Restat *et al. Journal of Big Data* (2025) 12:131
https://doi.org/10.1186/s40537-025-01158-x

Journal of Big Data

RESEARCH Open Access

FONDUE—Fine-Tuned Optimization: Nurturing Data Usability & Efficiency

Valerie Restat[1*], Indra Diestelkämper[1], Meike Klettke[2] and Uta Störl[1]

*Correspondence:
valerie.restat@fernuni-hagen.de

[1] Chair of Databases and Information Systems, University of Hagen, Universitätsstr. 1, 58097 Hagen, Germany
[2] Chair for Data Engineering, University of Regensburg, Bajuwarenstraße 4, 93053 Regensburg, Germany

Abstract

To provide good results and decisions in data-driven systems, data quality must be ensured as a primary consideration. An important aspect of this is data cleaning. Although many different algorithms and tools already exist for data cleaning, an end-to-end data quality solution is still needed. In this paper, we present FONDUE, our vision of a well-founded end-to-end data quality optimizer. In contrast to many studies that consider data cleaning in the context of machine learning, our approach focuses on various scenarios, such as when preprocessing and downstream analysis are separated. As an adaptive and easily extendable framework, FONDUE operates similarly to proven methods of database query optimization. Analogously, it consists of the following parts: Rule-based optimization, where the appropriate data cleaning algorithms are selected based on use case constraints, optimizer hints in the form of best practices, and cost-based optimization, where the costs are measured in terms of data quality. Accordingly, the result is an optimized data cleaning pipeline. The choice of different optimization goals enables further flexibility, e.g. for environments with limited resources. As a first building block of FONDUE, we present CheDDaR, which is used to detect errors and measure data quality. Both are important tasks for improving data quality with FONDUE.

Keywords: Data quality, Data cleaning, Optimization

Introduction

Since data are important resources in all organizations, it is of crucial importance to ensure their quality. For this reason, data cleaning is a central aspect of working with data. At the same time, it is a complex and very often time-consuming process that has been intensively studied by the data management community. Since many different types of error can occur in data, it usually requires many different algorithms and tools. Abedjan et al. [1] have investigated the best strategy for a holistic approach to error detection. They conclude that "there is no single dominant data cleaning tool for all data sets and blindly combining the tools will likely decrease precision" [1]. Therefore, they emphasize the importance for an *end-to-end data quality solution* [1].

In many works (e.g. [2–4]), data cleaning is generally considered in the context of machine learning. This is useful if the machine learning model is known and the entire process can be accessed. Nevertheless, data cleaning often takes place separately from

a subsequent analysis [5]. We have experienced this ourselves with applications in the banking sector. Various teams work on the cleaning of the data and the subsequent analysis. People who preprocess data have no knowledge of its subsequent use. Vice versa, people who analyze the data and create machine learning models cannot influence data preprocessing. This requires a data preparation solution independent of the downstream application.

The contribution of this paper is our vision of such a solution as an end-to-end data quality optimizer, called *FONDUE* (Fine-tuned Optimization: Nurturing Data Usability & Efficiency). For this, we propose an adaptable and extensible framework. It consists of a sequence of a rule-based and a cost-based optimization of data cleaning pipelines. In the context of rule-based optimization, the appropriate data cleaning methods are selected depending on the use case. In addition, optional best practices comparable to optimizer hints can be included. Depending on these optimizations, the pipeline that results in the best data quality is selected in the cost-based optimization. Different optimization goals can be set for this, e.g. for environments with limited resources or applications in which the data must be processed near real time. The topic of optimization is important in many different areas of application. In this work, the focus is dedicated to the optimization of data quality. This allows our approach to be applied to use cases in which where the subsequent analysis is unknown.

This article is an extended version of a conference paper with the title *Towards an End-to-End Data Quality Optimizer* [6]. The present version additionally includes the presentation of our newly developed tool *CheDDaR* (Checking Data – Data Quality Review). In the context of FONDUE, it is used to detect initial errors, but also to determine the quality of a data engineering pipeline. Measuring data quality in terms of metrics is an open research question without any standardization yet in existence. We introduce our proposed solution in Sect. "CheDDaR".

The remainder of the paper is structured as follows: First, Sect." State of the Art" provides an overview of the state of the art. FONDUE, our vision of an end-to-end data quality optimizer, including rule-based optimization, best practices, and cost-based optimization is presented in Sect. "FONDUE". Our tool for measuring data quality – CheDDaR – is then introduced in Sect. "CheDDaR". Section "Conclusion and outlook" concludes with a summary and an outlook.

