous approaches and each individual system result. Importantly, the *stacking* approach on the SALARIES dataset achieves 14% F_1 score, which is the best result among all single systems, baselines and the bagging method. The reason for this increase is that the meta-model *Logistic Regression* balances between the precision score 98% from two models, *Decision Tree* and *Neural Network*, and the recall score $9,3\%$ from the third model *Naive Bayes*. In conclusion, using the *stacking* ensemble learning algorithm on the metadata-based features matrix achieves better results compared to the *bagging* approach with the same features. Basically, ensemble learning approaches perform clearly better than the baselines in Table 5.6 and every single system in Table 5.4. In particular, *Stacking with Metadata* delivers the best result on all datasets compared to the best scores from the baselines. The F_1-score increased as follows: 28% on ADDRESS, 14% on HOSP, 1% on SALARIES and 19% on FLIGHTS. Comparison with the single approaches provided in Table 5.4 showed an increase in error detecting, such as 43% on ADDRESS, 14% on HOSP, 2% on SALARIES and 18% on FLIGHTS. Although the F_1 scores might still be low, this strongly depends on the performance of the error detection constituent systems. Our approach is still able to provide significantly higher quality in the system aggregation process.

5.4.6 Aggregating the Most Effective Error Detection Systems

Influenced by the fact that many data cleaning systems subsume the error detection results of others (see Figure 5.1) and that they also produce false positives, we apply our *Best-K* strategy to select the most effective combination of the error detection systems. In the

K	ADDRESS	HOSP	SALARIES	FLIGHTS
2	dBOOST(Gauss) NADEEF(FD)	WRANGLER NADEEF(D)	dBOOST(Gauss) dBOOST(Hist)	WRANGLER dBOOST(Gauss)
3	WRANGLER BOOST(Gauss) NADEEF(FD)	WRANGLER NADEEF(FD) NADEEF(D)	dBOOST (Gauss) dBOOST(Hist) NADEEF(FD)	WRANGLER dBOOST(Gauss) NADEEF(FD)
4	WRANGLER dBOOST(Gauss) NADEEF(FD) NADEEF (D)	WRANGLER dBOOST(Hist) NADEEF(FD) NADEEF(D)	WRANGLER dBOOST(Gauss) dBOOST(Hist) NADEEF(FD)	WRANGLER dBOOST(Gauss) NADEEF(FD) NADEEF(D)

Table 5.7: Error detection systems selection on all datasets. The parameter K denotes the cluster number, which is the number of selected error detection systems. Source [232].

		ADDRESS			HOSP			SALARIES			FLIGHTS		
		P	R	F-1	P	R	F-1	P	R	F-1	P	R	F-1
	Baseline												
	PBO δ=0.1	0.3626	0.4582	**0.4049**	0.2548	0.9866	**0.405**	0.1479	0.1093	**0.1257**	0.6124	0.9945	**0.7581**
	PBO δ=0.3	0.463	0.2657	0.3376	0.963	0.1745	0.2955	0.0	0.0	0.0	0.6124	0.9945	**0.7581**
	PBO δ=0.5	0.5177	0.1696	0.2555	0.963	0.1745	0.2955	0.0	0.0	0.0	0.6125	0.9942	**0.7581**
K	**Aggregation Strategy**												
2	BAGGING	0.523	0.1773	0.2648	0.9462	0.2023	0.3334	0.0	0.0	0.0	0.6186	1.0	0.7643
	BAGGING + META	0.5615	0.8857	0.6873	0.7938	0.2693	0.4022	0.0	0.0	0.0	0.9157	0.977	**0.9454**
	STACKING	0.3691	1.0	0.5392	0.9436	0.1982	0.3276	0.1658	0.1076	0.1305	0.6191	1.0	0.7647
	STACKING + META	0.5615	0.8857	0.6873	0.7944	0.2702	0.4033	0.1894	0.1071	0.1368	0.9157	0.977	**0.9454**
3	BAGGING	0.5479	0.1403	0.2234	0.9462	0.2023	0.3334	0.0	0.0	0.0	0.6186	1.0	0.7643
	BAGGING + META	0.5606	0.8794	0.6847	1.0	0.2017	0.3357	0.0	0.0	0.0	0.9157	0.977	**0.9454**
	STACKING	0.3695	1.0	0.5396	0.2623	0.9932	0.415	0.1518	0.1103	0.1278	0.6192	1.0	0.7648
	STACKING + META	0.5602	0.9103	0.6936	0.8743	0.2695	0.412	0.1875	0.1084	0.1373	0.9157	0.977	**0.9454**
4	BAGGING	0.5589	0.1279	0.2082	0.7708	0.301	0.4329	1.0	0.0161	0.0317	0.6193	0.9948	0.7634
	BAGGING + META	0.5606	0.8794	0.6847	0.7951	0.354	**0.4899**	1.0	0.0161	0.0317	0.9157	0.977	**0.9454**
	STACKING	0.3691	1.0	0.5392	0.7767	0.3022	0.4351	0.1538	0.1083	0.1271	0.619	1.0	0.7647
	STACKING + META	0.5606	0.9108	**0.694**	0.7951	0.354	**0.4899**	0.1881	0.1088	**0.1379**	0.9157	0.977	**0.9454**

Table 5.8: The evaluation results of system aggregation strategies on the sub-set of the most effective systems. K denotes the number of the selected error detection systems, and the exact systems selection is shown in Table 5.7. The baseline is the *Precision Based Ordering* approach. The sample size to determine the sequence of error detection systems is the same as above: 1% of the results of all error detection systems. Source [232].

second series of experiments, we show that: *there is a sub-selection of constituents which is similarly effective as the complete set of error detection approaches.* The corresponding baseline is the *Precision Based Ordering (PBO)*, which was developed to select the most effective combination of error detection methods. The PBO results are shown in Table 5.8.

Selecting the most effective systems. We ran our experiments for different group sizes K ranging from 2 to $N-1$, where N is the total number of the participating error detection systems. The details of the selection of systems for each K is provided in Table 5.7. There is a clear selection preference for the outliers detection system dBOOST for the SALARIES dataset. For the setting $K = 2$, the *K-means* clustering algorithm created

two clusters: (1) NADEEF(D), dBOOST(Gauss), NADEEF(FD) and (2) WRANGLER + dBOOST(Hist). The precision values from Table 5.4 motivate the preference for the two dBOOST approaches. Setting K to 4 yields the following systems distribution: 1st cluster: dBOOST(Hist); 2nd cluster: NADEEF(D), NADEEF(FD); 3rd cluster: systems per 3 cluster: dBOOST(Gauss); 4th cluster: WRANGLER. Hence, we observe that NADEEF(FD) overlaps the NADEEF(D) system results, which supports the fact that the SALARIES dataset does not reveal duplicates, and therefore NADEEF(D) is the least effective system.

The most effective systems result combination. Given the selected sub-set of error detection systems as described above, we now apply our classification approach to combine results from these systems. This line of experiments is also proving the robustness of the ensemble learning approaches. The *stacking* approach appears to be robust for the settings $K = 2$ and $K = 3$ on the SALARIES dataset, whereas the *bagging* method was not able to train any effective classifier for $K = 2$ and $K = 3$. We explain such behaviour by the small error rate in this dataset - 2.33% - and the characteristics of the data errors. The SALARIES dataset mainly contains non-detectable numerical outliers on *basepay, overtime pay, other pay, benefits, total pay, total pay benefits* columns. For these reasons, the training data for $K = 2$ and $K = 3$ was highly sparse and not sufficient to build an error classification model.

We noticed that for the FLIGHTS and ADDRESS datasets the combination of two, three, or four error detection systems remains the same in terms of F_1 measure. This leads to the conclusion that adding more error detection systems will have minimal contribution to the improvement of error detection.

Extending the Best-K system combination with metadata-based features. In addition to the previous experiments on error classification, we augmented our core feature vectors of the *Best-K* systems with the metadata-based features. The results are shown in Table 5.8. These results support the experimental findings in Section 5.4.5 and state that augmenting feature vectors with metadata information will improve the overall performance of error classification. Therefore, by comparison to the combination methods without the metadata-based features, we are able to increase the F_1 by 16% on the ADDRESS dataset, by 5% on the HOSP dataset, by 1% on the SALARIES, and by 18% on the FLIGHTS dataset. Furthermore, we achieved significant improvement in error detection compared to the baseline PBO F_1-scores, namely 29% on ADDRESS, 8% on HOSP, 1% on SALARIES, and 19% on FLIGHTS.

5.5 Summary

As the experiments in Section 5.4 showed, using ensemble learning-based methods for aggregating data cleaning systems is an effective technique to improve overall error detection. Provided with several overlapping systems, we suggest leveraging the clustering-based method to specify the most useful combination of error detection systems. This will also reduce the complexity of the overall combination method, as fewer systems need to be integrated. Importantly, adding a metadata-based feature vector to the aggregation methods will provide significant improvement in error detection compared to the single error detection methods, since in this way the metadata can be incorporated into the aggregation process. The experimental results in this chapter support our findings from the previous Chapter 4 where we established the principal connection between metadata and data quality issues.

Basically, when labeling erroneous data is impossible, the only solution to aggregate error detection algorithms is to use unsupervised combination strategies, such as *Voting* [252], *Min-K* [2], or methods based on spectral analysis [63], as shown in Section 4.3.4. Generally, by applying ensemble learning, our methods capture considerably more errors than each error detection system individually. Following the augmentation of the feature vector with metadata-based features, the aggregation approach produces better error detection, as metadata reflects the characteristics of the dataset. Selecting the most effective error detection systems can produce similar performance as the complete set of all data cleaning systems. We showed that our method of combining error-detecting strategies achieves a higher F_1 score than single error detection approaches.

Modelling instance-based metadata features is a straightforward yet effective and intuitive process. However, incorporating schema-based metadata into the error classification feature vector remains challenging. Inter-column dependencies, such as *functional* or *matching dependencies*, are challenging to translate into a feature matrix $\mathcal{M}$. One way to encode functional dependencies is to use the notion of a *partition* of the functional dependency $\phi : X \rightarrow A$. Given that a dependency ϕ holds, then all dataset tuples that agree on X should also agree on A [82]. Therefore, assessing whether ϕ holds or not is performed by testing whether the dataset values of the attribute A agree on the right-hand side whenever they agree on the left-hand side X. The above assessment can be done by *partitioning* dataset tuples into disjoint sets (also called *equivalence classes*), such that each set has a unique value for the attribute set X [121].

The concept of *partitions* is similar to the concept of *clustering*, where each *cluster* represents one *partition* with respect to ϕ. Hence, functional dependencies might be featurized by using the *cluster membership* featurization technique [251]. However, we could run into another problem with the number of features. By encoding the functional dependency with the *cluster membership* technique, we might blow out the feature space.

For example, given that the LHS of ϕ is the primary key and its *uniqueness* ratio is one, then the number of acquired features is equal to N, where N=$|\mathcal{D}|$, the number of rows in the dataset $\mathcal{D}$. As a result, such feature space explosion might negatively impact the training process and the model's performance [251].

In the following chapter, we address this challenge by using a different approach. We apply statistical relational learning to precise modelling integrity constraints.

6

Probabilistic Data Curation Through Modelling Multi-column Metadata with Markov Logic

In this dissertation, we distinguish between five broad types of data cleaning techniques: *rule-based, statistical, hybrid, probabilistic & machine learning-based,* and *interactive* approaches. As introduced in Section 3.1, the ubiquitous *rule-based* data cleaning systems [80, 83, 87, 62, 91, 54] leverage miscellaneous integrity constraints to enable error detection and correction [223, 2]. This means that these systems require the specification of denial constraints, functional dependencies, and matching dependencies to specify data quality rules [54, 91, 62, 132, 202]. However, there are several limitations of rule-based data cleaning and to outline these limitations, we introduce an illustrative data cleaning scenario. Please note that in Section 2.2.2, we provided an overview of dependencies profiling, which is a foundation for our approach, as presented this chapter.

id	firstname	lastname	street	city	zipcode	phone
c_1	Ron	Howard	1 Sun Dr.	Los Angeles	90001	12345
c_2	Max	Miller	12 Hay St.	Napa	94558	11234

Table 6.1: CUSTOMER table (master data)

Example 6.0.1 *We first consider two relations: the* CUSTOMER *relation (Table 6.1), records the address and contact details of each customer. The* TRANSACTION *table (Table 6.2), lists each purchased item, together with the personal details entered by the customer during the purchase. The example data in the* TRANSACTION *table reveals at least three quality issues:*

id	item	price	type	firstname	lastname	street	city	zipcode	phone
t_1	iPhone6	500	phone	**R.**	Howard	1 Sun Dr.	**L.A.**	**null**	**null**
t_2	Galaxy5	600	phone	**null**	Miller	12 Hay St.	**null**	94558	11234
t_3	Nexus7	359	tablet	**Howard**	**Ron**	**null**	**null**	90001	12345

Table 6.2: TRANSACTION table (Erroneous values are marked in **bold**.)

1. ***Missing values****, indicated by* null *values;*
2. ***Wrong word order****, e.g., the customer "Ron Howard" is involved in transaction t_3, but his name is falsely recorded as "Howard Ron"; and*
3. ***Ambiguous values****, which are values that represent the same concept, e.g., the city of "Los Angeles" is sometimes entered into the table as "L.A." and the first name "Ron" is once abbreviated as "R.".* ■

Hence, the data values presented in the TRANSACTION Table 6.2 are corrupted. To correct these issues, we need to identify the corrupted entries for each transaction. Then we need to use master data from the CUSTOMER table to automatically clean the transaction data by applying data cleaning rules (see Section 2.1.3). Defining automatic cleaning rules to accomplish this task is not trivial. For instance, a rule, which states that the *city* attribute values "L.A." and "Los Angeles" always denote the same, intuitively makes sense. However, for the *firstname* values "R." and "Ron", we would rather need a *soft rule* that indicates that both strings *possibly* refer to the same entity.

To capture errors in attributes of the TRANSACTION relation, we might use functional dependencies [7]. The following data cleaning rule, which is based on a functional dependency, declares that the two fields *city* and *phone* in the TRANSACTION table together uniquely determine the two fields *street* and *zipcode*:

$$fd : \text{TRANSACTION}([city, phone] \rightarrow [street, zipcode])$$

Although two instances of the customer "Ron Howard" are recorded in Table 6.2, one is missing the value of the attribute *phone*. In this case, the rule only applies when combined with additional data cleaning rules that impute missing values.

The next rule is specified by using conditional functional dependency (CFD) [82]:

$$cfd : \text{TRANSACTION}([zipcode] \rightarrow [city], T_1 = (90001 \mid\mid \text{Los Angeles}))$$

The above rule states that every tuple in which the value for *zipcode* equals 90001 must have its *city* attribute set to "Los Angeles". In our example, in Table 6.2, this rule imputes the *null* value in the transaction tuple t_3. Provided the rule, which states that the *city*

attribute values "L.A." and "Los Angeles" are always the same, it would correct the *city* attribute to "Los Angeles" in the tuple t_1.

We might also use master data from the CUSTOMER relation to correct *"ambiguous values"*. In the context of data cleaning, *matching dependencies* have been exploited to detect tuples referring to the same real-world entities [81]. To identify matching entities, for the *firstname* attribute in transaction t_1 in Table 6.2, we could define a rule indicating that the values "R." and "Ron" refer to the same name, but cannot always be certain whether this is the case. To match transaction tuple t_1 to the customer tuple c_1, the above rule might be formulated as the following matching dependency (MD) [82, 81]:

$$\begin{aligned} md : \text{TRANSACTION}[lastname, city, street] = \text{CUSTOMER}[lastname, city, street] \\ \wedge \text{TRANSACTION}[firstname] \approx \text{CUSTOMER}[firstname] \\ \rightarrow \text{TRANSACTION}[firstname] \rightleftharpoons \text{CUSTOMER}[firstname] \end{aligned}$$

This rule shows how the notion of similarity may be included in matching rules, but only within first-order logic, i.e., the similarity condition is either *true* or *false*. In reality, we might need to determine a more fine-grained similarity, i.e., some types of similarity that we deem to be more probable ("L.A." and "Los Angeles"), and others that are less probable ("R." and "Ron"). Moreover, we may have different levels of confidence regarding the functional dependencies or matching dependencies that we define.

Triggered by the examples above, the research challenges, which are addressed in this chapter, are the following:

Interleaved Rules. Each data quality issue is addressed by at least one data quality rule. However, these data quality rules typically interact with each other because the data quality issues are interacting [82, 87]. During automatic data cleaning, the optimal execution order of data cleaning rules is difficult to achieve [62, 21]. This problem contradicts the automation principle of data curation systems [222]. Hence, we are confronted with the challenge of the optimal order execution of data cleaning rules that maximizes error detection and error correction.

Usability and Domain Knowledge Integration. Previous research in data quality management resulted in several data cleaning approaches, algorithms, and systems [62, 190, 91, 245, 54, 131]. These approaches typically address the detection and repair of specific error types. However, we are facing the challenges of data cleaning system revealing that (1) data cleaning systems should maximally cover the spectrum of data quality issues, and (2) these systems should cover the maximum spectrum of customized rules without having to specify user-defined functions [62].

Probabilistic Data Cleaning Most data cleaning systems produce a single clean database instance by following the *minimality repair* principle [52, 136, 156] (see Section 2.1.3). However, the principle of *minimality* does not consider the likelihood of the *repair* [68]. Therefore, the challenge is how to define the most probable data repair [27, 26].

To address these challenges, we suggest to model data quality rules *jointly*, rather than as separate tasks. Additionally, data quality rules might need to be specified approximately, meaning be declared as *"soft"* or *"hard"* rules. We present our approach to data cleaning based on the Statistical Relational Learning (SRL) formalism called Markov logic [73] and probabilistic inference [204]. In this chapter, we propose a method that (1) utilizes the probabilistic joint inference over interleaved data cleaning rules to improve data quality; (2) remove the need to specify the order of rule execution; and (3) expresses data quality rules as a first-order logic formula to directly translate into the predictive model in our SRL framework.

Please note that the preliminary material, which is relevant for this chapter is provided in Sections 2.2 and 2.3. This chapter is organized as follows:

1. In Section 6.1, we describe the specification of the data cleaning rules based on integrity constraints.
2. In Section 6.2, we propose a data cleaning approach based on Markov Logic formalism.
3. In Section 6.3, we define data cleaning as the probabilistic inference problem.
4. In Section 6.4, we extend our data cleaning approach to non-relational data.
5. The experimental results are provided in Section 6.5.
6. Finally, we summarize our SRL-based data cleaning approach in Section 6.6.

6.1 Integrity Constraints as Data Quality Rules

One of the most essential questions in data cleaning is how to specify that the data is clean. Measuring data quality requires the specification of data quality constraints [34]. In this section, we make a connection between data quality rules and integrity constraints.

As already discussed in Chapter 2, integrity constraints define the semantics of data in a declarative way. The violation of data quality dimensions is often expressed in the violation of the integrity constraints [80]. Furthermore, all five *data quality dimensions* (see Chapter 2.1.1) can be defined in terms of data dependencies [82]. Therefore, data quality rules can be naturally declared on such integrity constraints that have been discovered from the data [182, 23]. Practically, canonical integrity constraints, such as

Data Quality Dimension	Integrity Constraint (captures the DQ dimension violation)	Relaxation Criteria or Operator
Accuracy	CFD [83] Conditional Functional Dependency eCFD [34] Extended CFD CFD^{p} [47] CFD with built-in predicates MFD [139] Metric Functional Dependency	Pattern tableau Constraint (logical) operator
Consistency	MFD [139] Metric Functional Dependency CIND [47] Conditional Inclusion Dependency	Pattern tableau Constraint (logical) operator
Uniqueness	MD [81] Matching Dependency CMD [217] Conditional Matching Dependency DD [216] Differential Dependency	Matching operator Similarity function Closeness function Difference function
Completeness	CFD [83] Conditional Functional Dependency CIND [47] Conditional Inclusion Dependency	Pattern tableau
Timelines	CC [82] Currency constraint	Order operator

Table 6.3: Mapping five central data quality dimensions to integrity constraints. The methodology for this mapping is the following: if the integrity constraint captures the violation of the respective data quality dimension, then the connection between the dimension and the integrity constraint is established.

functional dependencies, are insufficient to improve data quality [83, 43] because they target improving the quality of the database schema [7] and are not able to capture data quality requirements for particular data values [24]. For this reason, to improve data quality, we consider a broad spectrum of dependencies [43], which declaratively address the corresponding data quality dimension. Table 6.3 summarizes *canonical* and *relaxed* integrity constraints, which are relevant for data quality management, as outlined by Caruccio L. et al. [43]. Moreover, this table shows the integrity constraints to capture the violation of the respective data quality dimension, such as *accuracy, consistency, uniqueness, completeness*, or *timelines* [82]. For instance, to capture the violation of the *accuracy* dimension, we identified the following group of integrity constraints: *CFD, eCFD, CFD^{p}*, and *MFD* [43]. These integrity constraints target data errors, such as *incorrect data, misspellings, misfielded values, missing data, different aggregation level*, or *FD violation*. For a detailed overview of the mapping between the data quality dimensions and data errors that violate those dimensions, please refer to Table 2.1.

6.2 Modelling Data Quality Rules as Markov Logic Programs

After specifying dependency constraints for data quality rules, we deal with the next question regarding *the effective use of data quality rules to improve the data quality along*

the five central data quality dimensions. In the following, we explain how to convert them into a logical framework [79], to address challenges related to the *rule-based* data cleaning approaches.

To address the challenges of this chapter, such as *interleaved rules formulation, automation of rules execution, domain knowledge integration*, and *probabilistic data cleaning*, we need a formalism, which unites *declarativity* and *uncertainty* for data cleaning rules formulation, and *probabilistic inference* for deriving probable repair.