State of the Art

Optimization plays an important role in many different areas of application. For this reason, there are many different optimization goals and methods (e.g. [7–10]). Our aim is to develop a solution that is as general as possible and can be used in as many application scenarios as possible. The focus is currently on structured and hierarchical data that arrives as a sequence of *batches*. We assume that each batch has a *schema*.

For this reason, streaming data and real-time databases are not currently the subject of this work. These types of data pose new challenges [11, 12]. We would like to investigate this in detail in the future. The same applies to time series [13]. Moreover, even though data cleaning systems should of course always operate in a reasonable time, we would first like to describe the general concept in this paper. Intensive performance measurements will be the subject of our future research.

The focus of this work is on the concept of optimizing data quality. Data quality has been the subject of research for more than 30 years [14, 15]. To evaluate data quality, suitable metrics are needed. Existing metrics (such as in [16] and [17]) address only a few aspects of data quality. An evaluation system for data quality called Deequ is described in [18]. It takes into account completeness, consistency and accuracy. By combining quality constraints and user-defined validation code, different assessments are possible. The authors of [19] look at data quality particularly in the big data context. To this end, they propose a framework of 12 metrics. Besides this, there are various testing methods from research and practice that are automated to varying degrees. However, none of them covers all possible error types. Well-known examples of open source tools are pandera,[1] pydantic,[2] and great-expectations.[3] There are also various verification methods in research. Examples are Raha [20] and HoloDetect [21], which are designed to identify data errors with as little manual effort as possible using machine learning. Other approaches, such as [22] and [23], rely on manual evaluation through the involvement of domain experts. In our approach with CheDDaR, flexible use is possible, depending on how much domain knowledge is available or whether ground truth exists. A standardized solution does not yet exist. In [24], we described the theoretical basis of CheDDaR. We have since expanded this concept and implemented it. This is presented in Sect. 4 of this paper.

Similarly, appropriate transformation processes need to be developed to improve data quality. The topic is of great interest in research and practice. For this reason, a range of different data cleaning methods exist. A description of the end-to-end data cleaning process can be found in [25]. Numerous methods for error detection and repair are described there. Analogous to the multitude of methods, there are also many different data cleaning tools. For example, the combination of Raha and Baran represents an end-to-end data cleaning pipeline [26]. Nonetheless, its error classification only covers a part of all possible errors, and it is only suitable for structured data and not as flexible as our approach. Many other tools also exist, but they are either use-case-specific or only deal with one or a few error types. An overview can be found in [27]. However, as described, Abedjan et al. [1] have shown that there is no single dominant data cleaning tool yet. Accordingly, users are challenged to decide on the most suitable method or tool as well as composition for a particular problem [28]. This emphasizes the need for an end-to-end data quality solution.

This need is also discussed in [2] and a vision of a holistic data cleaning framework is presented. While the authors also use optimizers to combine different signals (data cleaning information) and produce a pipeline to detect and repair errors, the extent to which different signals and optimizers can be integrated into a holistic data cleaning approach remains an open research question. Furthermore, their focus lies on machine learning and human interaction. This is also the focus of most other work in this area [2–4, 29]. These approaches use a downstream model or application to optimize cleaning. In contrast, our approach is suitable for applications in which the subsequent analysis is unknown. Hence, we use data quality for optimization. There

[1] https://github.com/unionai-oss/pandera.

[2] https://github.com/pydantic/pydantic.

[3] https://github.com/great-expectations/great_expectations.

is no standard definition of data quality yet, as the authors of [2] have also pointed out. Moreover, in distinction to other work, we provide the ability to define further optimization goals and attempt to reduce human interaction where possible.

FONDUE

In this section we present FONDUE, our vision of an end-to-end data quality optimizer. The proposed process is comparable to well-established methodologies from the database management field of query optimization [30, 31]. The aim of query optimization is to translate a user-submitted query into an effective plan. In FONDUE, the aim is the translation of a concrete user request (data set, available resources, time constraints, etc.) into a data cleaning pipeline that results in the best possible data quality. An overview of FONDUE and the corresponding input and output is shown in Fig. 1.

A *data profile* and an *error profile* are first extracted from the data to be cleaned. A data profile comprises a concrete set of metadata. This is an important source of information about the data [32]. A dedicated tool must be selected for extracting the data profile (e.g. ydata-profiling [33]). The error profile provides information about the data quality. It describes which errors need to be cleaned by the pipeline. In the context of our work, CheDDaR is used to extract the error profile. We first introduced CheDDaR in [24] and explained the theoretical concept. We have since expanded and implemented this concept (see Sect. "CheDDaR"). CheDDaR was developed as a stand-alone data quality evaluation tool, but can also be integrated into FONDUE. This is presented in more detail in Sect. "Application example for FONDUE". The two profiles – data profile and error profile – are used as input for FONDUE.

The subsequent optimization process of FONDUE is based on the field of query optimization as explained. Accordingly, it consists of the steps *rule-based optimization* (Sect. "Rule-based Optimization"), *best practices* (Sect. "Best Practice") and *cost-based optimization* (Sect. "Cost-based Optimization"). As can be seen in Fig. 1, CheDDaR is also used in the context of the cost-based optimization

As described, the output of FONDUE is a data cleaning pipeline that leads to the data quality being maximized. In order to offer a solution that is as general as possible and suitable for a variety of use cases, the output pipeline is described in a technology-independent manner. This is implemented in the form of a *pipeline*

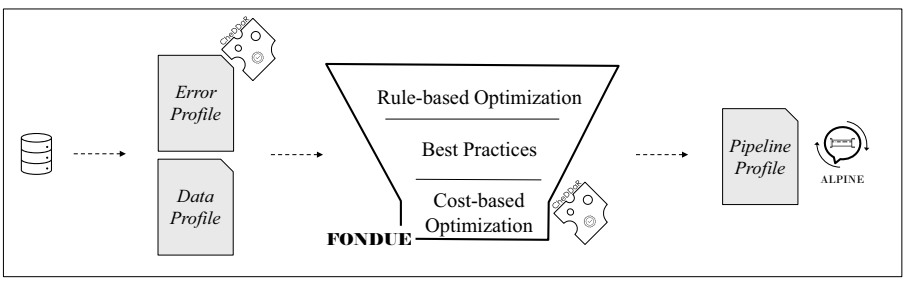


Fig. 1 High-level representation of FONDUE and the associated input/output

profile. ALPINE [34] is used for this purpose. ALPINE is a description language that can be used to abstract from the concrete implementation.

To further address the requirements of different use cases, various optimization goals are possible. In some use cases, there are limitations of hardware. This applies to our smart city projects, for example. Our project partners often reported that cities would only have limited hardware with few resources. In other use cases, processing may need to be as "green" as possible or as close to real time as possible. It is therefore helpful to be able to select different optimization goals.

Overview: Optimization Process

An overview of the optimization process within FONDUE and the steps outlined is shown in Fig. 2. The first step is the *rule-based optimization* [30]). Its goal is to narrow down the initial search space with all possible algorithms. Since different algorithm classes are required and a large number of different algorithms exist for each of these, this initially generates a very large search space that cannot be searched manually. Thus, Part 1 filters algorithms not suitable for the use case (by constraints such as data characteristics or the available resources) and Part 2 filters algorithms which do not affect the order of the pipeline. In a second step, similarly to optimizer hints [35], *best practices* can be included. Subsequently, the next step is the *cost-based optimization* [31]. In this case, costs are measured in terms of data quality. Hence, the best possible pipeline in terms of data quality is selected in the remaining search space.

The components are built step-by-step and modularly expandable. Each of these steps will be described below. Figure 3 shows how the search space is successively reduced.