One of the possible solutions to these challenges is to apply *Statistical Relational Learning (SRL)* to solve the above-mentioned challenges. SRL combines (1) *statistical learning*, which addresses uncertainty in data by applying statistical methods, and (2) *relational learning*, which describes complex relational structures between data attributes in a general manner (e.g by using first-order logic) [93]. Currently, the research community has proposed different SRL models, such as Markov Logic [73], Relational Markov Models [11], Inductive Logic Programming [130], and Relational Dependency Networks [176]. Please refer to Section 2.3 for the introduction and preliminaries of SRL.

We propose to use Markov logic [73] as our representation language for data cleaning [234]. We argue that Markov logic is a natural fit for modelling interacting data quality rules in a flexible and extensible way. The Markov logic formalism satisfies a number of prerequisites, which address the data cleaning challenges as mentioned at the beginning of this chapter.

1. Markov logic incorporates both first-order logic and probabilistic graphical models to perform the probabilistic inference [93], which addresses the *interleaved rules* and *probabilistic data cleaning* challenges.

2. Markov logic enables a clear and straightforward representation of the SRL problem and facilitates the use of domain knowledge [93], which addresses the *usability and domain knowledge integration* problem.

The core of our approach is compiling data quality rules into the Markov logic formalism [73, 93], which interprets arbitrary first-order logic formulae in a probabilistic way. The high-level overview of our approach is shown in Figure 6.1. Our method includes three main components:

1. An *Evidence and Query* component that includes a number of data sources, including relational and semi-structured data, as well as auxiliary information, such as similarity lexica, gazetteers or dictionaries. The *Query* component consists of queries about potential errors and probable correction of corrupted data.

2. A *Model and Inference* component includes a *Model* that expresses general knowledge about correct data, which is based on the data quality constraints and domain-specific

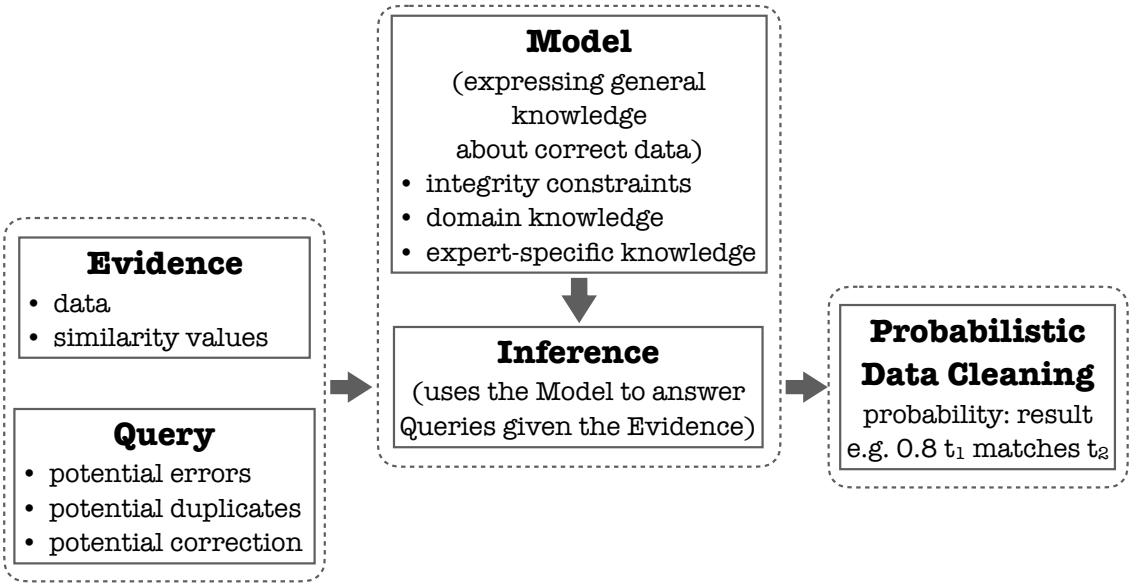

Figure 6.1: Overview of the proposed probabilistic data cleaning approach.

knowledge to address different data quality issues. The *Inference* component uses the previously specified *Model* and runs the inference algorithm. This component answers queries about potential errors and probable repairs of dirty data given the evidence from the *Evidence and Query* component. The *Model and Inference* component is based on probabilistic inference performed by the Markov logic framework.

3. A *Probabilistic Data Cleaning* component is an error detection and repair prediction component that is based on probabilistic inference results performed by the *Model and Inference* component.

Generally, we regard the *"dirty"* dataset as evidence and as a *"dirty"* version of a hidden clean dataset [125, 68]. We also consider multiple signals, such as external master data, as evidence.

In this chapter, we show that data quality rules, which are expressed as integrity constraints, can be translated into first-order logic sentences [79]. After that, we compile acquired data cleaning rules and domain-specific information into the Markov logic program. We reason about the irregularities in the data and possible repair by running a probabilistic inference on the created model given the evidence. In the following, we describe each component of our approach in more detail.

6.2.1 Mapping Data Cleaning Concepts to Markov Logic Predicates

As already mentioned in Section 2.3.1, the Markov logic interface consists of first-order logic sentences. To formulate data cleaning routine as a Markov logic program, we need

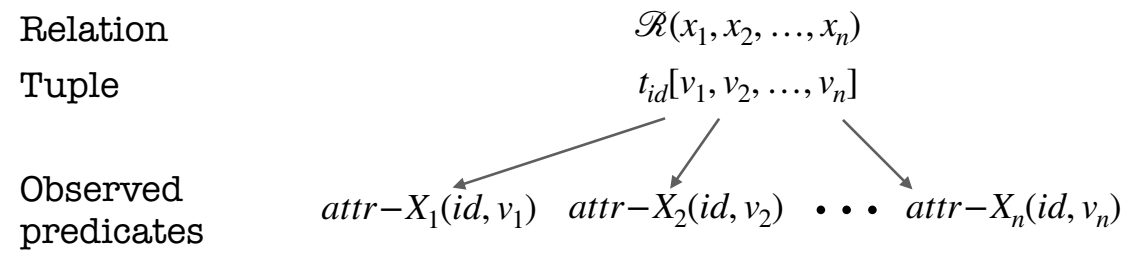

Figure 6.2: Specification of the observed predicates. A relation tuple is translated into n atomic sentences.

to compile data cleaning rules as the Markov logic formulae. To create these formulae, we would need to define predicates initially. These predicates then construct the above first-order logic sentences. In addition to the Markov logic preliminaries that have been explained in Section 2.3.1, we define a concrete *vocabulary* for the compilation of data quality rules into the Markov logic formalism.

The basis of the Markov logic programs is the *predicate calculus* [92], which is later used to build the data quality rules. We distinguish between *hidden* and *observed* predicates to designate the concepts used in data quality rules. First, we define our dataset as *observed predicates* and then we propose the formulation of data quality concepts as *hidden predicates*.

A ground predicate whose state is known (i.e., true or false), is called an *observed predicate*. To express a tuple in the relation $\mathcal{R}(x_1, x_2, \ldots, x_n)$ as observed predicates, we define n atomic sentences, such as $attr\text{-}X_1(id, v_1), \ldots, attr\text{-}X_n(id, v_n)$, where $attr\text{-}X_i(id, v_i)$ denotes the attributes value v_i of the i-th column in the id-th row in relation $\mathcal{R}$, as shown in Figure 6.2. Generally, all predicates that are specified for the relation tuples and all data defined in the *Evidence* component of our approach are observed.

Hidden predicates are ground predicates with unknown states. Besides the representation of the tuple values, we formulate concepts, such as *similarity, equality*, or *matching*, as *hidden predicates* to reason about them during the inference phase. These concepts are an integral part of canonical and approximate integrity constraints, as highlighted in Table 6.3. The set of Markov logic predicates as a representation of data cleaning concepts, relaxation criteria and operators is summarized in Table 6.4. Importantly, all these predicates are defined in the *Query* component of our approach. In the following, we list the above concepts and the corresponding Markov logic predicates:

- **Similarity.** The $similar\text{-}X(id_i, id_j)$ predicate denotes the similarity of two values of the id_i-th and id_j-th tuples of the attribute X. The similarity is calculated according to some particular similarity metric, such as *cosine, Jaccard*, or *euclidean* [78]. The similarity predicate represents a similarity relation and therefore is symmetric, reflexive and domain-specific.

Concept	Operator	Markov Logic Predicate
Similarity	$t_i[y] \approx t_j[y]$	*similar-Y*(id_i, id_j)
Equality	$t_i[y] = t_j[y]$	*equal-Y*(id_i, id_j)
Non-equality	$v_i < v_j$	e.g. *lessThan*(v_i, v_j)
Matching	$\mathcal{R}_1[y] \rightleftharpoons \mathcal{R}_2[y]$	$R1/$*match-Y*$/R2(id_i, id_j)$
Pattern tableau	$t_i[X] =$ "100"	*tableau-X*$(i, 100)$
Currency [82]	$t_i \prec_X t_j$	*moreCurrent-X*(id_i, id_j)

Table 6.4: Mapping Markov logic predicates to data quality concepts Hidden predicates summary.

- **Equality.** The *equal-X*(id_i, id_j) predicate denotes that, given two tuples with ids id_i and id_j, the values of the attribute X are equal.
- **Non-equality.** Data cleaning rules might use a set of non-equality operators. This set includes $\{<, \leq, >, \geq\}$ and the corresponding predicates are the following: *lessThan*(v_i, v_j), *lessThanEq*(v_i, v_j), *greaterThan*(v_i, v_j), *greaterThanEq*(v_i, v_j).
- **Matching.** The *match-X*(id_i, id_j) predicate is defined on a single relation $\mathcal{R}$ and determines two tuples, id_i and id_j of the attribute X, which are identified to match. If the matching operator is specified on two relations, such as master data relation $\mathcal{M}$ and the dataset $\mathcal{R}$, then the Markov logic predicate is designated as $M/$*match-Y*$/R(id_i, id_j)$. Please note that the *matching* operator is used in *matching dependencies*. The respective definitions can be found in Section 2.2.2.
- **Pattern tableau.** The *tableau-X*(id, v) predicate encodes the constraint value v on the attribute X. As shown in Table 6.4, the constraint on the attribute X, such as $t_1[X] =$ "100", is translated into the following predicate *tableau-X*$(1, 100)$. Please note that the concept of *pattern tableau* is similar to the *equality* concept. The distinction is that the right-hand side (RHS) of the *pattern tableau* operator is a constant value.
- **Order operator.** This operator is used for assessing the *currency* quality dimension and specifying the currency order. The predicate *moreCurrent-X*(id_i, id_j) formulates that for the given attribute X, the value of the id_j-th tuple is more current than the value of the id_i-th tuple.
- **Custom predicate.** This predicate encodes auxiliary constraints or domain knowledge that should be incorporated into the Markov logic program for data cleaning. For example, the *synonyms*(v_i, v_j) predicate tests whether two values v_i and v_j are synonyms.

To conclude, we now have a *predicate calculus* to express data cleaning rules. These predicates will be used in the Markov logic programs for data quality, as provided in the next section.

6.2.2 Data Quality Constraints as Markov Logic Program

In this section, we formulate the data cleaning problem as an SRL problem and provide an algorithm to translate data quality rules into the Markov logic by using the *predicate calculus* for data cleaning. To detect inconsistencies in data, we usually define data cleaning rules in the form of integrity constraints, such as *functional, matching,* and *inclusion dependencies*, as specified in Table 6.3. To define data cleaning workflow as Markov logic program, we first examine how to translate each individual integrity constraint, such as *CFDs/FDs, CMDs/MDs*, and *CINDs/INDs* into Markov logic formulae.

Translating Functional Dependency as a Markov Logic Formula: Given a functional dependency $\phi : X \rightarrow Y$, we use the methodology provided by Fagin R. [79] and Nicolas J. [177] and express ϕ as a *first-order logic* sentence [79, 40, 177] in the following way:

$$\forall\, \mathsf{x}, \mathsf{y_1}, \mathsf{y_2}, \mathsf{z_1}, \mathsf{z_2}\ \mathcal{R}(\mathsf{x}, \mathsf{y_1}, \mathsf{z_1}) \wedge \mathcal{R}(\mathsf{x}, \mathsf{y_2}, \mathsf{z_2}) \Rightarrow \mathsf{y_1} = \mathsf{y_2} \tag{6.1}$$

Given this representation and the *predicate calculus* provided in the previous Section 6.2.1, we describe our conceptualization of the data quality rules with predicate-calculus sentences. In general, we propose Algorithm 6.1, to convert data cleaning rules based on CFDs/FDs into the Markov logic formulae. This algorithms takes a set of CFDs/FDs as input and returns a set of corresponding Markov logic formulae. After formulating the integrity constraint as first-order logic formula, we translate it into the Markov logic syntax by applying the following convention:

- Every tuple of the relation $\mathcal{R}(x_1, x_2, \ldots, x_n)$ in the RHS of ϕ is translated into n *observed predicates*, such as $attr\text{-}X_1(id, v_1), \ldots, attr\text{-}X_n(id, v_n)$ (Algorithm 6.1 code lines 11-14, and Figure 6.2).
- Every data quality concept in the left-hand side (LHS) of ϕ is translated as *hidden predicates* according to Table 6.4 (Algorithm 6.1 code lines 15-17). For example, $y_1 = y_2$ is expressed as $equal\text{-}Y(id_1, id_2)$.

Now, we use the *predicate calculus* provided in Section 6.2.1 to compile the above functional dependency $\phi : X \rightarrow Y$ and the corresponding first-order logic sentence in Formula 6.1 to the Markov logic formula in the following way:

$$attr\text{-}X(id_1, x) \wedge attr\text{-}X(id_2, x) \Rightarrow equal\text{-}Y(id_1, id_2)$$

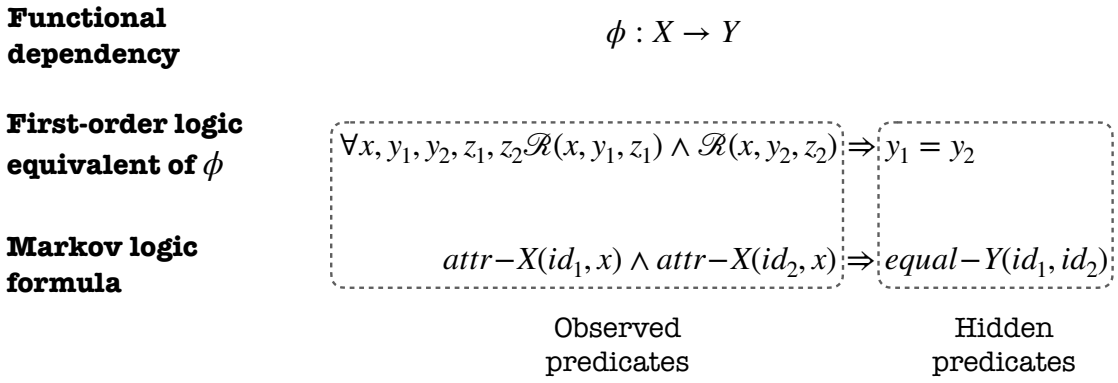

Figure 6.3: Specification of data quality rules with predicate-calculus sentences. This Figure shows all stages of the translation of functional dependencies into the Markov logic formalism.

See also Algorithm 6.1 code lines 18-19 for FD rule compilation into the Markov logic formula. Additionally, the above translation is also shown in Figure 6.3.

For example, to demonstrate the compilation of FD to the Markov logic formula, we consider the FD-rule from the motivation of Example 6.0.1, which states that if any two tuples agree on attribute values for *city* and *phone*, then the attribute values in *street* and *zip* should agree as well:

$$\phi : \text{TRANSACTION}([city, phone] \rightarrow [street, zipcode])$$

To enable straightforward translation into the first-order logic, we assume that FDs are provided in the normal form [82]. This means that if the FD is formulated as $\psi : (X \rightarrow Y_1, Y_2, \ldots, T_p)$, then ψ will be decomposed into several FDs, where $RHS(\psi)$ (right hand side of ψ) becomes a single attribute: $\psi_1 : (X \rightarrow Y_1, T_p)$, $\psi_2 : (X \rightarrow Y_2, T_p) \ldots$. Following the normalization principle for functional dependencies, we split the ϕ rule into two rules and write them as a CFDs, because canonical FDs are a special case of CFDs, in which the pattern tableau are empty and contain "_" [82]:

$$\mathsf{cfd}_1 : \text{TRANSACTION}([city, phone] \rightarrow [street], \mathsf{t}_1 = (\_, \_ \parallel \_))$$
$$\mathsf{cfd}_2 : \text{TRANSACTION}([city, phone] \rightarrow [zipcode], \mathsf{t}_2 = (\_, \_ \parallel \_))$$

Algorithm 6.1 Conditional Functional Dependencies Compilation to Markov Logic Rules

function TRANSLATECFDSTOMARKOVLOGIC(F) ▷ F is a set of normalized (C)FDs
  $T \leftarrow []$
  $A \leftarrow []$
  $e \leftarrow \emptyset$
  $R \leftarrow []$
  for $f \leftarrow F$ **do**
    for $i \leftarrow$ *attributes in tableau in* f *of the form* $'attr = const'$ **do**
      $T \leftarrow T +' attr\text{-}i(id_1, const)'$
      $T \leftarrow T +' attr\text{-}i(id_2, const)'$
    end for
    for $j \leftarrow$ *attributes in LHS of* f **do**
      $A \leftarrow A +' attr\text{-}j(id_1, v)'$
      $A \leftarrow A +' attr\text{-}j(id_2, v)'$
    end for
    for $l \leftarrow$ *attribute in RHS of* f **do**
      $e \leftarrow' equal\text{-}l(id_1, id_2)'$
    end for
    $rule_f \leftarrow \wedge_{t \in T}\ t \wedge_{a \in A}\ a \Rightarrow e$
    $R \leftarrow R + rule_f$
  end for
  return R ▷ The set of Markov logic formulae
end function

Following the conversion rules formulated by Fagin R. [79] and Nicolas J. [177], we represent cfd_1 and cfd_2 as two first-order logic formulae:

$$1)\ \forall\ city, phone, street_1, street_2\ \text{TRANSACTION}(city, phone, street_1) \wedge$$
$$\text{TRANSACTION}(city, phone, street_2) \Rightarrow street_1 = street_2$$
$$2)\ \forall\ city, phone, zip_1, zip_2\ \text{TRANSACTION}(city, phone, zip_1) \wedge$$
$$\text{TRANSACTION}(city, phone, zip_2) \Rightarrow zip_1 = zip_2$$

Provided that every attribute from the schema TRANSACTION can be expressed as a first-order logic predicate, we formulate two predicates, namely *city*(id, city) and *phone*(id, phone) to encode the LHS of ϕ. They indicate the values for the fields *city* and *phone* for each tuple (Table 6.5 shows the full example of observed predicate definition and data presentation as grounded atoms). Furthermore, we define two additional predicates for our data quality rule, namely *equal-street*(id, id) and *equal-zip*(id, id). These predicates model the equality of two values of the attribute *street*, respectively *zip* (as denoted by the ϕ rule above). The

Phase	Example
1) Schema definition	t_2(item, firstname, lastname, street, city, zipcode, phone)
2) Observed predicates MLN declaration	*firstname*(id, firstname) *lastname*(id, lastname) *street*(id, street) *city*(id, city) *zip*(id, code) *phone*(id, num)
3) Data	t_2(Galaxy 5, NULL, Miller, 12 Hay St., NULL, 818, 11234)
4) Grounded (evidence) atoms	*item*(2, Galaxy5) *lastname*(2, Miller) *street*(2, 12HaySt.) *zip*(2, 818) *phone*(2, 11234)

Table 6.5: MLN declaration process and creation of grounded atoms for Tuple 2 in the TRANSACTIONS example table.

first-order logic formulae above are now expressed in the predicate calculus as follows:

$$1)\mathit{city}(\mathsf{id}_1, \mathsf{city}) \wedge \mathit{city}(\mathsf{id}_2, \mathsf{city}) \wedge \mathit{phone}(\mathsf{id}_1, \mathsf{phone}) \wedge \mathit{phone}(\mathsf{id}_2, \mathsf{phone}) \Rightarrow \mathit{equal\text{-}street}(\mathsf{id}_1, \mathsf{id}_2)$$

$$2)\mathit{city}(\mathsf{id}_1, \mathsf{city}) \wedge \mathit{city}(\mathsf{id}_2, \mathsf{city}) \wedge \mathit{phone}(\mathsf{id}_1, \mathsf{phone}) \wedge \mathit{phone}(\mathsf{id}_2, \mathsf{phone}) \Rightarrow \mathit{equal\text{-}zip}(\mathsf{id}_1, \mathsf{id}_2)$$

The two different tuples are distinguished by providing their identifiers, such as id_1 and id_2. Having declared the data quality rules, we infer potential predicates, such as *equal-street*(id, id). This is a *hidden* predicate and holds the information about possible repairs on the attribute *street* in the TRANSACTIONS table. Reasoning about such hidden predicates gives us a probabilistic value of attributes that need a particular repair.

Translating Matching Dependency as a Markov Logic Formula. To capture duplicate entities in the dataset, we formulate data quality rules by using *matching dependencies* (MDs) [81]:

$$\mu : \mathcal{S}_1[x_1] \approx \mathcal{S}_2[x_2] \rightarrow \mathcal{S}_1[y_1] \rightleftharpoons \mathcal{S}_2[y_2]$$

MDs are defined in terms of the matching operator $\rightleftharpoons$ and *similarity predicates*, such as $\mathcal{S}_1[x_1] \approx \mathcal{S}_2[x_2]$. Correspondingly to the compilation steps for *functional dependencies*, we use the *predicate calculus* provided in Section 6.2.1 and Table 6.4 and translate a matching dependency μ on a database instance $\mathcal{D}$, with two relational schemas $\mathcal{S}_1$ and $\mathcal{S}_2$, into the

Markov logic formula as follows:

$$\textit{attr-X/S1}(id_1, x_1) \wedge \textit{attr-X/S2}(id_2, x_2) \wedge \textit{similar}(x_1, x_2) \Rightarrow \textit{S1/match-Y/S2}(id_1, id_2)$$

The general procedure of matching dependencies compilation to Markov logic formulae is shown in Algorithm 6.2. This Algorithm takes a set of CMDs/MDs as input and returns a set of corresponding Markov logic formulae. The translation of each CMD' LHS into the *predicate calculus* is provided in code lines 7-18, where every tuple of the relation $\mathcal{S}(x_1, x_2, \ldots, x_n)$ in the RHS of an CMD is translated into n *observed predicates*, such as $\textit{attr-X}_1(id, v_1), \ldots, \textit{attr-X}_n(id, v_n)$. The *similarity predicate* $\mathcal{S}_1[x_1] \approx \mathcal{S}_2[x_2]$ is resolved as $similar(x_1, x_2)$ in code lines 14-18.