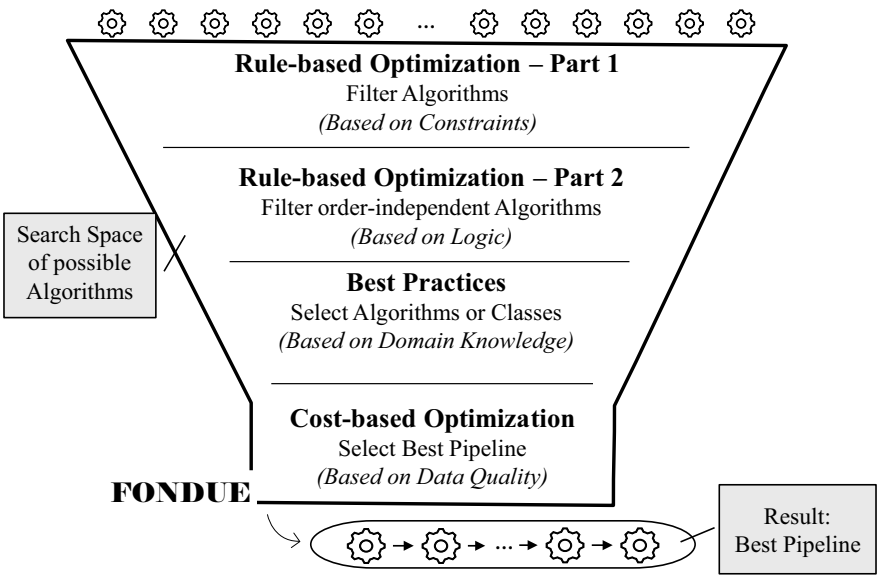
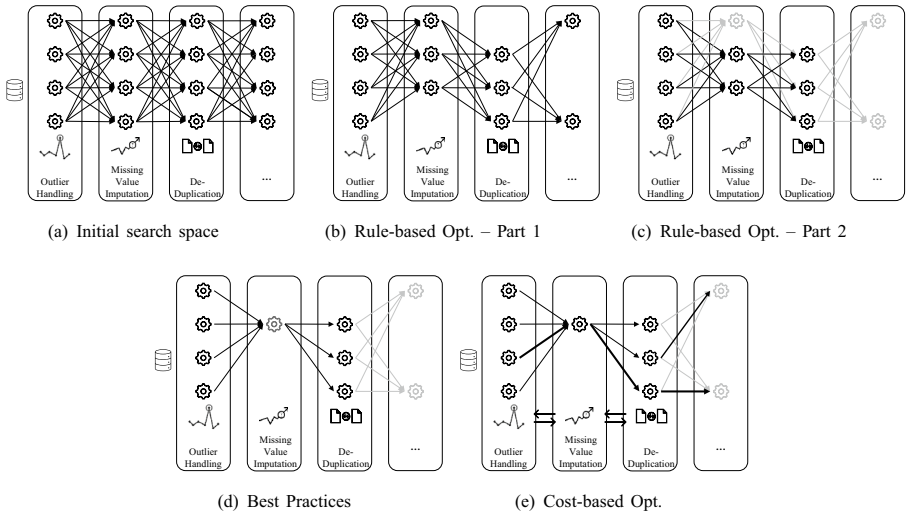


Fig. 2 Overview of the optimization process within FONDUE, from the initial large search space of possible algorithms to the pipeline with the best possible data quality



(a) Initial search space (b) Rule-based Opt. – Part 1 (c) Rule-based Opt. – Part 2

(d) Best Practices (e) Cost-based Opt.

Fig. 3 Optimization procedure of FONDUE: The initial search space (**a**) is reduced step by step by rule-based optimization part 1 (**b**) and part 2 (**c**) as well as best practices (**d**) and subsequently the different possible pipelines are evaluated in terms of data quality by the cost-based optimization (**e**)

Notation

Before explaining the process, we introduce the notation used. In the following, a distinction is made between *algorithms* and *algorithm classes*. An algorithm class is represented by separate slices in Fig. 3. For each type of error, there is an algorithm class. An example of an algorithm class would be missing value imputation. An algorithm is a concrete method for repairing the corresponding error type. An example would be missing value imputation using mean imputation. In Fig. 3, algorithms are represented by gears.

Rule-based Optimization

The selection of the most suitable data cleaning pipeline begins with rule-based optimization. The search space of possible algorithms is reduced by constraints and rules.

Filtering Algorithms by Constraints

In the first step, algorithms that do not suit the use case constraints, e.g. the data, are filtered out. Coming back to the example of missing value imputation: If a column consists of string values, a mean imputation is not suitably applicable. Another example would be the influence of missing rates. In [36] it was shown that the K-nearest Neighbors Method leads to significantly worse results compared to other algorithms once the missing rate increases. A rule could be created for this case. Based on this, it can be concluded that this method should not be used if the missing rate is high. So, these algorithms can be removed from the search space. Figure 3 (b) shows the result of this step. Individual algorithms (= gears) have been removed.

Research challenges: To perform this step, future research needs to investigate which algorithms – depending on the input data and the optimization goals – are suitable and which are not. As a first step, our current research focuses on missing value imputation.

Our initial analyses show that many different aspects have an impact, such as size, dimensionality, data type, variability, distribution, associations, missing mechanism, missing rate, and missing pattern. Detailed data profiles are required to extract the characteristics of the data. We are currently working on a solution with which the required metadata can be extracted. Another challenge at this point is the efficient linking of data profiles and constraints. This step is particularly time-consuming with high-dimensional and heterogeneous data. An efficient solution that allows fast filtering is thus of great importance.

Filtering order-independent Algorithms

The next step is to filter out those algorithms or classes which are not relevant for the pipeline order. These are still part of the pipeline. They can be executed at any time, since they do not play any role in the order of the pipeline.

First-order Logic

Some data engineering algorithms generate the same result regardless of the order of the algorithms in the pipeline. For example, if we have one algorithm that contains a projection (e.g., to shape the data set) and another that replaces the values of an attribute with other values (e.g., to convert them to a different physical unit). In that case, the order of execution of both algorithms is arbitrary. We can generalize this to data engineering algorithms that can be represented in first-order logic. Unfortunately, only a few real-world algorithms fall into this class. In addition to the projection mentioned before, selection (e.g., a sample selection in data engineering) and type cast (e.g., converting integer to double) belong to this class as well. Other data cleaning algorithms are more complex and belong to the next class.

Second-order Logic, if overlap-free

The next class is second-order logic. Execution of algorithms that belong to this class is not arbitrary. For example, if you use a method to impute missing values and the method is based on clustering the entire tuples, it will produce different results if an algorithm has modified the values of a column before it. Hence, it means that the order is important here. But even in this class of algorithms, you can find cases where you can swap the order of the algorithms without affecting the result. To identify these algorithms whose execution can occur at any point in the pipeline, it is necessary to check which part of the data is accessed by the corresponding algorithm. For example, if an algorithm is cleaning a column that is not used by any other algorithm, this cleaning is *overlap-free* and thus not relevant for the order of the pipeline.

Figure 3 (c) shows the result of this step. Individual algorithms and classes are greyed out. As noted, these must still be performed, although they do not play a role in the choice of the execution order.

Research challenges: To perform this step, future research needs to analyze which algorithms belong to first-order logic and which to second-order logic. Even though it is likely that only very few algorithms belong to first-order logic and most algorithms are not free of overlap, it is still worth investigating these cases to possibly minimize the search space. Thus, it needs to be analyzed which algorithms are free of overlaps.

For error detection, we have already classified different error levels [37]. In our current research, we investigate the same for the repair of errors. We further investigate which

special features arise when working with semi-structured data. This also includes the analysis of error types of structured data that may also occur in semi-structured data and which new error types emanate. When working with semi-structured data, greater schema flexibility is allowed. As a result – in addition to errors in the data itself – structural errors can also occur. An example of this would be missing attributes.

Best Practice

After the rule-based optimization, only the applicable algorithms relevant for the order of the pipeline are left. To further restrict the search space, additional best practices can be applied. In the analogy to query optimization, this step corresponds with optimizer hints. These are used to guide the optimizer into the right direction [35]. Such hints can be set separately for each individual query [35]. Similarly, in our vision, depending on the use case domain experts can contribute their knowledge. For example, if they know that a certain algorithm will produce the best results for the data, or that certain algorithms already proved to work together effectively.

This is comparable to the *explore* and *exploit* principle known in the AutoML area [38]. Here, it is possible to balance exploring (evaluating as many hyperparameters as possible) and exploiting (allocating more resources to promising hyperparameters). With our proposed best practices, it is possible to balance between focusing on promising data cleaning methods (exploit) and searching the entire search space (explore). This is enhanced by different optimization goals.

In contrast to rule-based optimization, in this step algorithms can be selected specifically for inclusion in the final pipeline. Additionally, an explicit order of algorithms for the pipeline can be specified in this step. To continue with the example of missing value imputation: Hasan et al. [39] state that it is recommended to standardize the data first when using distance-based algorithms (such as missing value imputation with K-nearest Neighbors). They also state that when using regression-type algorithms, outlier rejection must first take place. Median-based algorithms, on the other hand, are robust to outliers, since they calculate the missing values from the most frequent values, which must not be outliers [39].

Figure 3 (d) shows the result of this step. An algorithm is marked in red here. This was explicitly selected by the best practices. As a result, the other algorithms in this algorithm class are not considered further.

Research challenges: In addition to recommendations for the explicit use of algorithms – depending on the use case – research related to the ordering of algorithms is particularly relevant in this area. The size of the search space is largely influenced by the number of permutations. When more best practices exist for ordering, it will lead to fewer permutations and thus minimize the search space. Doing so will notably improve the efficiency of the cost-based optimization.

Cost-based Optimization

After the search space has been narrowed down further according to best practices, the order of all algorithms is determined. As in database systems [40], different levels of optimization can be set. Depending on the search space and the best practices given, the optimization may vary regarding compilation time and the quality of the

pipeline generated. For example, if not enough time and resources are available, the search space can be limited by best practices. If sufficient time and resources are provided, algorithms not specified by best practices can still be considered for cost-based optimization.