Algorithm 6.2 Conditional Matching Dependencies Compilation to Markov Logic Rules

```
1:  function TranslateCMDsToMarkovLogic(M)
2:      E ← []
3:      S ← []
4:      m ← ∅
5:      R ← []                                   ▷ R is set of Markov logic formulae
6:      for f ← M do
7:          for i ← tableau attributes in LHS(f) such as 'S1[x] = const' do
8:              T ← T +' attr-i/S1(id1, const)'
9:          end for
10:         for j ← equalityfunctions in LHS(f) such as S1[x] = S2[x] do
11:             E ← E +' attr-j(id1, v)'
12:             E ← E +' attr-j(id2, v)'
13:         end for
14:         for l ← similarity functions in LHS(f) such as S1[y] ≈ S2[y] do
15:             S ← S +' attr-l(id1, v1)'
16:             S ← S +' attr-l(id2, v2)'
17:             S ← S +' similar(v1, v2)'
18:         end for
19:         for k ← matching operator in RHS(f) such as S1[y] ⇌ S2[y] do
20:             m ←' S1/match-Y/S2(id1, id2)'
21:         end for
22:         rule_f ← ∧t∈T t ∧e∈E e ∧s∈S s ⇒ m
23:         R ← R + rule_f
24:     end for
25:     return R                                 ▷ The set of Markov logic formulae
26: end function
```

The RHS of the *matching dependency* μ contains the matching operator $\mathcal{S}_1[y_1] \rightleftharpoons \mathcal{S}_2[y_2]$, which is translated into the hidden predicate $\textit{S1/match-Y/S2}(id_1, id_2)$. It denotes *dynamic* semantic [82, 24] of the matching dependency and indicates the values of the tuples id_1 and id_2 of the attribute Y that must be made equal (Algorithm 6.2 code lines 19-21).

Translating Conditional Inclusion Dependency as a Markov Logic Formula. To capture the inconsistencies between tuples across different relations, we formulate data cleaning rules based on interrelational dependencies, namely *(conditional) inclusion dependencies* (CINDs/INDs). To demonstrate how we translate CINDs/INDs into Markov logic formulae, first, consider the following example of CIND:

$$\gamma : \textsc{Transaction}[item, price, type =' phone'] \subseteq \textsc{Phone}[item, price]$$

The γ means that the attribute *type* is *'phone'* for each transaction entry, so there should be a corresponding entry in the PHONE relation. According to Bertossi L. [24], the above CIND γ can be rewritten in first-order logic, as follows:

$$\forall i, p, t \ \textsc{Transaction}(i, p, t) \wedge t =' phone' \Rightarrow \textsc{Phone}(i, p)$$

Similarly to the previous compilation procedures, we use the *predicate calculus* provided in Section 6.2.1 and Table 6.4 and translate the above CIND γ into the Markov logic formulae:

$$\textit{attr-I/T}(id_1, i) \wedge \textit{attr-P/T}(id_1, p) \wedge \textit{attr-T/T}(id_1, \text{"}phone\text{"}) \Rightarrow \textit{attr-I/P}(id_2, i) \wedge \textit{attr-P/P}(id_2, p)$$

Although the Markov logic formula above does not contain any hidden predicates, the whole formula must be satisfiable for the corresponding CIND to hold on the data. The general procedure of CINDs compilation into Markov logic program is provided in Algorithm 6.3. This Algorithms takes a set of CINDs/INDs as input and returns a set of corresponding Markov logic formulae. The translation of the γ LHS into the *predicate calculus* is provided in lines 7-13. The translation of the γ RHS is performed in lines 14-17.

From the previously described examples and algorithms, we see that, since integrity constraints can be seen as a set of sentences written in first-order logic [79, 177, 24], they can be transferred into the Markov logic formalism. Generally, a particular advantage of Markov logic is its *modularity* while modelling data cleaning operations. We consider each declared data quality rule (represented as CFDs/FDs, CMDs/MDs, or CINDs/INDs) as a single-unit probabilistic model that can be combined to a "compound" model [234].

In the next section, we present the mechanism behind the inference of the hidden predicates.

6.3 Uncertain Data Cleaning as a Probabilistic Inference Problem

In this section, we make a connection between data cleaning and probabilistic inference.

Algorithm 6.3 Conditional Inclusion Dependencies Compilation to Markov Logic Rules

function TRANSLATECINDSTOMARKOVLOGIC(C)
  $T \leftarrow []$
  $A \leftarrow []$
  $D \leftarrow []$
  $R \leftarrow []$ ▷ R is set of Markov logic formulae
  for $f \leftarrow C$ **do**
    for $i \leftarrow$ *tableau attributes in* $LHS(f)$ *such as* $'S_1[x] = const'$ **do**
      $T \leftarrow T +' attr\text{-}i/S1(id_1, const)'$
    end for
    for $j \leftarrow$ *attributes in* $LHS(f)$ **do**
      $A \leftarrow A +' attr\text{-}j/S1(id_1, v)'$
      $A \leftarrow A +' attr\text{-}j/S1(id_2, v)'$
    end for
    for $l \leftarrow$ *attributes in* $RHS(f)$ **do**
      $D \leftarrow D +' attr\text{-}l/S2(id_1, v)'$
      $D \leftarrow D +' attr\text{-}l/S2(id_2, v)'$
    end for
    $rule_f \leftarrow \wedge_{t \in T}\ t \wedge_{a \in A}\ a \Rightarrow \wedge_{d \in D}\ d$
    $R \leftarrow R + rule_f$
  end for
  return R ▷ The set of Markov logic formulae
end function

By assuming that real-world data is usually dirty [201, 199, 223], we specify data errors as a violation of integrity constraints. We view data cleaning as a probabilistic inference, which results in multiple possible repairs [27, 26]. This is because the data cleaning principle of *minimality* (see Section 2.1.3) does not consider the likelihood of the *possible repair* [68], and the *minimal repair* is not necessarily the correct repair [202, 68]. We follow the idea of the *most-probable database (MPD)* that denotes a database instance, which satisfies a number of data quality constraints [101]. Hence, we define data cleaning as a probabilistic inference problem that infers the most probable clean database given the initial database and attribute correlations, which are expressed as integrity constraints. We use the notion of the *Most Probable Explanation (MPE)* task from statistical relational learning [227, 93] and adapt it to data cleaning to derive the *most probable repair*, which is a repaired dataset, given a set of integrity constraints and domain-specific knowledge.

Definition 6.3.1 *Given a database $\mathcal{R}$ that represents a probability distribution $P(\cdot)$ and a set of integrity constraints $\boldsymbol{I}$, which are formulated in first-order logic sentences, the Most Probable Repair (MPR) $\mathcal{R}'$ is defined as*

$$MPR(I) = \arg\max_{\mathcal{R}' \models I} P(\mathcal{R}'|\mathcal{R}) \ \blacksquare$$

Given a dirty dataset, we can compute a most probable repair that adheres to the provided data constraints $\mathbf{I}$ in first-order logic form [79, 177]. *Most Probable Explanation* thus provides a principled probabilistic framework for data cleaning.

Formally, we reduce the MPR to MPE that can be adopted to solve the MPR problems by applying existing algorithms for probabilistic inference. For any MPR problem, we can formulate a Markov logic program as proposed in Section 6.2.2 and compute the MPR as a most probable state of the MLN model [101]. Semantically, a MLN is a log-linear model [73], which defines the probability distribution over possible repairs in the database. The general procedure of the MLN formulation and MPR inference is provided in Algorithm 6.4. The algorithm takes as input two arguments: a dataset $\mathcal{R}$ and the set of data cleaning rules expressed as integrity constraints. As a result, the algorithm returns a set of grounded hidden predicates, which are candidates for data repair.

We formulate MLNs by specifying a set of first-order logic sentences with weights by using predicates that represent relations between attribute values. To specify *soft* and *hard* rules, we set the weight of *soft* rules to 1.0 [100], whereas *hard* rules are assigned infinite weights. As mentioned earlier, we distinguish between observed and hidden predicates. *Observed predicates* define relations between objects which exist in a given dataset, such as $attr\text{-}X_1(\mathsf{id}, \mathsf{v_1}) \ \ldots \ attr\text{-}X_\mathsf{n}(\mathsf{id}, \mathsf{v_n})$. Furthermore, we define a number of *hidden predicates*, which present data cleaning concepts and are not present in the evidence. The specification of *observed* and *hidden* predicates is provided in Algorithm 6.4 code lines 6-11. These *hidden* predicates will be inferred through the probabilistic inference. Concretely, all Markov logic predicates defined in Table 6.4 are hidden. Given the initial dataset $\mathcal{R}$, we can compute the most *probable database repair* that adheres to the data constraints IC, which were defined as data quality rules. In other words, we perform an inference task on hidden predicates as a prediction for data cleaning. The probabilistic inference consists of *grounding* and *search* steps [178]. For the grounding, we take the content of the database (a set of tuples) and produce a set of *grounded* predicates by replacing predicate variables with domain constants as shown in Figure 6.4. These groundings are then used in the *search* step, which is joint inference of the most probable state for data cleaning.

While modelling data quality constraints as MLN, as provided in Algorithm 6.4 code lines 13-24, we denote $h \in \mathcal{H}^n$ as a hidden predicate with n *literals* (random variables) $H_1, ..., H_n$, where each literal H_i has 2 discrete states, $H_i = \{0, 1\}$ (0 denotes *false* and 1 - *true*). We similarly define $o \in \mathcal{O}^m$ as an observed predicate with m *literals* (random variables) $O_1, ..., O_m$. In this way, the specified MLN represents a joint distribution on $H_1, ..., H_n, O_1, ..., O_n$ random variables that is specified by a vector $\phi(h, o)$ of d integer values, where each element represents the number of true groundings of the corresponding literal in the formula. We specify a weight/parameter vector as $\theta \in \mathcal{R}^d$:

Algorithm 6.4 Probabilistic data cleaning approach based on the compilation to Markov logic and performing the MAP inference.

1: **procedure** ProbDataCleaning($\mathcal{R}, IC$) ▷ $\mathcal{R}$ is a dataset; IC is a set of data cleaning rules;
2: $P \leftarrow []$
3: $H \leftarrow []$
4: $F \leftarrow []$
5: $MLF \leftarrow []$ ▷ Markov logic formulae
6: $A \leftarrow attr(\mathcal{R})$ ▷ Set of attributes in $\mathcal{R}$
7: **for** $a \leftarrow A$ **do**
8: $P \leftarrow P +$ getObservedPredicate(a)
9: **end for**
10: **for** $i \leftarrow IC$ **do**
11: $H \leftarrow H +$ getHiddenPredicate(i)
12: **end for**
13: **for** $f \leftarrow IC$ **do**
14: **if** isCFD(f) **then** ▷ Evaluates f constraints for being an CFD
15: $MLF \leftarrow MLF +$ TranslateCFDsToMarkovLogic(f)
16: **end if**
17: **if** isCMD(f) **then** ▷ Evaluates f constraints for being an CMD
18: $MLF \leftarrow MLF +$ TranslateCMDsToMarkovLogic(f)
19: **end if**
20: **if** isCIND(f) **then** ▷ Evaluates f constraints for being an CIND
21: $MLF \leftarrow MLF +$ TranslateCINDsToMarkovLogic(f)
22: **end if**
23: **end for**
24: $MLN \leftarrow P \cup H \cup MLF$ ▷ Formulates an MLN for data cleaning
25: $E \leftarrow$ getEvidenceAtoms($\mathcal{R}$) ▷ Transforms the $\mathcal{R}$ dataset into the evidence grounded atoms form
26: $P(H|E) \leftarrow$ runMAPInference(MLN, E) ▷ Performs MAP inference on MLN given grounded evidence predicate atoms E
27: **return** H ▷ Candidate repair: grounded hidden predicates with maximal probability
28: **end procedure**

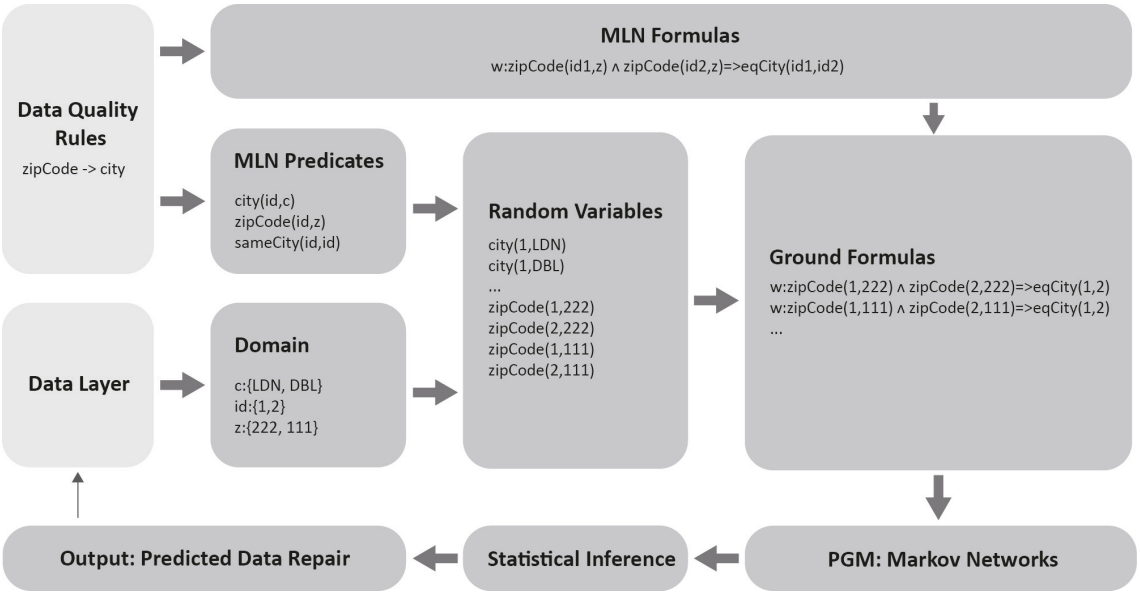

Figure 6.4: Data Cleaning Workflow In the context of a data cleaning workflow, the Markov Logic Network grounding process consists of two phases: I) MLN definition by (a) fixing MLN schema by defining observed and hidden predicates (b) domain, which is created from the existing data by considering the MLN schema, and (c) specification of weighted first-order logic formulae that represent data cleaning rules; II) MLN instantiation by assigning truth values to all possible instantiations of the MLN predicates by consideration of the domain (Random Variables) and using these ground atoms in the formulae. These ground formulae constitute a Markov Network to compute the MAP inference and to estimate the most likely data repairs. Source [234].

$$P\left(h|o,\theta\right) = \frac{1}{Z(o,\theta)} \exp\left(\langle\theta, \phi(h,o)\rangle\right), \; where \; Z(o,\theta) = \sum_{h\in\mathcal{H}^n, o\in\mathcal{O}^n} \exp\left(\langle\theta, \phi(h,o)\rangle\right),$$

where $\langle\theta, \phi(h,o)\rangle$ denotes a dot product. $Z(o,\theta)$ is the normalization constant to obtain valid probabilities. Since the partition function is a constant and the exponential is monotonic, finding the MAP assignment in our data cleaning problem is equivalent to finding all of the assignments $\phi(h_i, o_i)$ that maximizes the probability $P\left(h|o,\theta\right)$. Probabilistic models, such as Markov logic networks [73], can be reduced to a weighted first-order model counting representation, meaning to compute the sum of the weights of all satisfying assignments of the formulae in MLNs [230, 95].

A data repair operation is the probabilistic output of the inference process, which is described above. For instance, the inferred hidden predicate *equal-street* from the Example 6.0.1 may have the following possible value: *equal-street*$(1, 3)$. It points out that the *street* attribute for transaction 3 should have the same value as the *street* attribute of transaction 1. Consequently, the data repair operation should therefore replace the NULL value in transaction 3 with the address "1 Sun Dr.".

Generally, the inference computes the most likely state of the entire Markov logic network with regard to all integrity constraints. It finds the most likely data error (an irregularity according to the data quality dimensions violation) and its possible repair given the evidence (noisy data, auxiliary master data, etc.) By running the inference over the entire database, we predict the most likely data repairs for our dataset by determining the most likely grounding of the hidden predicates (Algorithm 6.4 code line 26). We utilize the *Cutting Plane Inference (CPI)* method that performs exact MAP inference and is guaranteed to converge in a finite number of steps [205]. The MPE thus provides a principled probabilistic framework for data cleaning by reducing the MPR to the most probable state of the MLN model.

Complexity of the approach. The inference of the most probable repair in Markov logic is either in PTIME [101] or #P complete [73]. A sharp dichotomy theorem [64] states that the probability of any hidden grounded atoms (or conjunctive queries) can either be computed in PTIME in the size of the database or is #P.

The biggest challenge for MAP inference arises when developing an efficient method to reason about the large number of groundings. To address this challenge, the MAP inference is casted to an *integer linear program* (ILP)[220]. Another alternative algorithm called *message passing* performs *belief propagation* along the edges of the graphical model. Despite being straightforward to implement, message passing might not converge [211, 90], and might return more inadequate results than ILP [179].

6.4 Markov Logic-Based Data Cleaning on Non-Relational Data

Our approach should work on non-relational data, which is often encountered in data analysis over unstructured data and, in particular, in information discovery methods that do not pre-specify a fixed schema of information [125]. To understand the problem of data cleaning for non-relational data, we start with the following example.

In Figure 6.5, we illustrate two issues identified in knowledge base induction, namely the problems of incomplete information and the existence of duplicates. We represent the knowledge base as a matrix. The rows of the matrix represent *facts* whose variable X can be replaced with a value from its columns. If, after replacing X with a value, the fact exists in the knowledge base, the matrix cell contains a 1. Otherwise, the cell value is 0. For example, the fact `"student may acquire degree"` does not exist in the knowledge base while `"pupil may acquire degree"` exists. The terms `pupil` and `student` are synonyms and therefore, the first fact can also be true. At the same time, according to the WORDNET

	X : degree	X : knowledge	X : answer
student may acquire X	0	1	0
pupil may acquire X	1	0	0
someone may **summarize** X	0	0	1
someone may **summarise** X	0	0	0

Figure 6.5: An example of an incomplete knowledge base represented as a matrix. The rows of the matrix represent facts whose variable X can be replaced with a value from its columns. If, after replacing X with a value, the fact exists in the knowledge base, the matrix cell contains the value 1. Otherwise the cell value is 0.

lexicon database [88], `summarize` and `summarise` are two possible spellings of the same word. Therefore, the *answer* can be a possible value for both facts that contain these verbs.

We consider *missing facts* in a knowledge base similar to what *missing values* are for a relational database. The difference, and the main challenge at the same time is that databases have a fixed set of attributes in the schema, while we consider textual data modeled according to the *Universal Schema* [247]. This schema unifies all extracted relations (from all sources), which means that an arbitrary number of distinct facts may be present in the database and the schema is *dynamic*. We adapt all data quality rules defined previously to the *Universal Schema* by considering meta-information about facts (e.g., similarity, synonymy) instead of the semantics of the relations itself. This means, we do not examine the meaning of the relations `"may summarize"`, `"may acquire"`, etc. Instead, we model whether two entities are synonymous and share the same relation, in which case two relations are similar. For example, we consider the relations `"may summarize"` and `"may summarise"` to be similar.

6.4.1 Data Cleaning Rules for Non-Relational Data

We now develop a solution to represent data cleaning rules for non-relational datasets. The main challenge here is that we are operating on schema-less data and, therefore, should create the appropriate dependency language to declare the data quality rules. We will describe how to define an MLN and how such rules are formalized to express *missing facts imputation*, using first-order logic. Assuming an existing common-sense knowledge base, our goal is to predict correct facts in the triple form (s, R, o), where s and o are nouns and represent *subject* and *object* respectively. The relation between them is R. This triple structure is also known as *Resource Description Framework (RDF)* data model [111]. Additionally, we assume that $S(n)$ is the set of synonyms of a particular noun n. We denote two nouns n_1 and n_2 as synonymous if $n_2 \in S(n_1)$.

Consider two triples (s_1, R, o_1) and (s_2, R, o_2) from the knowledge base. For each pair (o_1, o_2) of synonymous nouns, we define a rule, which extends the conditional functional dependency [6]. Similarly to the definition in Section 6.2, we define a synonymy predicate

as $\approx$, denoting the pair (o_1, o_2) as similar nouns. Then, in first-order logic we write:

$$\forall \mathsf{R}, \mathsf{o}_1, \mathsf{s}_1, \mathsf{o}_2, \mathsf{s}_2((\mathsf{s}_1, \mathsf{R}, \mathsf{o}_1) \wedge (\mathsf{s}_2, \mathsf{R}, \mathsf{o}_2) \wedge \mathsf{o}_1 \neq \mathsf{o}_2 \wedge \mathsf{o}_1 \approx \mathsf{o}_2 \Rightarrow (\mathsf{s}_1, \mathsf{R}, \mathsf{o}_2))$$
$$\forall \mathsf{R}, \mathsf{o}_1, \mathsf{s}_1, \mathsf{o}_2, \mathsf{s}_2((\mathsf{s}_1, \mathsf{R}, \mathsf{o}_1) \wedge (\mathsf{s}_2, \mathsf{R}, \mathsf{o}_2) \wedge \mathsf{o}_1 \neq \mathsf{o}_2 \wedge \mathsf{o}_2 \approx \mathsf{o}_1 \Rightarrow (\mathsf{s}_2, \mathsf{R}, \mathsf{o}_1)),$$

where $\mathsf{o}_1 \neq \mathsf{o}_2 \wedge \mathsf{o}_1 \approx \mathsf{o}_2$ denotes *not equal but synonyms*. In other words, for every two synonyms on the left-hand side of the triples (s_1, R, o_1) and (s_2, R, o_2), if they share the same relation and the entities on the right-hand side are not identical, then the following triple can be a potential fact (s_1, R, o_2). Taking into account that the synonymy is a symmetric relation, the analogous rule is true for the potential relation (s_2, R, o_1). In the following, we will walk through all the phases of the MLN creation for inducing potential relationships in the incomplete knowledge base, by considering the fact $(Person,\ may\ be,\ Smoker)$:

Observed predicates definition. Initially, we consider triples of the form (s, R, o), where s and o are nouns and represent subject and object respectively. R is the relationship between subject and object. We also model statements (expressions where only one part, subject or object, is present) and facts. To model similarity, we create a *synonyms* predicate between two concepts (nouns). The similarity calculation is performed by using an external lexicon, such as WordNet [88]. The *missing facts* imputation requires the definition of the MLN predicates. In the following, we specify the observed predicates:

1. subject(concept) - denotes that the variable *concept* is a subject of the triple (s, R, o).
2. object(concept) - denotes that variable *concept* is an object of the triple (s, R, o).
3. inRelation(concept, relation) - denotes that the *concept* belongs to the relation R in a given triple (s, R, o).
4. hasFact(concept, relation, concept, fact) - encodes the mapping between the tuple $(concept, relation, concept)$ and the *fact*-expression. For example, the fact *"person may be smoker)"* corresponds to the tuple *(Person, person may be X, Smoker)*. Where *Person* and *Smoker* are concepts representing object and subject respectively. X is a placeholder in relations for either object of subject.
5. synonyms(concept, concept) - whenever two concepts are synonyms, they are encoded as the arguments in the *synonyms*-predicate. For example, two concepts, such as *Student* and *Pupil* are synonyms. Therefore they are encoded as the observed predicate *synonyms*.

Ground atoms creation. We consider the fact $(Person,\ may\ be,\ Smoker)$ as an example of the grounded observed predicates (discussed above), which are generated from this fact. The domain of the variable concept includes the following

constants {Person, Smoker, Individual}. The domain of the variable relation contains {person may be X}. By using these domains, the ground predicates for the fact (*Person*, *may be*, *Smoker*) are specified as follows:

1. subject(Person)
2. object(Smoker)
3. inRelation(Person, person may be X)
4. inRelation(Smoker, person may be X)
5. synonyms(Individual, Person)
6. hasFact(Person, person may be X, Smoker, person may be smoker)

Please note that two concepts subject(Person) and object(Smoker) are sharing the same relation *person may be X*. Therefore we have created two grounded atoms, such as inRelation(Person, person may be X) and inRelation(Smoker, person may be X). According to the WordNet lexicon, one possible synonym for the *Person* concept is the *Individual* concept. Therefore, the observed predicate synonyms(concept, concept) is grounded as synonyms(Individual, Person). We include all synonyms of the provided concepts for either subject or object in all observed tuples.

Hidden Predicates Specification. Hidden predicates (also, query predicates) are predicates that we reason about. To infer whether a concept belongs to some relation, we specify the following hidden predicate: potentialRelation(concept, relation), which indicates that the defined *concept* potentially belongs to the provided *relation*. We also define a set of auxiliary hidden predicates to compute the final predicate potentialRelation(concept, relation), namely, allowedSubjConcepts($\mathsf{concept}_1$, $\mathsf{concept}_2$) to specify similar concepts, which are subjects in triple (s, R, o). The predicate allowedSubjRelation(concept, relation) is specified to reason when the concept is connected to the relation. The auxiliary hidden predicates will not be evaluated because they are used to reason about the query predicate potentialRelation(concept, relation).

Markov Logic Program Definition. Missing value imputation for non-relational data is presented below by the Markov logic program snippet below. The symbols w_i denote real-numbered weights assigned to each Markov logic formula.

1. w_1 $\mathsf{subject}(\mathsf{c}_1) \wedge \mathsf{subject}(\mathsf{c}_2) \wedge \mathsf{synonyms}(\mathsf{c}_1, \mathsf{c}_2) \Rightarrow \mathsf{allowedSubjConcepts}(\mathsf{c}_1, \mathsf{c}_2)$
2. w_2 $\mathsf{allowedSubjConcepts}(\mathsf{c}_1, \mathsf{c}_2) \wedge \mathsf{inRelation}(\mathsf{c}_1, \mathsf{r}_1) \Rightarrow \mathsf{allowedSubjRelation}(\mathsf{c}_1, \mathsf{r}_1)$

3. w_3 $\mathsf{allowedSubjConcepts}(c_1, c_2) \wedge \mathsf{allowedSubjRelation}(c_1, r_1) \wedge \mathsf{allowedSubjRelation}(c_2, r_2)$ $\Rightarrow \mathsf{potentialRelation}(c_1, r_2)$

The code snippet above considers the inference of the potential relationship (fact) between an concept and relation, which is not present in the knowledge base. The concepts are represented as predicates $subject(c_1)$ and $subject(c_1)$. The relations are encoded as r_1 the r_2 variables in all Markov logic predicates. Both the first and second formulas are intended to infer the intermediate hidden predicates $allowedSubjConcepts(c_1, c_2)$ and $allowedSubjRelation(c_1, r_1)$. These predicates are used to infer the final hidden predicate $potentialRelation(c_1, r_2)$, which denotes that the concept c_1 potentially relates to the relation r_2 as either (c_1, r_2, c_a) or (c_a, r_2, c_1) (c_a denotes any concepts). The potential object of the above relation r_2 is inferred in the similar way as above.

Inference result We derive missing relations (facts) in the KB by using the MAP inference. For instance, one of the potential relations is expressed in the inferred hidden predicates as follows: $\mathsf{potentialRelation}(\mathsf{Individual}, \mathsf{X\ may\ be\ smoker})$, which represents a triple (*Individual*, *may be*, *Smoker*).

In this section, we extended our approach to work on non-relational data and developed a compilation workflow for data cleaning rules into the Markov logic formalism. We primarily focus on the *information completeness* issue, which deals with missing values imputation in existing knowledge bases. In the following we present experimental results for cleaning relational and non-relational data.

6.5 Experiments

In this section, we provide results of probabilistic data cleaning experiments, which were performed on relational and non-relational data. We first, conduct experiments on capturing the data issue interaction in Sections 6.5.2 and 6.5.4. Next, we provide experiments on the effect of a different order of data cleaning rules execution in Section 6.5.3. We study how Markov logic capabilities cover the requirements of data cleaning systems in Section 6.5.5. Finally, we conduct experiments on Markov logic for data cleaning on non-relational data in Section 6.5.6.

The code base for the experiments that are presented in this chapter is available online[1]. Next, we describe our experimental settings, such as datasets and the evaluation metric.

6.5.1 Experimental Setup

The experiments are conducted on the following four real-life and synthetic datasets.

[1]The Markov logic programs are provided online: `http://bit.ly/mln-for-dq`

HOSP. The HOSP dataset has been published by the US Department of Health & Human Services[2]. This dataset comprises 9 attributes: *addr*, *city*, *cond*, *country*, *hospname*, *measure*, *phone*, *state*, *zip*. We use 6 CDFs and one MD, which have been manually designed by Dallachiesa M. et al. [62]. They specified an MD that makes use of master data, namely US ZIP codes: ZIPCode[3]. The dataset contains $43k$ tuples with two fields: *zip* and *state*. The above MD states that if two tuples from *hosp* and ZIPCode possess the same *zip* code values, and the *state* values are distinct, then the *state* value from the ZIPCode table should be adopted.

TPC-H. To conduct experiments on synthetic data, we use the TPC-H[4] dataset. For our purposes, we selected two tables - *Customer* and *Orders* - which we joined to introduce duplications on the *Customer* relations data. The final dataset contains 17 attributes, such as *c_custkey*, *c_name*, *c_address*, *c_nationkey*, *c_phone*, *c_acctbal*, *c_mtksegment*, *c_comment*, *o_orderkey*, *o_custkey*, *o_orderstatus*, *o_totalprice*, *o_orderdate*, *o_orderpriority*, *o_clerk*, *o_shippriority*, *o_comment*.

HOSP and TPC-H: Ground truth and dirty data. We randomly injected errors into the relational datasets HOSP and TPC-H to produce dirty data. We introduced several kinds of errors, such as missing values, domain violation, and illegal values (accuracy violation). The initial data is considered clean and acts as ground truth. Furthermore, we ensured that the ground truth is consistent with respect to the given CFDs and MDs. We performed error injection with different rates ranging from 2% to 10%. Moreover, we introduced errors into different dataset sizes ranging from one thousand to one hundred thousand data points. The error rate depicts the ratio of the number of erroneous values to the total number of values in the dataset. Data errors are injected only into attributes that are included in the data cleaning rules.

To conduct experiments on cleaning non-relational data, we have chosen two datasets: MICROSOFT ACADEMIC GRAPH (MSAG) [214] and WELTMODELL [9].

MSAG. Due to the integration of data from different sources, web data is suffering from various data quality issues [153]. Thus we include another dataset - MICROSOFT ACADEMIC GRAPH (MSAG)[5] [214] - to evaluate our approach on the web data cleaning. MSAG is a heterogeneous entity graph comprised of six types of entities that models real-life academic relationships, such as field of study, author, institution, paper, venue, and conference instances. The raw data has been obtained from different deep web sources, such as academic publishers and web-pages indexed by the Bing search engine. This dataset is

[2] http://www.medicare.gov/hospitalcompare/Data/Data-Download.html
[3] http://databases.about.com/od/access/a/zipcodedatabase.htm
[4] http://www.tpc.org/tpch/
[5] http://research.microsoft.com/en-us/projects/mag/

Paper	*Author*	*Organization*
paper_id	*author_id*	*affiliation_id*
publish_year		*origin_name*
		normalized_name

Table 6.6: Entities ***Paper***, ***Author*** and ***Organization*** and their attributes that have been used in experiments for Web data cleaning with Markov logic. Source [234].

provided in the form of a connected entities graph. For our experiments, we chose three entities from the whole graph, namely *author*, *organization*, and *paper*. This sub-graph of MSAG demonstrates typical errors of the extracted web data, such as missing and inconsistent values. For instance, we found out that there are inconsistencies in organization names for the same author. A number of author entries have a missing affiliation. Table 6.6 shows the attributes of the selected entities in MSAG, which we use to model and run data cleaning on this dataset.

WELTMODELL. This dataset is a knowledge base of common sense statements and related concepts, automatically derived from the text of 3.5 million Google Books [98]. The knowledge base contains over 6 million distinct statements applied to 3 million distinct concepts. Both its schema, in the form of statements, as well as its content, in the form of concepts to which statements may apply, are induced and therefore may be erroneous or incomplete. Besides count statistics for every concept-statement pair, the knowledge base provides similarity measures that indicate the similarity of two concepts, as well as the similarity of two statements.

Please note that the integrity constraints for all datasets are specified in Section 6.5.5.

Technical Details. All experiments were performed on a Linux machine with 256 GB RAM and 2 CPUs with 12 Cores each 2.3 GHz. For Markov logic modelling and performing statistical inference we use the inference engine for Markov logic *RockIt*, developed by Noessner J. et al. [179].

6.5.2 Holistic Data Cleaning: Uniqueness and Accuracy Data Quality Dimensions

Following the idea that capturing the data issue interaction increases the overall accuracy in data cleaning [82], we created an experiment to show the connection between uniqueness and data accuracy interaction. The evaluation is performed on different noise rates and different dataset sizes. Figure 6.6 illustrates the results of using Markov logic programs for data cleaning on the HOSP dataset, while plots for the results for the TPC-H dataset are shown in Figure 6.7.

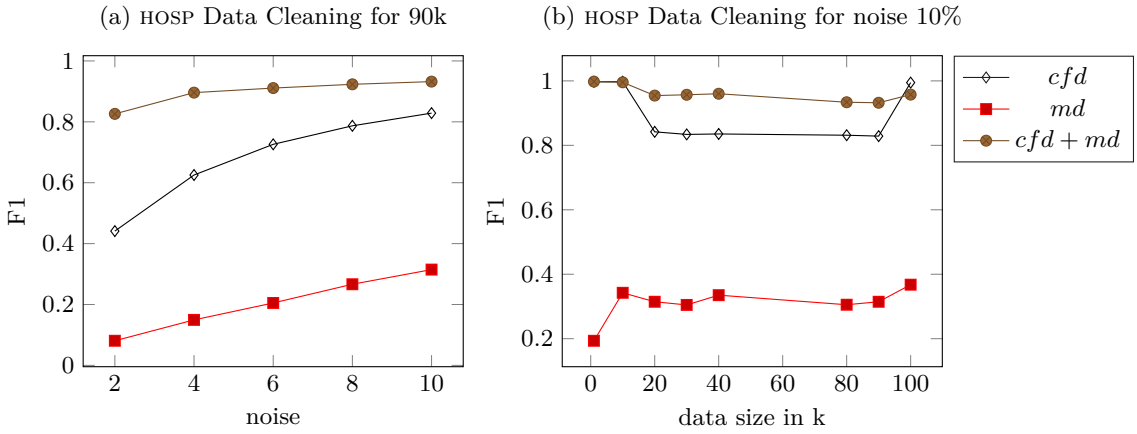

Figure 6.6: Evaluation of the data repair method based on Markov logic applied on the HOSP dataset. Source [234].

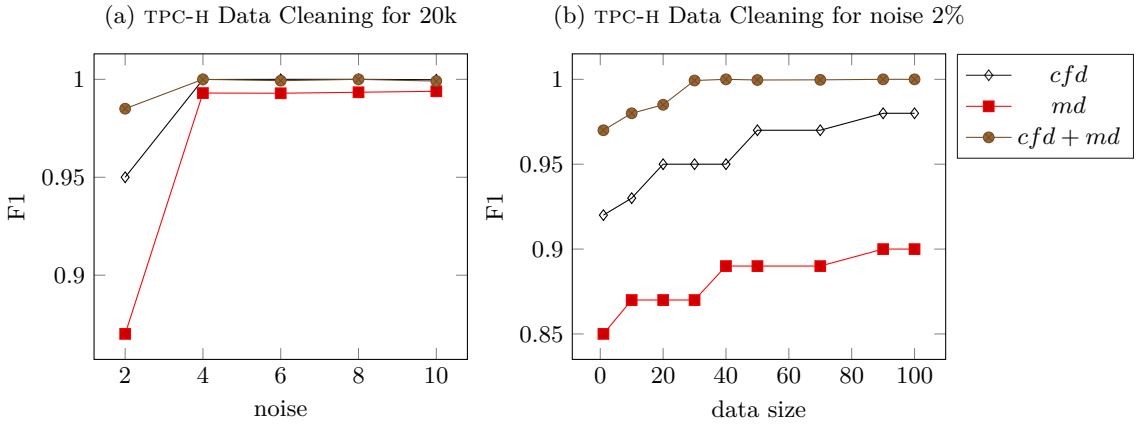

Figure 6.7: Evaluation of the data repair method based on Markov logic applied on the TPC-H dataset. Source [234].

We run several experiments, and in the first series, we fix the size for the HOSP and TPC-H datasets to $90k$ respectively $20k$ tuples, while varying the error rate from 2% to 10%. First, we compare the results of the execution of CDFs and MDs, which were performed in parallel and independently of each other.

This experiment includes the execution of the combined CFDs and MDs. The provided results in Figures 6.6 and 6.7 clearly demonstrate that jointly modelling CFDs and MDs improves the overall result because joint execution involves two processes simultaneously: matching (for duplicates) and erroneous values detection. When CFDs and MDs are specified together in a single Markov logic program for identifying dirty data, hidden predicates are inferred simultaneously, and hence, our method automatically picks the order of the data quality rules execution. We observe the low F_1 score of MDs in Figures 6.6 and 6.7, which results from low precision and high recall. However, by extending the data cleaning routine with CFDs, the overall F_1 score improves. The upward trend of

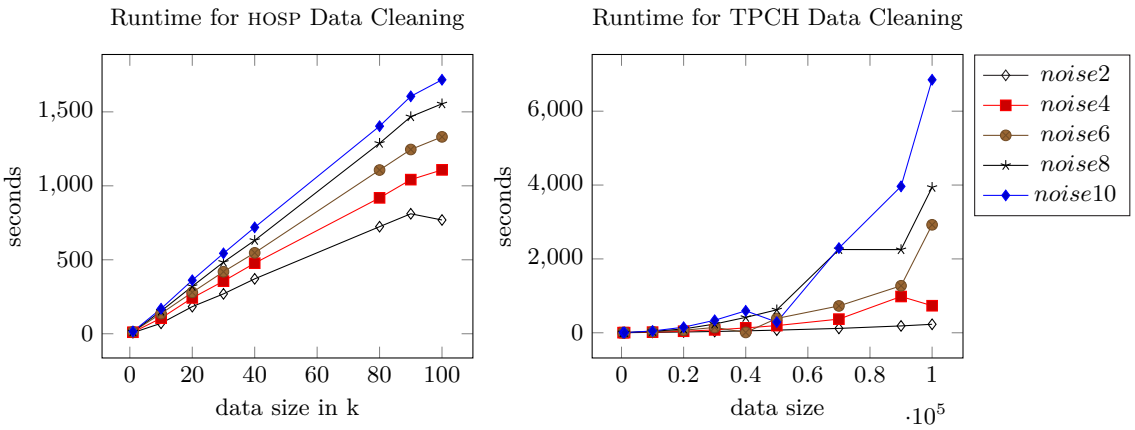

Figure 6.8: Runtime for Markov logic based data cleaning applied on the HOSP and TPC-H datasets. Source [234].

the F_1 values while increasing error rate in the datasets in Figure 6.6 (a) and Figure 6.7 (a) is explained by the algorithmic behaviour of the *cutting plane inference* (CPI) [205]. We employ the CPI algorithm, which leverages optimization techniques such as integer linear programming (ILP) [205, 179]. ILP maximizes the objective function under a set of constraints, such as all formulae in MLN, which are converted into an ILP constraint. For datasets with a low percentage of errors, the algorithm requires only a few iterations to converge. Technically, it means that the ILP approximately determines the maximum objective score. Looking at the runtime in Figure 6.8, we observed that, with increasing noise, the underlying solver determines an objective score until the maximum number of iterations is reached. Therefore, more data errors are identified.

We performed the next series of experiments with varying size of data and a fixed error rate to 10% on HOSP and to 2% on the synthetic TPC-H (see Figure 6.6 (b) and Figure 6.7 (b)). We ran three variations of data cleaning rules on all dataset sizes ranging from one thousand to one hundred thousand tuples. The results show that our data cleaning method delivers robust results regardless of the data size despite the increasing runtime of the algorithm (see Figure 6.8) by increasing the size of the dataset. The reason for this behaviour is that the convergence of CPI is guaranteed [204]. Moreover, Figure 6.8 shows detailed runtime values for the HOSP and TPC-H datasets. While we increase the data size, we observed that the average runtime growth rate is 1.9, which demonstrates a characteristic of the underlying MAP inference algorithm [205]: fewer errors in the dataset lets the algorithm converge faster. Therefore the runtime values are lower for data with fewer errors.

To summarize, the modelling of data cleaning problems with Markov logic improves the accuracy compared to single data quality rules. While using joint inference, its runtime

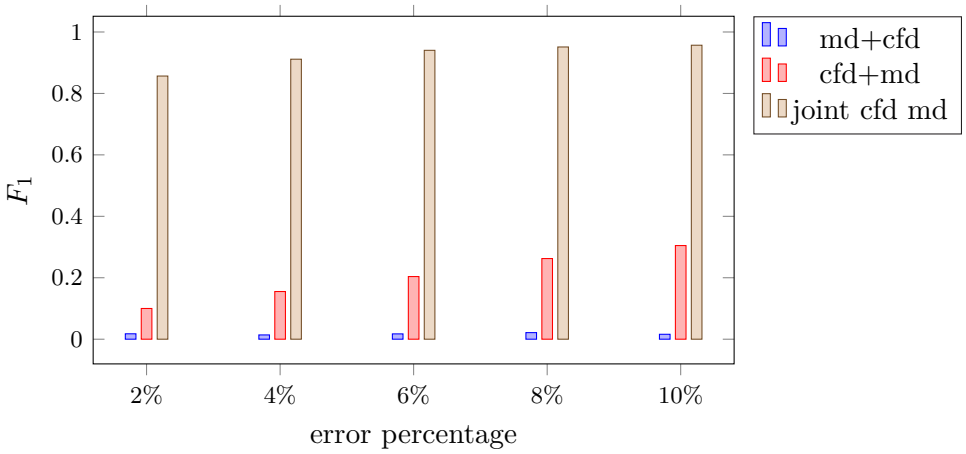

Figure 6.9: The evaluation of the different experimental settings of the execution order of data cleaning rules translated into Markov logic. The experiments are performed on the real-world Hosp dataset. Source [234].

performance will depend on the underlying inference engine: the larger the dataset and the more noise it contains, the longer it takes to clean it using joint inference.