Depending on the remaining size of the search space, all permutations are formed or, in case too many combinations remain, a Monte Carlo simulation [41, 42] is conceivable. The result is a multitude of different pipelines. It should be noted that these pipelines also include those algorithms that are irrelevant to the execution order. For each possible pipeline, the cost – that is, the data quality – are measured and evaluated. This step is demonstrated graphically in Fig. 3 (e).

Finally, the pipeline that achieves the highest data quality is selected. For this step, a framework of metrics is needed to assess data quality. We have already developed a framework called CheDDaR (see Sect. "CheDDaR") to measure data quality and evaluate data cleaning algorithms. This will be applied for the cost-based optimization. The advantage of CheDDaR is that it takes many different aspects of data quality into account. The metrics correspond to the extensive error classification that we have performed in [37]. The fewer errors in the data, the better the data quality. Furthermore, CheDDaR can be used flexibly, depending on the extent to which ground truth is available or if domain experts are available for verification. It is thus well suited for evaluating the different pipelines and allows to select the pipeline that achieves the highest data quality.

Research challenges: Evaluating data quality is hardly possible without manual effort [24]. Nevertheless, domain experts should be supported more effectively and human interaction minimized. This is what we want to achieve with CheDDaR. We also plan to incorporate the domain knowledge that is generated when using CheDDaR directly into the optimization as best practices to further reduce the search space.

Another challenge is the selection of the appropriate algorithm for optimization. The size of the search space depends on the dimensionality of the data. The more attributes a data set has, the more permutations are possible. The different optimization goals bring about further variation. Depending on the desired execution time, energy consumption, or CO_2 emissions, the optimization must be designed differently. In addition to the possible use of Monte Carlo simulation, among others, we want to examine the suitability of nature-inspired algorithms, which are popular for many optimization problems [43]. For example, we are currently analyzing whether and how such metaheuristic techniques can be used for cost-based optimization. Here we are examining methods that arrange possible pipeline candidates in form of a matrix and generate new candidates by recombining existing ones. Examples include the genetic algorithm [44] and harmony search [7]. Another group of methods that we are currently investigating are those in which one or more agents create new candidate solutions by moving through continuous space. Examples include simulated annealing [45] and particle swarm optimization [10]. The main challenge here is to find a suitable representation of the search space and the pipeline candidates it contains.

Due to the analogy to query optimization, we also want to analyze whether methods analogous to learning query optimizers such as LEO [46] are suitable. This compares the estimated costs with the actual costs regularly during the execution of the current query and can change the query plan before the end of execution if there is a large discrepancy. A learning component is also provided here, which is intended to improve cost estimates based on past experience. However, the lack of sufficient training data sets poses particular challenges.

Further Improvements

In addition to the described procedure, further improvements are conceivable:

Human in the Loop

Data cleaning essentially focuses on humans [2]. This is taken into account in our data quality framework CheDDaR [24], which distinguishes between different impacts of domain knowledge. It is nonetheless also conceivable to involve the user even more. Although the envisioned optimizer aims to minimize human involvement, it could be beneficial to offer an optional possibility for manual adaption. One approach would be an interactive dashboard, for example, in the form of a WebUI. Here, users, primarily domain experts with insider knowledge of the data, could intervene and manually adjust the pipeline. The proposed incorporation of best practices, where domain experts can provide input, already enables such an interactive human-in-the-loop approach.

Fairness

Fairness should be considered as early as possible in the pipeline to avoid a potential bias. For this, it is conceivable that tools like *mlinspect* [47] are integrated into the pipeline. Mlinspect is a library to assist in bias detection, as this cannot be fully automated [47].

Robustness of Pipelines

Another challenge for data engineering pipelines is the constant change of data and their schema. Algorithms vary in their robustness to such changes. For example, a missing value imputation where a location is imputed depending on the zip code may no longer work if the attribute zip code is renamed by a structural change. In contrast, deduplication is robust to this change. In our current research, we purposefully perform a comprehensive analysis of the robustness of data engineering pipelines depending on different influencing factors. We investigate structural as well as semantic changes. If we know which algorithms are robust to what kind of change, we can determine the impact on the pipeline. In addition, we know which algorithms need to be adapted because they may no longer work after changes.