6.5.3 Impact of Rule Execution Order

In the next line of experiments we intend to demonstrate the effects of different orders of data cleaning rule execution. We will show that it is difficult to achieve the optimal order of rules manually [62]. We leverage probabilistic inference for simultaneous rule execution instead of a manual specification of the order in which the data cleaning rules should be executed. Our results are shown in Figure 6.9. We created the Markov logic program on the attributes *state* and *zip* of the real-world Hosp dataset because they both are included in CFD and MD data cleaning rules. To conduct the experiment, we use the dataset with $90k$ tuples, and errors are varying from 2% to 10%. The methodology for this experiment is the following:

1. Execute MD rules and then execute CFD rules;
2. Execute CFD rules and then execute MD rules;
3. Execute MD and CFD rules jointly;

The first results confirm the previous experiment about the joint modelling data cleaning rules (see Figures 6.6 and 6.7). The MD rules showed poor performance. The model in the first case achieved the overall F_1 score between 0.01 and 0.02. We explain such result by low precision and recall values due to error propagation after the MD-rules execution to CFDs execution. By changing the sequence of execution to running CFD rules before MD rules in the second scenario, we observe a slight improvement in the F_1 scores. The second

Method	Pruning Threshold	Precision	Recall	F-1
HoloClean with *Metadata-Driven Error Detection* approach from [232]	0.0	0.9164	0.4154	0.5716
	0.1	0.9161	0.4161	0.5722
	0.2	0.9174	0.4140	0.5706
	0.3	0.9231	0.4067	0.5646
	0.4	0.9295	0.4033	0.5626
	0.5	0.9297	0.4040	0.5633
	0.6	0.9367	0.4147	0.5749
	0.7	0.9384	0.4241	0.5842
	0.8	0.9383	0.4241	0.5841
	0.9	0.9383	0.4241	0.5841
	1.0	0.9383	0.4241	0.5841
HoloClean with ideal error detection	-	0.8431	0.8294	0.8362
SRL based approach [234]	-	0.7523	0.916	0.8361

Table 6.7: Comparison to the HoloClean system [202]. The experiments have been conducted on the Hosp dataset. The pruning threshold is required by the HoloClean algorithm and ranges from 0.0 to 1.0. The error detection method is *Metadata-Driven Error Detection* from Visengeriyeva et al. [232]. Additionally, we performed data repair on HoloClean with *ideal* error detection by comparing the dirty data with its ground truth.

model achieves 0.1 to 0.3 F_1 scores for different noise values. We explain this with the fact that CFDs detect more violations and less errors are propagated to the MD-rules execution. Nevertheless, as in the previous part, error propagation leads to a low F_1 score. In the last part of this experiment, we perform the joint execution of CFD and MD rules, where we model duplicate detection and accuracy violation detection. Compared to the sequential execution of data cleaning rules, we observe an increase of all F_1 scores from 0.86 for 2% of errors to 0.96 for 10% of errors. The results described above confirm our hypothesis that the joint execution of multiple data cleaning rules by running the probabilistic inference eliminates the need to provide the order for these rules.

Comparison to state-of-the-art. To make a comparison to the state-of-the-art data cleaning systems, we also examine our approach against the HoloClean [202][6] and Nadeef [62] systems.

The HoloClean system provides holistic data repairing by using probabilistic inference, which is similar to our method. The experiments have been conducted on the Hosp dataset with a fixed size, $90k$ tuples and 10% of errors. We run different configurations of the pruning threshold, which is required by the HoloClean algorithm as an optimization technique. To see the effect of this optimization, the pruning score ranges from 0.0 to 1.0. Since HoloClean is designed for data repair and considers the error detection step as a black box, we adopted the metadata-driven error detection method [232], which we described in Chapter 5. Our error detection method based on stacking has achieved

[6]Our re-implementation of the HoloClean data repair approach uses the DeepDive language and inference engine [69]. The re-implementation is available online: `http://bit.ly/ed-holoclean-hosp`

System	2%	4%	6%	8%	10%
Baseline NADEEF system [62]	0.83	0.85	0.91	0.95	0.95
SRL-based approach [234]	0.86	0.89	0.90	0.91	0.96

Table 6.8: F_1 measure comparison of the jointly modeled data cleaning rules based on CFD and MD to the baseline system [62]. The experiments conducted on the HOSP dataset on size $90k$ and different error percentages ranging from 2% to 10%. Source [234].

Functionality	NADEEF [62]	HOLOCLEAN [202]	SRL Method [234]
Error detection	+	-	+
Error repair	+	+	+
Data cleaning rules	FD MD UDF	Denial constraints	FD MD IND Relaxed FD
Reasoning	SAT Solver	MAP inference (SQL impl.)	MAP inference (ILP impl.)
Representation (declaring rules)	Domain specific language	DDlog rules/queries [69]	First-order logic
Optimization	Partitioning Compression	Domain pruning Tuple partitioning Rules relaxation	Tuple partitioning

Table 6.9: Qualitative comparison of the SRL-based data cleaning to the state-of-the-art data cleaning systems, such as NADEEF [62] and HOLOCLEAN [202].

the following results: a precision score of 0.46, a recall score of 0.68, and an F_1 score as 0.55. Additionally, we performed data repair on HOLOCLEAN with *ideal* error detection by comparing the dirty data with its ground truth. The results are provided in Table 6.7. Since the HOLOCLEAN's recall is limited by the error detection method, we achieved different recall scores for error repair on the datasets with different error detection results, such as comparison to the ground truth and the stacking-based error detection [232]. Our Markov logic-based data cleaning approach achieves similar precision scores; however, by producing a higher recall score, we are able to outperform the HOLOCLEAN approach.

The NADEEF system also treats the integrity constraints holistically and achieves an average F_1-measure of 0.89 on the HOSP dataset, as shown in Table 6.8. Although their system demonstrates better performance than ours in regard to MD rules, we achieve better F_1-score values for the joint execution of the duplicate detection and accuracy violation detection rules: we demonstrate an average performance of F_1 0.90 on HOSP.

We also performed the qualitative evaluation of all three systems by comparing the functionality of our SRL-based approach against the HOLOCLEAN and NADEEF systems, as shown in Table 6.9. In contrast to these systems, our method relies on the ILP-based MAP inference [179]. While using the Markov logic formalism, we model our data cleaning rules in the ubiquitous first-order logic language. Similarly to NADEEF, our method performs error detection and repair in a joint manner, whereas HOLOCLEAN treats error detection as a black box.

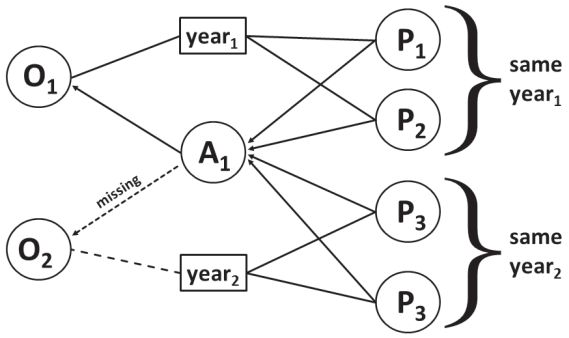

Figure 6.10: The part of the MSAG dataset with missing organization values. Markov logic captures the following evidence: if two papers of the same author have been published in the same year, then they may be published by the author of the same organization. Nodes notation: A - denotes *Author* entity; O - *Organization* and P - *Paper*. Missing edges are marked as dashed lines. Source [234].

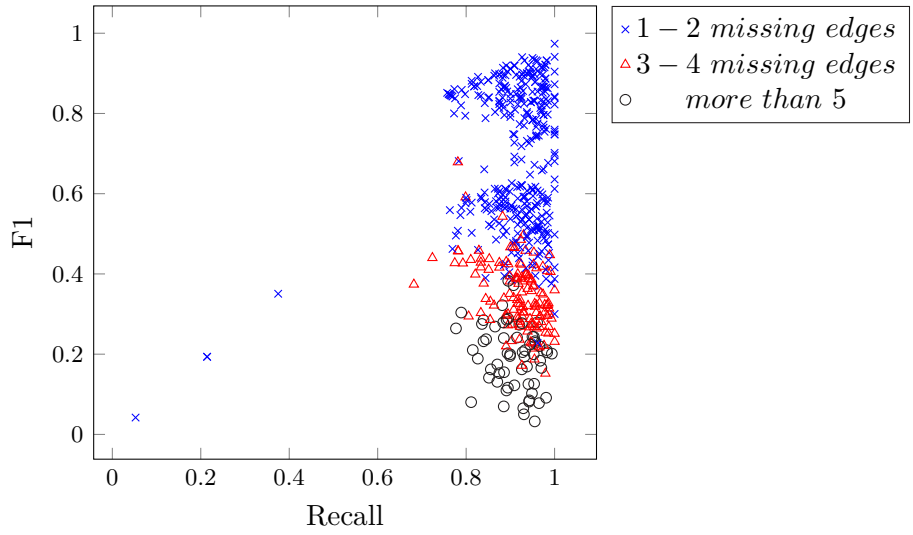

Figure 6.11: The accuracy of data cleaning on MSAG that depends on the amount of missing edges. Source [234].

6.5.4 Holistic Data Cleaning: Missing Value and Consistency Issues Interaction

The next series of experiments are conducted on web data, and we provide initial results on cleaning the highly imperfect graph-structured MSAG dataset. We study the connection between *completeness* and *data consistency* by providing the Markov logic data cleaning solution on web data. The *completeness* and *consistency* dimensions interact with each other in the following manner: missing values imputation helps to fix inconsistencies, and

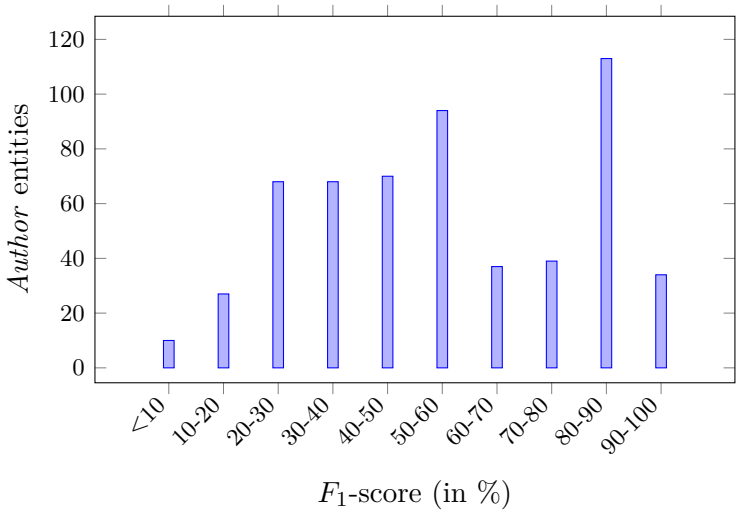

Figure 6.12: The distribution of the corrected *Author*-entities. Source [234].

by correcting values, we might identify further missing entities. We run this experiment on the randomly selected 600 *Author*-entities, for each *Author*-entity with at least 10 *Paper*-*Author* edges. The example of the MSAG graph is shown in Figure 6.10. The performance of our method on the MSAG dataset is provided in Figure 6.11. Depending on the number of missing edges, we distinguish between three kinds of *Author*-nodes (marked separately): nodes with one to two missing *Author*-*Organization* edges, nodes with three to four missing values for the *Author*-*Organization* connection, and nodes with more than 5 missing edges.

Furthermore, Figure 6.12 shows another perspective on this experiment by demonstrating the exact distribution of corrected *Author*-entities (x-axis) by F_1-score (y-axis). Our approach achieves overall F_1 score higher than 50% and recall higher than 78% for *Author*-entities with maximal two missing edges. The precision scores drop when repairing entities with more than three missing edges. However, we still achieve a recall ranging from 0.8 to 1.0. The reason is that our approach is producing more false positives with an increasing number of missing values. This experiment shows that the SRL-based data cleaning method produces reasonable results on web data with a small number of errors (e.g., missing values).

6.5.5 Usability of Modelling Data Cleaning Rules With Markov Logic

In this section, we discuss the usability of modelling data cleaning rules by using Markov logic on three datasets such as HOSP, TPC-H, and MSAG. This part of our experimental study is designed to show how the expressiveness of Markov logic facilitates modelling the data cleaning rules. To examine the usability of our method, we follow the research methodology from UMUX-LITE [149] by postulating the UMUX-LITE questionnaire item:

	HOSP	TPC-H	MSAG
observed predicates	*providerNr*/HOSP(hid, pn) *hospitalName*/HOSP(hid, n) *address*/HOSP(hid, add) *city*/HOSP(hid, c) *state*/HOSP(hid, st) *zipCode*/HOSP(hid, code) *phoneNumber*/HOSP(hid, numb) *condition*/HOSP(hid, cond) *measureCode*/HOSP(hid, mcode) *score*/HOSP(hid, score) *zip*/ZIPCODE(zid, code) *state*/ZIPCODE(zid, st)	*custKey*(id, key) *name*(id, n) *addr*(id, add) *natKey*(id, nkey) *phone*(id, ph) *acc*(id, a) *mrkt*(id, m) *orderKey*(id, okey) *totalPrice*(id, p) *orderDate*(id, d) *orderPriority*(id, pr) *clerk*(id, c)	*publishYear*(paperid, pubyear) *author*(paperid, authorid) *affiliation*(paperid, affilid) *inRange*(pubyear, pubyear) *originAffiliationName*(affilid, oname) *normalAffiliationName*(affilid, nname)
hidden predicates	*equal-HospitalName*/HOSP(hid, hid) *equal-Address*/HOSP(hid, hid) *equal-City*/HOSP(hid, hid) *equal-State*/HOSP(hid, hid) *equal-ZipCode*/HOSP(hid, hid) *equal-PhoneNumber*/HOSP(hid, hid) *equal-Condition*/HOSP(hid, hid) HOSP/*match-State*/ZIPCODE(hid, zid) HOSP/*match-ZipCode*/ZIPCODE(hid, code)	*equal-Names*(id, id) *equal-Addr*(id, id) *equal-Natkey*(id, id) *equal-Phone*(id, id) *equal-Acc*(id, id) *equal-Mrkt*(id, id) *match-Phone*(id, id) *match-Addr*(id, id)	*equal-Affiliation*(paperid, paperid) *equal-OriginNames*(oname, oname) *equal-OriginNamesByPaperId*(paperid, paperid) *equal-NormalNames*(nname, nname) *equal-NormalNamesByPaperId*(paperid, paperid) *missingOriginName*(paperid, oname)

Table 6.10: Markov logic predicates used in data quality rules. Source [234].

"Markov logic capabilities meet the requirements of data cleaning systems". As already mentioned in Section 6.2, Markov logic meets the main requirements of data cleaning systems, such as *holistic* data quality rules treatment [87, 62], automation [222, 223], and heterogeneous rules incorporation [54].

Modelling HOSP Quality Rules. For the HOSP dataset, we use 6 manually-designed CDFs, which result in 15 normalized CFDs. One MD rule is also normalized into 2 formulae. Markov logic predicates used for data quality formulae are shown in Table 6.10. The final data cleaning model consists of 21 Markov logic formulae. In Table 6.11, we provide an excerpt of these data quality rules. The MD rule is specified on two relations, such as the HOSP dataset and master data. After transforming the 100k HOSP tuples into Markov logic grounded atoms, the resulting data comprises 1.3M evidence atoms, which are used for inference.

Modelling TPC-H Quality Rules. We formulated 9 CFDs and 3 MDs for this dataset. One example of the rules is a CFD which states that two tuples match on *c_custkey*, then they should match on the *c_name* and *c_address* attributes. MDs are specified on the same schema. These MDs state that if the LHS is similar for any pair of tuples (t_1, t_2), then the attribute values on the RHS should be identified. The example of the data quality rules is provided in Table 6.11. Furthermore, Table 6.10 shows the Markov logic predicates, which are used to formulate data quality rules. After transforming the 100k TPC-H tuples into Markov logic grounded atoms, the resulting data comprises 1M evidence atoms.

Dataset	Data Cleaning Rules	Markov Logic Formulae
HOSP	cfd_1 : HOSP([*zip*] → [*state, city*], t1 = (_ ‖ _, _))	w_1 : *zip*/HOSP(id1, code) ∧ *zip*/HOSP(id2, code) ∧ *state*/HOSP(id1, s1) ∧ *state*/HOSP(id2, s2) ∧ !*state*/HOSP(id1, s2) ∧ !*state*/HOSP(id2, s1) ⇒ *equal-state*/HOSP(id1, id2) w_2 : *zip*/HOSP(id1, code) ∧ *zip*/HOSP(id2, code) ∧ *city*/HOSP(id1, c1) ∧ *city*/HOSP(id2, c2) ∧ !*city*/HOSP(id1, c2) ∧ !*city*/HOSP(id2, c1) ⇒ *equal-city*/HOSP(id1, id2)
	md_1 : HOSP[*zip*] = ZIPCODE[*zip*] ∧ HOSP[*state*] ≠ ZIPCODE[*state*] → HOSP[*state*] ⇌ ZIPCODE[*state*]	w_3 : *zip*/HOSP(id1, code) ∧ *zip*/ZIPCODE(id2, code) ∧ *state*/HOSP(id1, s1) ∧ *state*/ZIPCODE(id2, s2) ⇒ HOSP/*match-State*/ZIPCODE(id1, id2)
TPC-H	cfd_1 : T([*c_custkey*] → [*c_name,c_address*], t1 = (_ ‖ _, _))	w_1 : *custKey*(id1, key) ∧ *custKey*(id2, key)∧ *name*(id1, n1) ∧ *name*(id2, n2)∧ !*name*(id1, n2)∧ !*name*(id2, n1) ⇒ *equal-Names*(id1, id2) w_2 : *custKey*(id1, key) ∧ *custKey*(id2, key)∧ *addr*(id1, addr1) ∧ *addr*(id2, addr2)∧ !*addr*(id1, addr2)∧ !*addr*(id2, addr1) ⇒ *equal-Names*(id1, id2)
	md_1 : T[*c_address*] = T[*c_address*] → T[*c_phone*] ⇌ T[*c_phone*]	w_3 : *addr*(id1, addr) ∧ *addr*(id2, addr)∧ *phone*(id1, phone1) ∧ *phone*(id2, phone2)∧ ⇒ *match-Phone*(id1, id2)
MSAG	cfd_1 : M([*author_id, year*] → [*affiliation_id*], t1 = (_, _ ‖ _))	w_1 : *author*(pid1, aid1) ∧ *author*(pid2, aid2)∧ *publishYear*(pid1, y) ∧ *publishYear*(pid2, y) ⇒ *equal-Affiliation*(pid1, pid2)
	eCfd_1 : M([*author_id, year*] → [*affiliation_id*], t1 = (_, diff(_) ≤ 2 ‖ _))	w_2 : *author*(pid1, aid1) ∧ *author*(pid2, aid2)∧ *publishYear*(pid1, y1) ∧ *publishYear*(pid2, y2)∧ *inRange*(y1, y2) ⇒ *equal-Affiliation*(pid1, pid2)
		symmetry : ∞ : *equal-Affiliation*(pid1, pid2) ⇒ *equal-Affiliation*(pid2, pid1)
		transitivity : ∞ : *equal-Affiliation*(pid1, pid2) ∧ *equal-Affiliation*(pid2, pid3) ⇒ *equal-Affiliation*(pid1, pid3)

Table 6.11: Modelling data cleaning rules as Markov logic programs (an excerpt). MLN's *soft* rules are specified with the weights w_i set to 1.0 and *hard* rules are marked with infinite weights: ∞. Source [234].

Modelling MSAG Quality Rules. For the MSAG dataset, we develop data quality rules based on *extended* CFDs [48], and equality axioms, such as *symmetry* and *transitivity* (see Figure 6.10). For the *Paper-Author-Organization* entities, we specify two CFDs, one extended CFD and 8 additional rules that express the equality axioms for hidden predicates. Considering the semantical meaning of the data, we use the ability to incorporate domain knowledge into the data cleaning process. For example, the CFD

M([*author_id, year*]→ [*affiliation_id*], t1 = (_, _ ‖ _))

captures missing affiliations by the same author, which were published in the same year. Practically, we assume that in academia, an average contract lasts around 2-3 years. Therefore, we will extend our search range by augmenting this knowledge in the form of a predicate *inRange*(year, year). Additionally, we spot more missing values by adding equality axioms. For example, the hard rule

equal-Affiliation(id_1, id_2) ∧ *equal-Affiliation* (id_2, id_3) ⇒ *equal-Affiliation*(id_1, id_3)

specifies a transitive relationship between three different *Organization* entities. Thus, our final Markov logic program contains 21 lines of code (an example is shown in Table 6.11).

Inferred fact	True	False
Person may stop Smoking	✓	
Individual may recall Person	✓	
Director may have Office	✓	
Corporation may have Office	✓	
Head may contain Idea	✓	
Member may be Brain		✓
Brain may be Description		✓
Challenge may be Organization		✓
Organization may be Device		✓
Article may give Office		✓

Table 6.12: The evaluation of inferred data for the non-relational dataset as a series of true/false questions over inferred common sense facts. The top 5 facts in this table were annotated as correct by human annotators, while the lower 5 were annotated as incorrect.

6.5.6 Experiments on Non-Relational Data.

To conduct experiments on cleaning non-relational data, we use the Weltmodell [9] dataset. In the experiment, we face two challenges: (1) the absence of the master data, and (2) Weltmodell misses factual information (attributes), and we still do not know whether it misses statements (Open world assumption).

Please note that the MLN formulation for the Weltmodell dataset is provided in Section 6.4.1. For this dataset, the inference by using our MLN model resulted in a total of $21k$ potential facts as missing values. We evaluated the imputation of missing data by conducting a user study. This is made possible through the fact that the knowledge base consists of common sense facts that are easily evaluated as a series of true/false questions by a set of evaluators. To evaluate these inferred facts, we randomly sampled 230 facts each from three clusters of facts that concern the concepts "Human", "Organization" and "Brain". These 690 facts were passed to 6 human annotators, and each of them was asked to decide whether an inferred common sense fact was true or not. Examples of this task are given in Table 6.12.

To measure to what extent evaluators agree on the evaluation task, we calculate the commonly-used *kappa*-coefficient [58], which gives a quantitative measure of the magnitude of agreement between annotators. In the following, we provide a calculation for *kappa*.