CheDDaR

In [24] we presented the concept of CheDDaR. This section describes the practical extension and implementation of the concept presented. CheDDaR originally has been developed as a stand-alone tool for evaluating data quality, but can also be integrated as an important component in FONDUE. In this context, it serves two purposes: It detects data errors and measures data quality. Both tasks are crucial to improve the data quality with FONDUE. CheDDaR aims at detecting data errors early and with little effort. For

that purpose, it relies on the examination of data with various data quality metrics, i.e., metrics that can quantify data quality. A distinguishing feature of CheDDaR is that it offers multiple verification methods for deriving and checking the metrics to keep manual effort low and, at the same time, maximize the scope of testing.

The metrics that are currently implemented in CheDDaR are presented in Sect. "Metrics" below. The precise definition of the metrics is of central importance for the practical implementation of CheDDaR, as it lays the theoretical foundation for it. Section "Verification Methods" then describes the different verification methods and indicates which of the metrics can be verified using which method. This is then illustrated using an example. To this end, the architecture of CheDDaR is described in Sect. "Architecture" and an exemplary application is shown in Sect. "Application Example". In a concluding Sect. "Application Example for FONDUE", the use of CheDDaR as a building block within FONDUE is discussed.

Metrics

In this stage of the CheDDaR implementation, it supports ten widely applicable data quality metrics that are briefly introduced and formally defined in this section.

The metrics definitions rely on Relational Algebra as defined in [48]. In brief, R represents a relation, A_i an attribute of type T, σ a selection and π a projection.

Missing Values

The metric *Missing Values* describes how many values are missing in an attribute A_i of the relation R and is defined as follows:

$$MV(R, A_i) := \frac{|\sigma_{A_i \ IS \ NULL}(R)|}{|R|} \tag{1}$$

The metric MV in the above equation is the quotient of the number of missing values, i.e., the null values in attribute A_i, and the number of tuples in R. The null values are calculated from a projection $\sigma_{A_i \ IS \ NULL}$ on R. Generally, lower values for this metric are better, as they typically imply a more complete data set. Further rules follow the same reading.

For instance, this metric may serve as an indicator of the quality of an analysis or predictive model built on top the data under observation. For, missing values can lead to distortions and reduce the accuracy of predictive models.

Syntax Violation

The *Syntax Violation* metric SV yields the proportion of values in an attribute A_i of R, that violate the syntax rule S. It is defined as follows:

$$SV(R, A_i, S) := \frac{|\sigma_{\neg satisfied(A_i,S)}(R)|}{|R|} \tag{2}$$

Using an appropriate syntax rule S, the above metric indicates data errors or formatting issues. Adhering to the correct syntax is crucial for the accuracy and consistency of data.

This is especially important for time and date information, where regional formatting differences can often lead to syntax violations.

Interval Violation

The *Interval Violation* metric checks whether the numerical values of an attribute A_i in a relation R ranges within the expected interval $[a_{min}, a_{max}]$:

$$IV(R, A_i, a_{min}, a_{max}) := \frac{|\sigma_{A_i < a_{min} \vee A_i > a_{max}}(R)|}{|R|} \tag{3}$$

Interval violations potentially indicate data errors or unusual patterns. Strict adherence to intervals is crucial to ensure that the data conforms to expected norms. Therefore, some flexibility in defining the interval is advisable.

Set violation

The *Set Violation* metric *SetV* checks whether the values of a specific attribute A_i in a relation R are within a predefined set of values S:

$$SetV(R, A_i, S) := \frac{|\sigma_{A_i \notin S}(R)|}{|R|} \tag{4}$$

The *Set Violation* metric is particularly useful for attributes that describe categories. It can indicate when certain values do not match the expected categories, which is important for the consistency and accuracy of the data.

Wrong Data Type

The *Wrong Data Type* metric identfies all tuples in R whose attribute A_i does not have the expected data type T:

$$WDT(R, A_i, T) := \frac{|\sigma_{TypeOf(A_i) \neq T}(R)|}{|R|} \tag{5}$$

This metric may indicate data loss during a data type conversion earlier in the data processing pipeline.

Uniqueness Value Violation

The *Uniqueness Value Violation* metric *UVV* checks whether the values of all tuples in R are unique within the attribute A_i. This metric is designed not only to detect the presence of a uniqueness violation but also to quantify how many tuples violate the uniqueness in A_i relative to the total number of tuples in R:

$$UVV(R, A_i) := \frac{|R| - |\pi_{A_i}(R)|}{|R|} \tag{6}$$

For example, this metric is of help in identifying key candidates in relations or uncovering duplicate entries in time series data.