Let A_o be the observed agreement that is the number of judgments on which the annotators agree divided by the total number of relations. Furthermore, let A_e be the number reflecting the fact that annotators agree on any category. In our case, we refer to *"correct"* and *"incorrect"* as a set of labels $\{1, 0\}$ respectively. Then

$$A_e = \sum_{k \in \{1,0\}} P(user_A|k) \cdot P(user_B|k)$$

Settings	Organiz.	AoC	Brain	AoC	Human	AoC
User 1 and 2	0.8438	66.07%	0.7456	55.26%	0.7512	65.07%
User 1 and 3	0.8348	64.29%	0.8158	51.32%	0.7512	57.89%
User 1 and 4	0.7991	61.61%	0.6535	32.89%	0.7033	52.15%
User 1 and 5	0.8571	62.95%	0.6667	43.42%	0.7943	59.33%
User 1 and 6	0.7813	62.95%	0.6579	42.54%	0.7321	54.07%
User 2 and 3	0.7946	61.61%	0.7193	50.88%	0.7321	63.64%
User 2 and 4	0.8214	62.05%	0.5658	32.89%	0.6746	57.42%
User 2 and 5	0.7902	58.93%	0.6491	46.93%	0.7273	62.68%
User 2 and 6	0.7768	62.05%	0.6667	47.37%	0.6746	57.89%
User 3 and 4	0.8393	61.61%	0.6799	31.14%	0.7223	52.63%
User 3 and 5	0.7813	57.14%	0.6140	37.72%	0.7368	55.98%
User 3 and 6	0.7321	58.48%	0.6842	40.79%	0.7033	52.15%
User 4 and 5	0.7813	56.25%	0.6096	27.19%	0.7273	52.15%
User 4 and 6	0.7589	58.93%	0.6447	28.51%	0.6938	48.33%
User 5 and 6	0.7366	56.25%	0.6491	38.6%	0.7273	52.63%
Kappa Value	0.5030	-	0.3309	-	0.3403	-

Table 6.13: User study evaluation. *kappa* statistics and interuser observed agreement. *kappa* interpretation according to Landis J.R and Koch G.G. from [146]: 0.0 ... 0.2 (slight); 0.2 ... 0.4 (fair); 0.4 ... 0.6 (moderate); 0.6 ... 0.8(substantial); 0.8 ... 1.0 (perfect). Column AoC denotes the percent of agreement on correct relations.

is the expected agreement by chance. Finally, we calculate the *kappa* coefficient as:

$$k = \frac{A_o - A_e}{1 - A_e}$$

In other words, *kappa* is a ratio between the number of agreements that were found beyond chance, to the total number of attainable agreements beyond chance.

Table 6.13 presents all the observed agreements between all combination pairs of 6 annotators and the percent of agreement on correct relations. We observe an overall kappa of 0.5030 for the "Organization"-cluster, 0.3309 for the "Brain"-cluster and 0.3403 for the "Human"-cluster. According to Landis, J. and Koch, G. from [146], the first result can be interpreted as *moderate* and the last two as *fair*. The results of the experiments on the WELTMODELL dataset support our claim that using SRL approach by adopting the integrity constraints rules is a promising direction for data cleaning on non-relational data.

6.6 Summary

The usual way to deal with data errors is to apply multiple data cleaning solutions [223], such as outlier detection [8], missing value imputation [245], or formulating data cleaning rules [62, 91]. Since *"one size does not fit all"* [223, 167], there is a need for domain- and application-specific cleaning solutions. In this section, we provided an approach of a

mapping data cleaning problem to probabilistic inference through a statistical relational learning formalism such as Markov logic. We developed a method to translate integrity constraints into a probabilistic logical language. This allows to reason over data quality dimensions violation in a probabilistic way. By applying the probabilistic joint inference, our method eliminates the need for the manual specification of the data cleaning rules execution order. We use the SRL formalism - Markov logic, which allows us to incorporate semantic constraints and to extend traditional data quality rules with domain- and application-specific knowledge. By means of probabilistic data cleaning, we obtained a method that (1) utilizes the probabilistic joint inference over interleaved data cleaning rules, and provides probabilistic repair; (2) takes the interleaved data quality issues into account and eliminates the need to specify the order of rule execution; and (3) expresses data quality rules (e.g. canonical and approximate integrity constraints) as a first-order logic formula to directly translate into the predictive model defined with the Markov logic formalism. The experimental results on relational and non-relational data showed that using the Markov logic formalism is a promising direction for probabilistic data cleaning.

7

Conclusion and Future Work

In this dissertation, we introduced data curation methods to make the data preparation process easier and more effective. We provided a systematic study for using metadata in data quality management, by creating a comprehensive mapping between metadata and data quality issues, such that it establishes the connection between dirty data and extractable metadata. Next, we categorized metadata by taking the composability of metadata into account. Based on the above categorization, we generalized the metadata composability as a set of EBNF rules, which enables data scientists to compose new metadata. Therefore, we provided an approach for exploiting canonical or newly generated metadata in error detection heuristics to create data cleaning strategies [233].

Furthermore, we have explained the problem of error detection as a classification task. We holistically combine several error detection strategies by using state-of-the-art ensemble learning algorithms [252], while taking the dataset's metadata into account [232, 159].

Finally, we proposed a declarative data cleaning approach based on statistical relational learning and probabilistic inference. We demonstrated how integrity constraints, expressed as first-order logic formulas, are translated into probabilistic logical languages, allowing us to reason over inaccuracies, inconsistencies or duplicates in a probabilistic way. Our approach allows the usage of probabilistic joint inference over interleaved data cleaning rules for the improvement of data quality. By using a declarative probabilistic-logical formalism such as Markov logic, we are able to incorporate more semantic constraints and, therefore, extend traditional data quality rules [234].

In this dissertation, we focus on effective data quality management by providing methods for data error detection and correction. We addressed several research questions of data quality management:

- How can we use the characteristics of the data in order to improve the error detection process?
- How can we effectively combine different data cleaning strategies to provide an effective and dataset-specific error detection method?
- How can we overcome the shortcomings of the *minimal repair* by the simultaneous treatment of error detection and correction, and using probabilistic inference for data cleaning?

The assertion of this dissertation is that it is feasible to build effective error detection methods that incorporate *dataset characteristics*, aggregate *various data cleaning strategies*, and employ the *probabilistic repair* operational semantic. Hence, we proposed (1) an approach for effective data quality assessment based on metadata information [233]; (2) a methodology for effective data cleaning strategies aggregation for data error detection [232, 159]; and (3) a concept for a declarative data cleaning approach based on statistical relational learning and probabilistic inference [234].

Our ultimate goal is to support data scientists to minimize the time and effort spent on data preparation in the data science workflow, by declaratively specifying probabilistic data cleaning rules and building agile and goal-oriented error detection routines, with the aid of the datasets metadata and machine learning approaches.

In future work, we continue to focus on data preparation and seek ways to advance them. With this goal in mind, we now describe three possible directions for the improvement of data preparation.

7.1 Error Detection and Repair as a Multi-Armed Bandit Problem

Error Detection. As the number of newly developed data cleaning systems increases [112, 36, 239], it becomes more important to experiment with the combination of all available systems and *select the most useful data cleaning strategies upfront.* Hence, fast experimentation has become increasingly relevant as systems become more complex and require more computational resources [234, 202]. Additionally to the *Best-K Systems Selection* approach suggested in Section 5.2.6, selecting the best m data cleaning strategies can also be formulated as a *multi-armed bandit* problem [39], where we face K unknown distributions, which denote K different data cleaning strategies. After performing n evaluations, our goal is to identify a subset of m distributions ($m \leq K$), also referred to as *arms*, corresponding to some specified criterion, such as the maximization of the reward (or the number of detected errors in our setting). Thus, applying the multi-armed bandit

algorithms is a promising direction to select the set of the most effective data cleaning systems.

Error Repair. In data cleaning, one of the most challenging tasks is *how to determine what is the best repair value for a given data error*. Optimization is commonly employed to determine the repair value if a set of potential repairs is available [202, 62, 234]. Being inherently a resource allocation technique, the bandit algorithms specify a strategy to select the most rewarding arm (error repair) in each turn. Considering the data correction system as an "arm", we could apply this theory on selecting the best possible "treatment" (system allocation or value assignment for repair) in each data point. The bandit algorithms offer the advantage of rapid experimentation because they test variants that have the greatest potential reward, meaning the *correct error repair*. The reward function might be expressed in terms of the data quality rules and integrity constraints (functional dependencies). Since bandit methods have demonstrated notable empirical performance [45, 151], these algorithms are suitable candidates for future research aimed at discovering the optimal assignment of the *erroneous values* corrections.

7.2 Reinforcement Learning for Data Cleaning

Considering the recent advances in machine learning and reinforcement learning [140, 158, 181], the current trend in database research is to learn various heuristics instead of specifying them by hand. Encouraged by these results, we are interested in addressing the problem of *how do we measure the quality of data cleaning in a given data science workflow upfront?* This challenge reveals a set of combinatorial issues if we consider the combinations of all possible data cleaning strategies, each with many parameters, the order of data cleaning steps, and various iterations of specific data cleaning tasks.

Generally, *reinforcement learning* models are able to map scenarios (suspicious data values) to appropriate actions (rules for detecting data errors), with the goal of maximizing a cumulative reward (overall error detection). Initially, the learner is not shown which action is best. Instead, the learner must discover the best error detection rules (heuristic) and their order through trial and error, by either exploiting current knowledge or exploring unknown variants of rules. We argue that by using our mapping between metadata and data quality issues (see Chapter 3) as an initial feature space, the specialized heuristics (e.g., learning thresholds) can be learned to minimize the false positives rate produced by all possible heuristics. Therefore one of the potential future directions is to develop an error detection workflow generator that takes the heuristics training set as input (as described in Chapter 3) and returns a set of predicates that identify dirty candidate records.

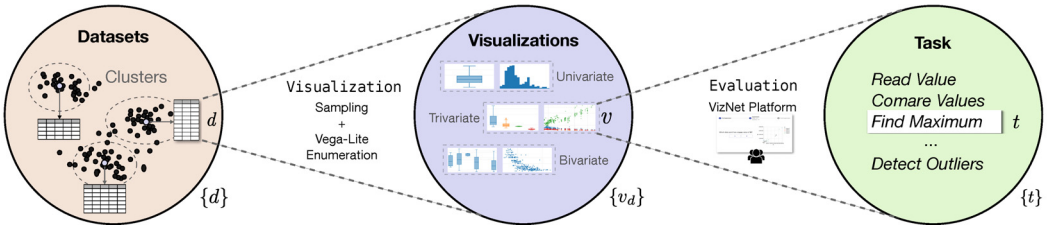

Figure 7.1: VizNet enables data scientists and visualization researchers to aggregate data and enumerate visual encodings. Figure used with permission [118].

7.3 Effectiveness of Visual Encoding for Data Curation

Driven by the requirement to integrate ever more resources, recent scientific and enterprise applications increasingly face the problem of data variety[1] [60]. Now, data has a variety of shapes, including textual content, IoT data, time series, NoSQL and SQL stores, and event streams [59]. Considering the increasing number of data types, the inspection of the data requires effective profiling [115] and visualization to characterize these datasets and make a decision whether the datasets have an acceptable quality [128]. On the other hand, data visualization is a combinatorial design problem: a dataset can be differently visualized, and a single visualization can be used for various analytic tasks [118, 134]. The recently proposed VizNet corpus is a large-scale visualization learning and benchmarking repository. This corpus provides a set of triplets consisting of *(data, visual encoding, data analytic task)* (see Figure 7.1). Hu K. et al. [118] trained a machine learning model to predict the effectiveness of the visualization for the particular task, such as *find maximum, compare averages, read value, compare values, cluster* and *find outlier*. This model could be used to provide a visualization recommendation system for data cleaning. We could leverage VizNet to study and develop effective visualizations for data preparation and data cleaning since data cleaning is a set of tasks comprising error detection and error correction. In future work we will therefore answer the question: *what visual encodings are efficient to perform tasks for data curation?*

[1] `http://bit.ly/big-data-3v`

References

[1] Abedjan, Z., Akcora, C. G., Ouzzani, M., Papotti, P., and Stonebraker, M. (2015a). Temporal rules discovery for web data cleaning. *Proceedings of the VLDB Endowment (PVLDB)*, 9:336–347.

[2] Abedjan, Z., Chu, X., Deng, D., Fernandez, R. C., Ilyas, I. F., Ouzzani, M., Papotti, P., Stonebraker, M., and Tang, N. (2016). Detecting data errors: Where are we and what needs to be done? In *Proceedings of the VLDB Endowment (PVLDB)*.

[3] Abedjan, Z., Golab, L., and Naumann, F. (2015b). Profiling relational data: a survey. In *VLDB Journal*, volume 24, pages 557–581.

[4] Abedjan, Z., Golab, L., Naumann, F., and Papenbrock, T. (2018). Data profiling. *Synthesis Lectures on Data Management*.

[5] Abedjan, Z. and Naumann, F. (2011). Advancing the discovery of unique column combinations. In *Proceedings of the International Conference on Data Engineering (ICDE)*.

[6] Abedjan, Z. and Naumann, F. (2014). Amending rdf entities with new facts. In *European Semantic Web Conference*, pages 131–143.

[7] Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.

[8] Aggarwal, C. C. (2015). Outlier analysis. In *Data mining*. Springer.

[9] Akbik, A. and Michael, T. (2014). The weltmodell: A data-driven commonsense knowledge base. In *9th Edition of the Language Resources and Evaluation Conference, LREC 2014*.

[10] Amazon (2019). "amazon mechanical turk". *online*. https://https://www.mturk.com/, Accessed: 2019-07-24.

[11] Anderson, C. R., Domingos, P., and Weld, D. S. (2002). Relational markov models and their application to adaptive web navigation. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*.

[12] Arasu, A., Ré, C., and Suciu, D. (2009). Large-scale deduplication with constraints using dedupalog. In *Proceedings of the International Conference on Data Engineering (ICDE)*.

[13] Arenas, M., Bertossi, L., and Chomicki, J. (1999). Consistent query answers in inconsistent databases. In *Proceedings of the 18-th symposium on Principles of database systems*.

[14] Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., Meng, X., Kaftan, T., Franklin, M. J., Ghodsi, A., et al. (2015). Spark sql: Relational data processing in spark. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1383–1394.

REFERENCES

[15] Arocena, P. C., Glavic, B., Mecca, G., Miller, R. J., Papotti, P., and Santoro, D. (2015). Messing up with bart: error generation for evaluating data-cleaning algorithms. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, volume 9, pages 36–47.

[16] Assadi, A., Milo, T., and Novgorodov, S. (2017). Dance: Data cleaning with constraints and experts. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 1409–1410.

[17] Barnett, V. and T., L. (1995). *Outliers in Statistical Data*. Wiley Online Library.

[18] Batini, C., Scannapieco, M., et al. (2016). *Data and information quality*. Springer.

[19] Bergé, J.-M., Fonkoua, A., Maginot, S., and Rouillard, J. (1993). Xnor operator. In *VHDL'92*, pages 81–83. Springer.

[20] Bergman, M., Milo, T., Novgorodov, S., and Tan, W.-C. (2015). Query-oriented data cleaning with oracles. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1199–1214.

[21] Berti-Equille, L. (2019). Learn2clean: Optimizing the sequence of tasks for web data preparation. In *The World Wide Web Conference*, pages 2580–2586. ACM.

[22] Berti-Equille, L., Dasu, T., and Srivastava, D. (2011). Discovery of complex glitch patterns: A novel approach to quantitative data cleaning. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 733–744.

[23] Berti-Equille, L., Harmouch, H., Naumann, F., Novelli, N., and Thirumuruganathan, S. (2018). Discovery of genuine functional dependencies from relational data with missing values. In *Proceedings of the VLDB Endowment (PVLDB)*, volume 11, pages 880–892.

[24] Bertossi, L. (2011). Database repairing and consistent query answering. *Synthesis Lectures on Data Management*.

[25] Bertossi, L., Kolahi, S., and Lakshmanan, L. V. (2013). Data cleaning and query answering with matching dependencies and matching functions. *Theory of Computing Systems*, 52(3):441–482.

[26] Beskales, G., Ilyas, I. F., and Golab, L. (2010). Sampling the repairs of functional dependency violations under hard constraints. In *Proceedings of the International Conference on Very Large Databases (VLDB)*.

[27] Beskales, G., Soliman, M. A., Ilyas, I. F., and Ben-David, S. (2009). Modeling and querying possible repairs in duplicate detection. In *Proceedings of the International Conference on Very Large Databases (VLDB)*.

[28] Biemer, P. P. and Lyberg, L. E. (2003). *Introduction to survey quality*. John Wiley & Sons.

[29] Bishop, C. M. (2006). Pattern recognition. *Machine Learning*.

[30] Bleiholder, J. and Naumann, F. (2006). *Conflict handling strategies in an integrated information system*. Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät.

[31] Blitzstein, J. K. and Hwang, J. (2014). *Introduction to probability*. Chapman and Hall/CRC.

[32] Blum, A. L. and Rivest, R. L. (1993). Training a 3-node neural network is np-complete. In *Machine learning: From theory to applications*, pages 9–28.

[33] Bohannon, P., Fan, W., Flaster, M., and Rastogi, R. (2005). A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 143–154.

[34] Bohannon, P., Fan, W., Geerts, F., Jia, X., and Kementsietsidis, A. (2007). Conditional functional dependencies for data cleaning. In *Proceedings of the International Conference on Data Engineering (ICDE)*.

[35] Box, G. E. and Draper, N. R. (1987). *Empirical model-building and response surfaces*. John Wiley & Sons.

[36] Breck, E., Polyzotis, N., Roy, S., Whang, S., and Zinkevich, M. (2019). Data validation for machine learning. In *Conference on Systems and Machine Learning (SysML)*.

[37] Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

[38] Bruni, R. and Sassano, A. (2001). Errors detection and correction in large scale data collecting. In *International Symposium on Intelligent Data Analysis*.

[39] Bubeck, S., Wang, T., and Viswanathan, N. (2013). Multiple identifications in multi-armed bandits. In *International Conference on Machine Learning*, pages 258–265.

[40] Burdick, D., Fagin, R., Kolaitis, P. G., Popa, L., and Tan, W.-C. (2015). A declarative framework for linking entities. In *Proceedings of the International Conference on Database Theory (ICDT)*.

[41] Cao, L., Yang, D., Wang, Q., Yu, Y., Wang, J., and Rundensteiner, E. A. (2014). Scalable distance-based outlier detection over high-volume data streams. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 76–87.

[42] Caruana, R. and Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proceedings of the International Conference on Machine learning*.

[43] Caruccio, L., Deufemia, V., and Polese, G. (2016). Relaxed functional dependencies—a survey of approaches. *IEEE Transactions on Knowledge and Data Engineering*.

[44] Casanova, M. A., Fagin, R., and Papadimitriou, C. H. (1984). Inclusion dependencies and their interaction with functional dependencies. *Journal of Computer and System Sciences*.

[45] Chapelle, O. and Li, L. (2011). An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*, pages 2249–2257.

[46] Chen, K., Chen, H., Conway, N., Hellerstein, J. M., and Parikh, T. S. (2011). Usher: Improving data quality with dynamic forms. *IEEE Transactions on Knowledge and Data Engineering*, 23:1138–1153.

[47] Chen, W., Fan, W., and Ma, S. (2009a). Analyses and validation of conditional dependencies with built-in predicates. In *International Conference on Database and Expert Systems Applications*.

[48] Chen, W., Fan, W., and Ma, S. (2009b). Incorporating cardinality constraints and synonym rules into conditional functional dependencies. *Information Processing Letters*, 109(14).

[49] Chen, X.-y. and Zhan, Y.-y. (2008). Multi-scale anomaly detection algorithm based on infrequent pattern of time series. *Journal of Computational and Applied Mathematics*.

[50] Chiang, F. and Miller, R. J. (2011). A unified model for data and constraint repair. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 446–457.

[51] Chodorow, M. and Leacock, C. (2000). An unsupervised method for detecting grammatical errors. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*.

[52] Chomicki, J. and Marcinkowski, J. (2005). Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197.

[53] Chu, X., Ilyas, I. F., Krishnan, S., and Wang, J. (2016). Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2201–2206.

[54] Chu, X., Ilyas, I. F., and Papotti, P. (2013). Holistic data cleaning: Putting violations into context. In *Proceedings of the International Conference on Data Engineering (ICDE)*.

[55] Chu, X., Morcos, J., Ilyas, I. F., Ouzzani, M., Papotti, P., Tang, N., and Ye, Y. (2015). Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[56] Chung, Y., Krishnan, S., and Kraska, T. (2017). A data quality metric (dqm): how to estimate the number of undetected errors in data sets. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, volume 10, pages 1094–1105.

[57] Codd, E. F. (1972). Further normalization of the database relational model. *Data Base Systems*.

[58] Cohen, J. (1960). Introduces kappa as a way of calculating inter rater agreement between two raters. *Educational and Psychological Measurement*.

[59] Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., et al. (2013). Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems*, 31(3).

[60] Council, U. N. R. (2013). *Frontiers in Massive Data Analysis*. The National Academies Press.

[61] Cussens, J. (2007). Logic-based formalisms for statistical relational learning. *Introduction to Statistical Relational Learning, Chapter 9*.

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

[63] Dalvi, N., Dasgupta, A., Kumar, R., and Rastogi, V. (2013). Aggregating crowdsourced binary ratings. In *Proceedings of the International World Wide Web Conference (WWW)*.

[64] Dalvi, N. and Suciu, D. (2013). The dichotomy of probabilistic inference for unions of conjunctive queries. *Journal of the ACM*, 59(6).

[65] Dasu, T. and Johnson, T. (2003). *Exploratory data mining and data cleaning*, volume 479. John Wiley & Sons.

[66] Dasu, T., Johnson, T., Muthukrishnan, S., and Shkapenyuk, V. (2002). Mining database structure; or, how to build a data quality browser. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 240–251.

[67] De, S., Hu, Y., Meduri, V. V., Chen, Y., and Kambhampati, S. (2016). Bayeswipe: A scalable probabilistic framework for improving data quality. *Journal of Data and Information Quality (JDIQ)*, 8(1):5.

[68] De Sa, C., Ilyas, I. F., Kimelfeld, B., Re, C., and Rekatsinas, T. (2019). A formal framework for probabilistic unclean databases. In *Proceedings of the International Conference on Database Theory (ICDT)*.

[69] De Sa, C., Ratner, A., Ré, C., Shin, J., Wang, F., Wu, S., and Zhang, C. (2016). Deepdive: Declarative knowledge base construction. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, volume 45, pages 60–67.

[70] Deng, D., Fernandez, R. C., Abedjan, Z., Wang, S., Stonebraker, M., Elmagarmid, A., Ilyasl, I. F., Madden, S., Ouzzani, M., and Tang, N. (2017). The data civilizer system. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*.

[71] Dietterich, T. G. (2000). Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15.

[72] Do, H.-H. and Rahm, E. (2002). Coma: a system for flexible combination of schema matching approaches. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 610–621.

[73] Domingos, P. and Lowd, D. (2009). *Markov logic: An interface layer for artificial intelligence*. Morgan & Claypool Publishers.

[74] Dong, X. L. (2015). *Big Data Integration*. Synthesis Lectures on Data Management, Morgan & Claypool.

[75] Dong, X. L. and Rekatsinas, T. (2018). Data integration and machine learning: A natural synergy. In *Proceedings of the 2018 International Conference on Management of Data*.

[76] Duda, R. O., Hart, P. E., and Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

[77] Efron, B. and Tibshirani, R. J. (1993). An introduction to the bootstrap. *Chapman & Hall*.

[78] Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. (2007). Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, 19(1):1–16.

[79] Fagin, R. (1980). Horn clauses and database dependencies. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*.

[80] Fan, W. (2008). Dependencies revisited for improving data quality. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[81] Fan, W., Gao, H., Jia, X., Li, J., and Ma, S. (2011). Dynamic constraints for record matching. In *Proceedings of the International Conference on Very Large Databases (VLDB)*.

[82] Fan, W. and Geerts, F. (2012). *Foundations of data quality management*. Morgan & Claypool Publishers.

[83] Fan, W., Geerts, F., Jia, X., and Kementsietsidis, A. (2008). Conditional functional dependencies for capturing data inconsistencies. In *ACM Transactions on Database Systems (TODS)*.

[84] Fan, W., Geerts, F., Tang, N., and Yu, W. (2013). Inferring data currency and consistency for conflict resolution. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 470–481.

[85] Fan, W., Geerts, F., and Wijsen, J. (2012). Determining the currency of data. In *ACM Transactions on Database Systems (TODS)*, volume 37, page 25.

[86] Fan, W., Li, J., Ma, S., Tang, N., and Yu, W. (2010). Towards certain fixes with editing rules and master data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, volume 3, pages 173–184.

[87] Fan, W., Ma, S., Tang, N., and Yu, W. (2014). Interaction between record matching and data repairing. *Journal of Data and Information Quality*.

[88] Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database*. Language, speech, and communication. MIT Press.

[89] Fellegi, I. P. and Holt, D. (1976). A systematic approach to automatic edit and imputation. *Journal of the American Statistical Association*.

[90] Felzenszwalb, P. F. and Huttenlocher, D. P. (2006). Efficient belief propagation for early vision. *International journal of computer vision*, 70(1):41–54.

[91] Geerts, F., Giansalvatore, M., Papotti, P., and Santore, D. (2013). The llunatic data cleaning framework. In *Proceedings of the VLDB Endowment (PVLDB)*.

[92] Genesereth, M. R. and Nilsson, N. J. (1987). *Logical foundations of artificial intelligence*. Intelligence. Morgan Kaufmann.

[93] Getoor, L. and Taskar, B. (2007). *Introduction to statistical relational learning*. MIT press.

[94] Gilks, W. R., Richardson, S., and Spiegelhalter, D. (1995). *Markov chain Monte Carlo in practice*. Chapman and Hall/CRC.

[95] Gogate, V. and Domingos, P. (2016). Probabilistic theorem proving. *Communications of the ACM*, 59:107–115.

[96] Gokhale, C., Das, S., Doan, A., Naughton, J. F., Rampalli, N., Shavlik, J., and Zhu, X. (2014). Corleone: hands-off crowdsourcing for entity matching. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 601–612. ACM.

[97] Golab, L., Karloff, H., Korn, F., and Srivastava, D. (2010). Data auditor: Exploring data quality and semantics using pattern tableaux. *Proceedings of the VLDB Endowment (PVLDB)*, 3(1-2):1641–1644.

[98] Goldberg, Y. and Orwant, J. (2013). A dataset of syntactic-ngrams over time from a very large corpus of english books. In *Second Joint Conference on Lexical and Computational Semantics, Association for Computational Linguistics*.

[99] Golshan, B., Halevy, A., Mihaila, G., and Tan, W.-C. (2017). Data integration: After the teenage years. In *Proceedings of the 36th Symposium on Principles of Database Systems*.

[100] Gribkoff, E., Suciu, D., and Van den Broeck, G. (2014a). Lifted probabilistic inference: A guide for the database researcher. *IEEE Data Eng. Bull.*

[101] Gribkoff, E., Van den Broeck, G., and Suciu, D. (2014b). The most probable database problem. In *Proceedings of the First International Workshop on Big Uncertain Data*.

[102] Guo, P. (2013). Data science workflow: Overview and challenges. *Communications of the ACM*.

[103] Guo, P. J. (2012). *Software tools to facilitate research programming*. PhD thesis, Stanford University Stanford, CA.

[104] Gupta, M., Gao, J., Aggarwal, C. C., and Han, J. (2014). Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 26(9):2250–2267.

[105] Haas, D., Krishnan, S., Wang, J., Franklin, M. J., and Wu, E. (2015). Wisteria: Nurturing scalable data cleaning infrastructure. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, volume 8, pages 2004–2007.

[106] Halevy, A., Doan, A., and Ives, Z. (2012). Principles of data integration. *Morgan Kaufmann*.

[107] Han, J., Pei, J., and Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

[108] Hartigan, J. (1975). *Clustering algorithms*. Wiley.

[109] Hazelwood, K., Bird, S., Brooks, D., Chintala, S., Diril, U., Dzhulgakov, D., Fawzy, M., Jia, B., Jia, Y., Kalro, A., et al. (2018). Applied machine learning at facebook: A datacenter infrastructure perspective. In *High Performance Computer Architecture (HPCA)*, pages 620–629.

[110] He, J., Veltri, E., Santoro, D., Li, G., Mecca, G., Papotti, P., and Tang, N. (2016). Interactive and deterministic data cleaning. In *Proceedings of the 2016 International Conference on Management of Data*, pages 893–907. ACM.

[111] Heath, T. and Bizer, C. (2011). *Linked data: Evolving the web into a global data space*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers.

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

[113] Heise, A., Kasneci, G., and Naumann, F. (2014). Estimating the number and sizes of fuzzy-duplicate clusters. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 959–968.

[114] Hellerstein, J. M. (2008). Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)*.

[115] Hellerstein, J. M., Sreekanti, V., Gonzalez, J. E., Dalton, J., Dey, A., Nag, S., Ramachandran, K., Arora, S., Bhattacharyya, A., Das, S., et al. (2017). Ground: A data context service. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*.

[116] Hodge, V. and Austin, J. (2004). A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126.

[117] Hould, J.-N. (2017). Craft beers dataset. Version 1.

[118] Hu, K., Gaikwad, N., Bakker, M., Hulsebos, M., Zgraggen, E., Hidalgo, C., Kraska, T., Li, G., and Demiralp, Ç. (2019a). Viznet: Towards a large-scale visualization learning and benchmarking repository. In *ACM Human Factors in Computing Systems (CHI)*.

[119] Hu, K. Z., Bakker, M. A., Li, S., Kraska, T., and Hidalgo, C. A. (2019b). Vizml: A machine learning approach to visualization recommendation. In *CHI*.

[120] Hua, M. and Pei, J. (2007). Cleaning disguised missing data: a heuristic approach. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 950–958.

[121] Huhtala, Y., Kärkkäinen, J., Porkka, P., and Toivonen, H. (1999). Tane: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal*, 42(2):100–111.

[122] Ioannidis, Y. (2003). The history of histograms (abridged). In *Proceedings of the VLDB Endowment (PVLDB)*.

[123] Ioannidis, Y. and Poosala, V. (1995). Histogram-based solutions to diverse database estimation problems. *IEEE Data Eng. Bull.*

[124] Jensen, C. S., Snodgrass, R. T., and Soo, M. D. (1996). Extending existing dependency theory to temporal databases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*.

[125] Jurafsky, D. and Martin, J. H. (2008). *Speech and language processing (prentice hall series in artificial intelligence)*. Prentice Hall.

[126] Kaggle (2019). "the state of machine learning and data science 2017". *Kaggle*. https://bit.ly/2KopcwB Accessed: 2019-02-11.

[127] Kandel, S., Paepcke, A., Hellerstein, J., and Heer, J. (2011). Wrangler: Interactive visual specification of data transformation scripts. In *ACM CHI Conference on Human Factors in Computing Systems*.

[128] Kandel, S., Parikh, R., Paepcke, A., Hellerstein, J. M., and Heer, J. (2012). Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pages 547–554.

[129] Kautz, H. A., Selman, B., and Jiang, Y. (1996). A general stochastic approach to solving problems with hard and soft constraints. *Satisfiability Problem: Theory and Applications*.

[130] Kersting, K. (2005). An inductive logic programming approach to statistical relational learning. In *Proceedings of the 2005 Conference on an Inductive Logic Programming Approach to Statistical Relational Learning*.

[131] Khayyat, Z., Ilyas, I. F., Jindal, A., Madden, S., Ouzzani, M., Papotti, P., Quiané-Ruiz, J.-A., Tang, N., and Yin, S. (2015a). Bigdansing: A system for big data cleansing. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[132] Khayyat, Z., Ilyas, I. F., Jindal, A., Madden, S., Ouzzani, M., Papotti, P., Quiané-Ruiz, J.-A., Tang, N., and Yin, S. (2015b). Bigdansing: A system for big data cleansing. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1215–1230.

[133] Kim, W., Choi, B.-J., Hong, E.-K., Kim, S.-K., and Lee, D. (2003). A taxonomy of dirty data. *Data mining and knowledge discovery*.

[134] Kim, Y. and Heer, J. (2018). Assessing effects of task and data distribution on the effectiveness of visual encodings. In *Computer Graphics Forum*, volume 37, pages 157–167.

[135] Knuth, D. E. (1964). Backus normal form vs. backus naur form. *Communications of the ACM*.

[136] Kolahi, S. and Lakshmanan, L. V. (2009). On approximating optimum repairs for functional dependency violations. In *Proceedings of the International Conference on Database Theory (ICDT)*.

[137] Koller, D., Friedman, N., and Bach, F. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

[138] Konda, P., Das, S., Suganthan GC, P., Doan, A., Ardalan, A., Ballard, J. R., Li, H., Panahi, F., Zhang, H., Naughton, J., et al. (2016). Magellan: Toward building entity matching management systems. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, volume 9, pages 1197–1208.

[139] Koudas, N., Saha, A., Srivastava, D., and Venkatasubramanian, S. (2009). Metric functional dependencies. In *Proceedings of the International Conference on Data Engineering (ICDE)*.

[140] Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. (2018). The case for learned index structures. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[141] Krishnan, S., Franklin, M. J., Goldberg, K., and Wu, E. (2017). Boostclean: Automated error detection and repair for machine learning. *arXiv preprint arXiv:1711.01299*.

[142] Krishnan, S., Haas, D., Franklin, M. J., and Wu, E. (2016a). Towards reliable interactive data cleaning: A user survey and recommendations. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*.

[143] Krishnan, S., Wang, J., Franklin, M. J., Goldberg, K., and Kraska, T. (2015a). Stale view cleaning: Getting fresh answers from stale materialized views. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, volume 8, pages 1370–1381.

[144] Krishnan, S., Wang, J., Franklin, M. J., Goldberg, K., Kraska, T., Milo, T., and Wu, E. (2015b). Sampleclean: Fast and reliable analytics on dirty data. *IEEE Data Eng. Bull.*, 38(3):59–75.

[145] Krishnan, S., Wang, J., Wu, E., Franklin, M. J., and Goldberg, K. (2016b). ActiveClean: interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment (PVLDB)*, 9(12):948–959.

[146] Landis, J. R. and Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*.

[147] Laranjeiro, N., Soydemir, S. N., and Bernardino, J. (2015). A survey on data quality: classifying poor data. In *Dependable Computing (PRDC)*.

[148] Laure, B.-E., Angela, B., and Tova, M. (2018). Machine learning to data management: A round trip. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 1735–1738.

[149] Lewis, J. R., Utesch, B. S., and Maher, D. E. (2013). Umux-lite: When there's no time for the sus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

[150] Leys, C., Ley, C., Klein, O., Bernard, P., and Licata, L. (2013). Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764–766.

[151] Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670.

[152] Li, L., Peng, T., and Kennedy, J. (2011). A rule based taxonomy of dirty data. *GSTF Journal on Computing (JoC)*, 1(2).

[153] Li, X., Dong, X. L., Lyons, K., Meng, W., and Srivastava, D. (2012). Truth finding on the deep web: Is the problem solved? In *Proceedings of the International Conference on Very Large Databases (VLDB)*.

[154] Li, Y., Gao, J., Meng, C., Li, Q., Su, L., Zhao, B., Fan, W., and Han, J. (2016). A survey on truth discovery. *SIGKDD Explor. Newsl.*, 17(2):1–16.

[155] LIM, E. P. and Srivastava, J. (1993). Entity identification in database integration: an evidential reasoning approach. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 294–301. NDA.

[156] Livshits, E., Kimelfeld, B., and Roy, S. (2018). Computing optimal repairs for functional dependencies. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[157] Lopatenko, A. and Bravo, L. (2007). Efficient approximation algorithms for repairing inconsistent databases. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 216–225.

[158] Ma, L., Van Aken, D., Hefny, A., Mezerhane, G., Pavlo, A., and Gordon, G. J. (2018). Query-based workload forecasting for self-driving database management systems. In *Proceedings of the 2018 International Conference on Management of Data*, pages 631–645.

[159] Mahdavi, M., Neutatz, F., Visengeriyeva, L., and Abedjan, Z. (2019). Towards automated data cleaning workflows. In *Proceedings of the LWDA 2019*.

[160] Maletic, J. I. and Marcus, A. (2000). Data cleansing: Beyond integrity analysis. In *Information Quality*, pages 200–209.

[161] Management, M. S. (2019). "data, analytics, and ai: How trust delivers value". *MIT Sloan Management Review*. http://bit.ly/mit-data-quality, Accessed: 20.03.2019.

[162] Mannino, M. V., Chu, P., and Sager, T. (1988). Statistical profile estimation in database systems. *ACM Computing Surveys (CSUR)*, 20(3):191–221.

[163] Maydanchik, A. (2007). *Data quality assessment*. Technics publications.

[164] Mayfield, C., Neville, J., and Prabhakar, S. (2010). Eracer: a database approach for statistical inference and data cleaning. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 75–86.

[165] McCarthy, J. and Levin, M. I. (1965). *LISP 1.5 programmer's manual*. MIT press.

[166] Metmuseum (2018). The metropolitan museum of art open access. https://github.com/metmuseum/openaccess Accessed:2019-03-28.

[167] Michael, S., Nik, B.-H., Liam, C., Larry, S., and Andy, P. (2018). *Getting Data Operations Right*. OReilly Media.

[168] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv*.

[169] Mintz, M., Bills, S., Snow, R., and Jurafsky, D. (2009). Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1003–1011.

[170] Müller, H. and Freytag, J. (2003). Problems, methods and challenges in comprehensive data cleansing. *Technical Report HUB-IB-164, Humboldt-Universität zu Berlin, Institut für Informatik.*

[171] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective.* The MIT Press.

[172] Naumann, F. (2003). *Quality-driven query answering for integrated information systems*, volume 2261. Springer.

[173] Naumann, F. (2014). Data profiling revisited. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, volume 42, pages 40–49.

[174] Naus, J. I., Johnson, T. G., and Montalvo, R. (1972). A probabilistic model for identifying errors in data editing. *Journal of the American Statistical Association*, 67:943–950.

[175] Nelder, J. A. and Wedderburn, R. W. (1972). Generalized linear models. *Journal of the Royal Statistical Society.*

[176] Neville, J. and Jensen, D. (2003). Collective classification with relational dependency networks. In *Proceedings of the Second International Workshop on Multi-Relational Data Mining.*

[177] Nicolas, J.-M. (1978). First order logic formalization for functional, multivalued and mutual dependencies. In *Proceedings of the International Conference on Management of Data (SIGMOD).*

[178] Niu, F., Ré, C., Doan, A., and Shavlik, J. (2011). Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, volume 4, pages 373–384.

[179] Noessner, J., Niepert, M., and Stuckenschmidt, H. (2013). Rockit: Exploiting parallelism and symmetry for map inference in statistical relational models. In *Proceedings of the National Conference on Artificial Intelligence (AAAI).*

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

[181] Ortiz, J., Balazinska, M., Gehrke, J., and Keerthi, S. S. (2018). Learning state representations for query optimization with deep reinforcement learning. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, page 4.

[182] Papenbrock, T., Bergmann, T., Finke, M., Zwiener, J., and Naumann, F. (2015). Data profiling with metanome. In *Proceedings of the VLDB Endowment (PVLDB).*

[183] Papenbrock, T. and Naumann, F. (2016). A hybrid approach to functional dependency discovery. In *Proceedings of the 2016 International Conference on Management of Data*, pages 821–833.

[184] Parisi, F., Strino, F., Nadler, B., and Kluger, Y. (2014). Ranking and combining multiple predictors without labeled data. *Proceedings of the National Academy of Sciences.*

[185] Park, H. and Widom, J. (2014). Crowdfill: collecting structured data from the crowd. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 577–588. ACM.

[186] Patil, D. (2012). *Data Jujitsu*. OReilly Media, Inc.

[187] Pearl, J. (2014). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier.

[188] Pearson, R. K. (2005). *Mining imperfect data: Dealing with contamination and incomplete records*. Siam.

[189] Pearson, R. K. (2006). The problem of disguised missing data. *ACM SIGKDD Explorations Newsletter*, 8(1):83–92.

[190] Pit Claudel, C., Mariet, Z., Harding, R., and Madden, S. (2016). Outlier detection in heterogeneous datasets using automatic tuple expansion. Technical Report, MIT.

[191] Platanios, E., Poon, H., Mitchell, T. M., and Horvitz, E. J. (2017). Estimating accuracy from unlabeled data: A probabilistic logic approach. *Advances in Neural Information Processing Systems*.

[192] Poon, H. and Domingos, P. (2008). Joint unsupervised coreference resolution with markov logic. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[193] Prokoshyna, N., Szlichta, J., Chiang, F., Miller, R. J., and Srivastava, D. (2015). Combining quantitative and logical data cleaning. volume 9, pages 300–311.

[194] Qahtan, A. A., Elmagarmid, A., Castro Fernandez, R., Ouzzani, M., and Tang, N. (2018). Fahes: A robust disguised missing values detector. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*.

[195] Rahm, E. and Do, H. H. (2000). Data cleaning: Problems and current approaches.

[196] Rahman, P., Hebert, C., and Nandi, A. (2018). Icarus: minimizing human effort in iterative data completion. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, volume 11, pages 2263–2276.

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

[198] Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., and Ré, C. (2017). Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, volume 11, pages 269–282.

[199] Redman, T. C. (2016a). "bad data costs the u.s. $3 trillion per year". *Harvard Business Review*. http://bit.ly/bad-data-costs, Accessed: 2019-02-11.

[200] Redman, T. C. (2016b). *Getting in front on data: who does what*. Technics Publications.

[201] Redman, T. C. (2018). "if your data is bad, your machine learning tools are useless". *Harvard Business Review*. https://bit.ly/2InCpnA, Accessed: 2019-02-11.

[202] Rekatsinas, T., Chu, X., Ilyas, I. F., and Ré, C. (2017). Holoclean: Holistic data repairs with probabilistic inference. In *Proceedings of the VLDB Endowment (PVLDB)*.

[203] Richardson, M. and Domingos, P. (2006). Markov logic networks. *Machine learning*.

[204] Riedel, S. (2008a). Improving the accuracy and efficiency of map inference for markov logic. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*.

[205] Riedel, S. (2008b). Improving the accuracy and efficiency of map inference for markov logic. In *Proceedings of the UAI 2018*, pages 468–475.

[206] Riedel, S. and Meza-Ruiz, I. (2008). Collective semantic role labelling with markov logic. In *The SIGNLL Conference on Computational Natural Language Learning (CoNLL)*.

[207] Riley, J. (2017). Understanding metadata. what is metadata, and what is it for? *Groups.niso.org*. http://bit.ly/niso-metadata Accessed: 2019-02-11.

[208] Roth, D. (1996). On the hardness of approximate reasoning. *Artificial Intelligence*.

[209] Roth, D. and Yih, W.-t. (2005). Integer linear programming inference for conditional random fields. In *Proceedings of the International Conference on Machine learning*.

[210] Sarawagi, S. and Bhamidipaty, A. (2002). Interactive deduplication using active learning. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 269–278.

[211] Schwing, A., Hazan, T., Pollefeys, M., and Urtasun, R. (2011). Distributed message passing for large scale graphical models. In *Proceedings of the Conference on Computer Vision and Pattern Recognition 2011*, pages 1833–1840.

[212] Sharat Menon, E. Z. (2019). "market guide for data preparation tools". *online*. https://www.gartner.com/en/documents/3906957/market-guide-for-data-preparation-tools, Accessed: 2019-07-24.

[213] Shen, W., Wang, J., and Han, J. (2015). Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*.

[214] Sinha, A., Shen, Z., Song, Y., Ma, H., Eide, D., Hsu, B.-j. P., and Wang, K. (2015). An overview of microsoft academic service (mas) and applications. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 243–246.

[215] Slaney, M. and Casey, M. (2008). Locality-sensitive hashing for finding nearest neighbors. *IEEE Signal processing magazine*, 25(2):128–131.

[216] Song, S. and Chen, L. (2011). Differential dependencies: Reasoning and discovery. In *ACM Transactions on Database Systems (TODS)*.

[217] Song, S., Chen, L., and Yu, J. X. (2010). Extending matching rules with conditions. In *Proceedings of the 8th International Workshop on Quality in Databases*.

[218] Song, S., Li, C., and Zhang, X. (2015a). Turn waste into wealth: On simultaneous clustering and cleaning over dirty data. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 1115–1124.

[219] Song, S., Zhang, A., Chen, L., and Wang, J. (2015b). Enriching data imputation with extensive similarity neighbors. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, volume 8, pages 1286–1297.

[220] Sontag, D. A. (2010). *Approximate inference in graphical models using LP relaxations*. PhD thesis, Massachusetts Institute of Technology.

[221] Statista (2019). "digital economy compass 2019". *The Statistics Portal*. https://www.statista.com/page/compass, Accessed: 2019-04-20.

[222] Stonebraker, M., Beskales, G., Pagan, A., Bruckner, D., Cherniack, M., Xu, S., Analytics, V., Ilyas, I. F., and Zdonik, S. (2013). Data curation at scale: The data tamer system. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*.

[223] Stonebraker, M. and Ilyas, I. F. (2018). Data integration: The current status and the way forward. *IEEE Data Eng. Bull.*

[224] Subramaniam, S., Palpanas, T., Papadopoulos, D., Kalogeraki, V., and Gunopulos, D. (2006). Online outlier detection in sensor data using non-parametric models. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 187–198.

[225] Suchanek, F. M., Sozio, M., and Weikum, G. (2009). Sofie: a self-organizing framework for information extraction. In *Proceedings of the International World Wide Web Conference (WWW)*.

[226] Susan Moore, G. I. (2018). "how to create a business case for data quality improvement". *online*. https://www.gartner.com/smarterwithgartner/how-to-create-a-business-case-for-data-quality-improvement/, Accessed: 2019-04-11.

[227] Taskar, B., Chatalbashev, V., Koller, D., and Guestrin, C. (2005). Learning structured prediction models: A large margin approach. In *Proceedings of the International Conference on Machine learning*.

[228] Trifacta (2019). Supported data types - trifacta wrangler. *Trifacta Documentation*. https://docs.trifacta.com/display/PE/Supported Data Types, Accessed: 2019-02-11.

[229] Tye, R., Joseph M., H., Jeffrey, H., Sean, K., and Connor, C. (2017). *Principles of Data Wrangling: Practical Techniques for Data Preparation*. OReilly Media, Inc.

[230] Van den Broeck, G., Meert, W., and Darwiche, A. (2014). Skolemization for weighted first-order model counting. In *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

[231] Vijaymeena, M. and Kavitha, K. (2016). A survey on similarity measures in text mining. *Machine Learning and Applications: An International Journal*, 3(2):19–28.

[232] Visengeriyeva, L. and Abedjan, Z. (2018). Metadata-driven error detection. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 1:1–1:12.

[233] Visengeriyeva, L. and Abedjan, Z. (2020). Anatomy of metadata for data curation. *ACM Journal of Data and Information Quality (JDIQ)*.

[234] Visengeriyeva, L., Akbik, A., and Kaul, M. (2016). Improving data quality by leveraging statistical relational learning. In *Proceedings of the 21st International Conference on Information Quality, ICIQ 2016, Ciudad Real, Spain, June 22-23*.

[235] Volkovs, M., Chiang, F., Szlichta, J., and Miller, R. J. (2014). Continuous data cleaning. In *2014 IEEE 30th International Conference on Data Engineering*, pages 244–255.

[236] Wand, Y. and Wang, R. Y. (1996). Anchoring data quality dimensions in ontological foundations. *Communications of the ACM*, 39(11):86–95.

[237] Wang, J., Kraska, T., Franklin, M. J., and Feng, J. (2012). Crowder: Crowdsourcing entity resolution. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, volume 5, pages 1483–1494.

[238] Wang, J. and Tang, N. (2014). Towards dependable data repairing with fixing rules. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 457–468.

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

[240] Wang, X., Dong, X. L., and Meliou, A. (2015). Data x-ray: A diagnostic tool for data errors. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[241] Williams, H. P. (2013). *Model building in mathematical programming*. John Wiley & Sons.

[242] Winkler, W. E. (2002). Methods for record linkage and bayesian networks. Technical report, Statistical Research Division, US Census Bureau.

[243] Wolpert, D. H. (1992). Stacked generalization. *Neural networks*.

[244] Wu, E. and Madden, S. (2013). Scorpion: Explaining away outliers in aggregate queries. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, volume 6, pages 553–564.

[245] Yakout, M., Berti-Équille, L., and Elmagarmid, A. K. (2013). Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 553–564.

[246] Yakout, M., Elmagarmid, A. K., Neville, J., Ouzzani, M., and Ilyas, I. F. (2011). Guided data repair. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, volume 4, pages 279–289.

[247] Yao, L., Riedel, S., and McCallum, A. (2012). Probabilistic databases of universal schema. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*.

[248] Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2001). Generalized belief propagation. In *Advances in neural information processing systems*.

[249] Yuan, Y. C. (2000). Multiple imputation for missing data: Concepts and new development. In *Proceedings of the Twenty-Fifth Annual SAS Users Group International Conference*, volume 267.

[250] Zhang, A., Song, S., and Wang, J. (2016). Sequential data cleaning: a statistical approach. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 909–924.

[251] Zheng, A. and Casari, A. (2018). *Feature engineering for machine learning: principles and techniques for data scientists*. O'Reilly Media, Inc.

[252] Zhou, Z.-H. (2012). *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC.

A

Mapping Between Data Quality Problems and Metadata

Table A.1 provides a mapping between data quality problems and metadata. The set of data quality issues is compiled from the literature, such as Laranjeiro et al. [147], Oliveira et al. [180], and Kim et al. [133]. The mapping between data quality issues and metadata is established by designing error detection heuristics. Two approaches are used to create these heuristics: (1) - A qualitative approach, where existing methods are reviewed; and (2) - A trivial relationship approach, where the connection between data errors and metadata is trivially established (marked as •).

Table A.1: Mapping between data quality problems and metadata.

Data Errors	Metadata	Heuristics to detect data quality issues
Missing data (explicit and implicit - default values)		
	Value length	If min(value length)==0, then ERROR; ▪
	Null values	Cells with null values indicate ERROR; ▪
	Histogram	Check extreme values in the histogram: Disguise missing values with the very large or very small value; [3]

Table A.1 (cont.) Mapping between data quality problems and metadata.

Data Errors	Metadata	Heuristics to detect data quality issues
	Constancy	Check the most frequent value; Can be an ERROR [189]
	Quartiles	Use quartiles for all histograms [17]
	First digit	If first digit == 9, then probably ERROR; (e.g., phone num: 9999-999-999) [3]
	Basic type	Check contradictions for the data type: Presence of strings in the numeric column might indicate some default values like "N/A" [3]
	Data type	Check contradictions for the data type: Presence of strings in the numeric column might indicate some default values like "N/A" [3]
	Size	If the size is 0, then ERROR; ▪
	Patterns (histogram)	Availability of the following patterns indicates disguised values: Strings with repeated substrings (123123123); Strings with repeated characters (e.g. "9999999999") [189]
	Data class	Check against common default values for the given data class (date/time: 1 January) [189]
	Domain	Check against common default values for the given semantic role (state:Alabama) [228, 189]
	Association rules	Identify attribute that violate the rule [6]
	Clustering	Disguised values: Detect values that are far from the rest of the values in the Euclidean space. [116]
	Outliers	Histogram-based outlier detection: Check extreme values in the histogram: Disguise missing values with the very large or very small value; Use quartiles for all histograms as threshold values; [114, 190, 189, 17]
Incorrect Data (value does not conform to the real entity)		

Table A.1 (cont.) Mapping between data quality problems and metadata.

Data Errors	Metadata	Heuristics to detect data quality issues
	Histogram	If a value is in the tail of the values distribution, then ERROR; For the alphabetic data: If one of the n-grams of the value is in the tail of the n-grams frequencies distribution, then ERROR; [114]
	Constancy	If frequency(value) $<$ min(frequency(top 10 values)), then ERROR [114]
	Quartiles	Compute quartiles for histogram + outliers, meaning that quartiles act as a threshold [114, 116, 17]
	First digit	Following Benford's law: P(d)= log(1+1/d); For int data type: if P(d) « P(1), then ERROR [3]
	Basic type	Check generic data type format [3]
	Data type	Check concrete data type format (varchar, int, text) [3]
	Domain	Check the domain values: If the value is out of the domain values, then ERROR; Check the semantic roles (zip code, state): If the value doesn't fit the semantic role, then ERROR; [228]
	Association rules	Identify data items that broke the rules and can be considered outliers (potential errors). [17]
	Outliers	Histogram-based outlier detection: Check extreme values in the histogram: Use quartiles for all histograms as threshold values; [8, 114, 116, 190]
	FD/CFD	FD $L \rightarrow R$ holds for any two rows u and v. suspected violation: $\{u \vert u, v \in \mathcal{D}, u(L) = v(L), u(R) \neq u(R)\}$; [62, 82, 239]
Misspellings		
	Distinct	If the attribute is non-PK, the high number of distinct values can be an indication for ERRORS is the column; ▪
	Uniqueness	If the attribute is non-PK, the higher is the number the more probable that the column contains ERRORS; ▪

Table A.1 (cont.) Mapping between data quality problems and metadata.

Data Errors	Metadata	Heuristics to detect data quality issues
	Histogram	If the value is in the tail of the values distribution, then ERROR; For the alphabetic data: If one of the n-grams of the value is in the tail of the n-grams frequencies distribution, then ERROR; For names: Check soundex code distribution; [3, 114, 190]
	Constancy	If frequency(value) $<$ min(frequency(top 10 values)), then ERROR [114]
	Quartiles	If the value is not in "middle fifty: 3Q-1Q", then ERROR [17]
	Size	If size(value) $>$ max, then ERROR; [195]
	Patterns (histogram)	If the value is in the tail of the value patterns distribution, then ERROR; [3, 114]
	Domain	Check the semantic role (e.g., zip, state) [228]
	Correlations	Ungrammatical/incorrect strings should produce n-gram probabilities that are much smaller than the product of the unigram probabilities (the value of mutual information MI will be negative) and indicate errors [51]
	Association rules	Identify data items that broke the rules and can be considered outliers (potential errors). [17]
	Clustering	Clustering with Levenshtein distance [128]
	Outliers	Histogram-based outlier detection: Check extreme values in the histogram: Use quartiles for all histograms as threshold values; [190]
	Summaries and sketches	For the particular attribute - Each row is considered as a set of values (create n-grams set of each value in the cell). Apply LSH (Locality sensitive hashing) on these sets. One of the two attribute values with the Jaccard similarity $>$ threshold can be considered as misspellings (ERROR) [3]
	Functional Dependencies	FD $L \rightarrow R$ holds for any two rows u and v. suspected violation: $\{u \vert u, v \in \mathcal{D}, u(L) = v(L), u(R) \neq u(R)\}$; [62, 82, 239]

Table A.1 (cont.) Mapping between data quality problems and metadata.

Data Errors	Metadata	Heuristics to detect data quality issues
Ambiguous data (non-interpretable data, abbreviations)		
	Value length	If length(value) $<$ min threshold, then ERROR; [195]
	Histogram	If the value is in the tail of the values distribution, then ERROR; [114]
	Quartiles	If the value is not in "middle fifty: 3Q-1Q", then ERROR [17]
	Basic type	Check data type format: Abbreviations are commonly used in alphabetic data types; [3]
	Size	If size(value) $<$ min threshold, then ERROR; [195]
	Patterns (histogram)	If the value is in the tail of the value pattern distribution, then ERROR; [3]
	Domain	Check the semantic role (zip, state) [228]
	Correlations	Ungrammatical/incorrect strings should produce n-gram probabilities that are much smaller than the product of the unigram probabilities (the value of mutual information MI will be negative) and indicate errors [51]
	Association rules	Identify data items that broke the rules and can be considered outliers (potential errors). [17]
	Outliers	Histogram-based outlier detection: Check extreme values in the histogram: Use quartiles for all histograms as threshold values; [114, 116]
Extraneous data (additional data represented, e.g. title+name)		

Table A.1 (cont.) Mapping between data quality problems and metadata.

Data Errors	Metadata	Heuristics to detect data quality issues
	Value length	If length(value) $>$ max threshold, then ERROR; [195]
	Histogram	Create a histogram of values (or its representation like n-grams, hash codes, soundex code) for outlier detection [114, 116]
	Quartiles	Use as thresholds [17]
	Patterns (histogram)	If the value is in the tail of the value pattern distribution, then ERROR; [114]
	Data class	Check the class format; [3, 128]
	Domain	Check the attribute semantic role (e.g not permitted values) [228]
	Association rules	Identify data items that broke the rules and can be considered outliers (potential errors). [17]
	Outliers	Histogram-based outlier detection: Check extreme values in the histogram: Use quartiles for all histograms as threshold values; [114, 116, 190]
Outdated temporal values		
	Patterns (histogram)	For time series, a pattern is defined as a subsequence of two consecutive points. A pattern p is called an anomaly, if there are very few other patterns with the same slope and the same length. [104, 49]
	Clustering	Clustering data rows according their entity id; Each cluster should be checked against currency order for the specified attribute A_i; (see [82] Ch.6 Example 6.2) [82, 104]
	Outliers	Residuals vs. moving average then HampelX84 [128]
	Functional Dependencies	Using temporal functional dependencies [124]
Misfielded values		

Table A.1 (cont.) Mapping between data quality problems and metadata.

Data Errors	Metadata	Heuristics to detect data quality issues
	null values	If "neighbor"-values, $attr_{i-1}$ or $attr_{i+1}$ are NULL, then $attr_i$ is a potential ERROR; ▪
	value length	If length(value) >max threshold, then ERROR; [195]
	pattern	If the value is in the tail of the value pattern distribution, then ERROR; [114, 116]
	z-value	Check z-value against a threshold; [195, 8]
	quartile	Use quartiles for all histograms as threshold values;[114, 116]
	clustering	Create numerical representations of values from a colum range [a_i-1, a_i, a_i+1],(eg.word2vec); Cluster these vectors; Tuples with "Misfielded values" should be captured within one cluster; [195]
	histogram	If a value is in the tail of the values distribution, then ERROR; For the alphabetic data: If one of the n-grams of the value is in the tail of the n-grams frequencies distribution, then ERROR;[114, 116]
	outliers	Histogram-based outlier detection: Check extreme values in the histogram; Use quartiles for all histograms as threshold values; [114, 116]
	association rule	Identify data items that broke the rules and can be considered outliers (potential errors) [17]
	data type	Check data type format [3]
	domain	Check semantic role (zip, city, state) with regex; [228]
	FD/CFD	FD $L \rightarrow R$ holds for any two rows u and v. suspected violation: $\{u \vert u, v \in \mathcal{D}, u(L) = v(L), u(R) \neq u(R)\}$; [62, 82, 239]

Table A.1 (cont.) Mapping between data quality problems and metadata.

Data Errors	Metadata	Heuristics to detect data quality issues
Incorrect reference (e.g. an employee is associated with the wrong dep.)		
	Correlation	Incorrect values should produce n-gram (chunk is one value) probabilities that are much smaller than the product of the unigram probabilities (the value of mutual information MI will be negative) and indicate errors [51]
	Association rule	Check the association rules; Find the frequent itemsets; If not in the frequent itemset, then ERROR; [3]
Duplicates (when the same data appears multiple times (e.g., two entries for the same client))		
	Distinct	If distinct values $<$ 100% then potential duplicates ▪
	Uniqueness	If uniqueness $<$ 1 then potential duplicates ▪
	Clustering	Clustering based on the attribute values and requires similarity matrix calculation; Clustering based on n-grams, patterns and soundex; Using different similarity measures; [3, 195]
	Unique column combinations	▪
	FD/CFD	FD acting as MD; [82]
	Matching Dependencies	If the attribute is in the MD and MD compliance violation, then ERROR; [82]

Table A.1 (cont.) Mapping between data quality problems and metadata.

Data Errors	Metadata	Heuristics to detect data quality issues
Structural conflicts (duplicate records in different sources)		
	Clustering	Clustering synonyms; [195]
	Summaries and sketches	Each column represents one set. Apply LSH (Locality sensitive hashing) on these sets. One of the two values with the Jaccard similarity > threshold can be considered as near-duplicates (ERROR) [215]
	Matching Dependencies	If MD violation, then ERROR [82]
Wrong word ordering (e.g second name + first name instead first+second)		
	Correlations	Ungrammatical/incorrect strings should produce n-gram probabilities that are much smaller than the product of the unigram probabilities (the value of mutual information MI will be negative) and indicate errors [51]
	Association rules	Identify data items that broke the rules and can be considered outliers (potential errors). [17]
	Clustering	Soundex code clustering -> same code should be in the same cluster; [3, 128]

Table A.1 (cont.) Mapping between data quality problems and metadata.

Data Errors	Metadata	Heuristics to detect data quality issues
	Summaries and sketches	For the particular attribute - Each row is considered as a set of values (create n-grams set of each value in the cell). Apply LSH (Locality sensitive hashing) on these sets. One of the two attribute values with the Jaccard similarity $>$ threshold can be considered as misspellings (ERROR) [215]
Wrong aggregation levels (week vs month. Also relates to diff representation units)		
	Size	Value patterns do not match; E.g. Aggregation over week produces smaller numbers than aggregation over months. ▪
	Patterns (histogram)	Value patterns do not match; E.g. Aggregation over week produces smaller numbers than aggregation over months. ▪
	Clustering	Clustering with euclidean distance (requires computation of similarity matrix) [128]
Temporal mismatch		
	Patterns (histogram)	For time series, a pattern is defined as a subsequence of two consecutive points. A pattern p is called an anomaly, if there are very few other patterns with the same slope and the same length. [104, 49]
	Clustering	Clustering data rows according their entity id; Each cluster should be checked against currency order for the specified attribute A_i; (see [82] Ch.6 Example 6.2) [82, 104]
	Functional Dependencies	Using temporal functional dependencies [124]

Table A.1 (cont.) Mapping between data quality problems and metadata.

Data Errors	Metadata	Heuristics to detect data quality issues
Wrong representations (abbreviations, alia names, encodings)		
	Distinct	Check distinct values for different representation for the same data, for example "Male/Female" -> "M/F" or true/false -> 1/0 [195]
	Patterns (histogram)	Compare distinct patterns to detect ERROR; [3]
	Domain	Check semantic role (zip, city, state, gender) with regex; [228]
Domain violation (illegal values)		
	Distinct	The cardinality of the values should be within the threshold; [195]
	Histogram	If the value is in the tail of the values distribution, then ERROR; For the alphabetic data: If one of the n-grams of the value is in the tail of the n-grams frequencies distribution, then ERROR; [3, 114, 116]
	Data type	Check the concrete data type constraints [195]
	Size	The min/max number of digits in numeric values should be within the datatype range; [195]
	Decimals	Check roundings; [3, 195]
	Patterns (histogram)	If the value is in the tail of the value patterns distribution, then ERROR; [3, 114]
	Data class	Check the data type format; [3]
	Domain	Check the attribute semantic role (e.g not permitted values) [228]

Table A.1 (cont.) Mapping between data quality problems and metadata.

Data Errors	Metadata	Heuristics to detect data quality issues
	Correlations	Ungrammatical/incorrect/illegal strings should produce n-gram probabilities that are much smaller than the product of the unigram probabilities (the value of mutual information MI will be negative) and indicate errors [51]
	Association rules	Identify data items that broke the rules and can be considered outliers (potential errors). [17]
	Outliers	Histogram-based outlier detection: Check extreme values in the histogram; Use quartiles for all histograms as threshold values; [114, 116]
FD Violation		
	Association rule	FD are defined as association rule. Identify data items that broke the rules and can be considered as potential errors [17]
	FD/CFD	FD $L \rightarrow R$ holds for any two rows u and v. suspected violation: $\{u \vert u, v \in \mathcal{D}, u(L) = v(L), u(R) \neq u(R)\}$; [62, 82, 239]
Wrong data type (syntax violation; data type constraint violation)		
	Data type	Check the data type constraint (e.g. int -> contains non-digit characters) [3, 195]
	Data class	Check generic data type constraint; [3]
	Domain	Check the semantic role constraint; [228]
Referential integrity violation (e.g. bank account has no associated client)		

Table A.1 (cont.) Mapping between data quality problems and metadata.

Data Errors	Metadata	Heuristics to detect data quality issues
	Inclusion dependencies	Inclusion constraint violation; [3, 82]
Uniqueness violation (the attribute eg. pass id should be unique and not null)		
	Null values	Null values are not permitted; 0% [3]
	Distinct	Distinct values should be 100% [3]
	Uniqueness	Uniqueness of the attribute should be 1; [3]
	Histogram	Uniform ▪
	Unique column combinations	If violating UCC dependency, then ERROR; [3]
Use of synonyms (two identifiers are mapped to the same concept) e.g. Professor and Teacher		
	Clustering	Clustering data points (rows) will place values with different identifiers into the same cluster (requires computation of the similarity matrix according to the synonymity criteria.) [195]
Use of special characters (e.g.space, no space, dash, parenthesis)		
	Histogram	Create a histogram of used characters: Does long tail contains special characters? [114, 128]
	Data type	Data type format compliance (with regex); [3]

Table A.1 (cont.) Mapping between data quality problems and metadata.

Data Errors	Metadata	Heuristics to detect data quality issues
	Association rules	Identify data items that broke the rules and can be considered outliers (potential errors). [17]
	Clustering	Clustering on structure extraction (patterns) [128]
	Outliers	Histogram-based outlier detection: Check extreme values in the histogram; [114, 116, 190]
Different encoding formats		
	Data type	Data type format compliance [3]
	Domain	Check domain compliance with regex [228]