Outlier

The *Outlier* metric identifies patterns in data that fall outside the value range determining expected behavior. It is defined using a function *isOutlier* applied to an attribute A_i in the relation R:

$$Outlier(R, IsOutlier, A_i) = \frac{|\sigma_{IsOutlier(A_i)}(R)|}{|R|} \tag{7}$$

Note that multiple approaches exist to define the outlier function *isOutlier* [49, 50]. *CheDDaR* currently employs the interquartile range method to detect outliers due to its simplicity. However, it can be extended to support further methods.

This metric is often used to detect inaccuracies or errors in measurement instruments, for example.

Missing Attribute

The *Missing Attribute* metric identifies a missing attribute A in R:

$$MA(R, A) := |\pi_A(R)| = \emptyset \tag{8}$$

Because this metric indicates that an attribute is missing from the schema, it can be particularly useful when data undergoes various transformations. This metric helps to ensure that a transformation does not accidentally remove a necessary attribute.

Additional Attribute

The *Additional Attribute* metric detects an extra attribute A_i in a relation R. It can be understood as the counterpart to the *Missing Attribute* metric and is defined as follows:

$$AA(R, A_i) := |\pi_{A_i}(R)| \neq \emptyset \tag{9}$$

An additional attribute describes an attribute that is not expected in the data. In such a case, it must be checked whether there is a change in the data that needs adjusting or whether an error has occurred.

Zero Variance

The *Zero Variance* metric checks whether an attribute A_i in a relation R shows no variation among its values:

$$ZV(R, A_i) := |\pi_{A_i}(R)| = 1 \tag{10}$$

This metric clearly identifies attributes holding a single value in all instances. Since these attributes lack variance, they typically offer little informational value.

In summary, these metrics identify a variety of data errors that FONDUE aims to repair. As part of CheDDaR these metrics can be extended to identify further data errors of different types.

CheDDaR employs these metrics in the context of one or multiple verification methods that are introduced next.

Verification Methods

Verification methods are different approaches to apply and compute the previously introduced metrics in CheDDaR and, hence, to find data errors.

There is not a single verification method that efficiently and effectively identifies all data errors leveraging data quality metrics [1]. Some of the introduced metrics require different input parameters such as interval boundaries, outlier detection methods or data types. Furthermore, data errors may require tedious manual work to be identified. Therefore, CheDDaR comes with support of five different methods to systematically, efficiently and effectively reveal all kinds of data errors. It can thus be used flexibly, depending on how much domain knowledge is available.

The five methods are shown in Fig. 4. They are typically named after the information sources used to derive quality metrics in order to detect data errors. In the following, we briefly introduce the internals of each method and provide a mapping between CheDDaR's data quality metrics and verification methods.

Verification with Ground Truth

In the context of this paper, *Ground Truth* is considered a reference data set that has the same size and an identical data schema as the data set under observation.

Verification with the *Ground Truth* compares individual data values from both data sets one by one. Hence, this method identifies all data errors reliably, is computationally feasable and does not even need to rely on data quality metrics as introduced in the previous section.

This method is applicable in multiple real-world applications. For example, it can be used to detect targeted data manipulations. If the data set under observation has been manipulated and a backup exists, the unmanipulated backup can serve as *Ground Truth* to reveal the intent behind the manipulations. In another scenario, data is transmitted but the transmission is error-prone. Comparing the received data set with the original data set can identify these errors and help improve transmission quality in the future.

While this verification precisely pinpoints data errors it is only applicable when a *Ground Truth* exists. In practice, many data sets with data errors exist that do not come with *Ground Truth*. For these data sets, CheDDaR offers the following verification methods.

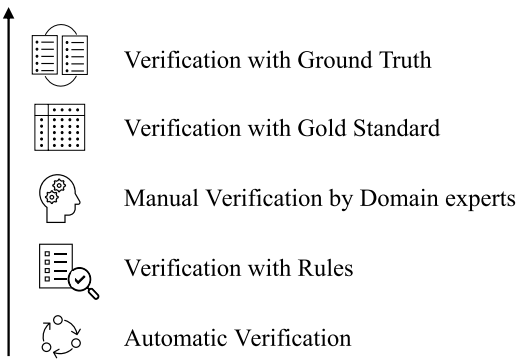


Fig. 4 Verification methods (expanded based on [24])

Verification with Gold Standard

Like the *Ground Truth*, the *Gold Standard* is a reference data set used to determine quality metrics. It is compared with the (original) data set under observation. Unlike the *Ground Truth*, the *Gold Standard* does not necessarily need to have the same size as the data set under observation and may have fewer or more attributes in its data schema. The data types can also differ.

Creating a *Gold Standard* requires a high level of domain knowledge and, like *Ground Truth*, is typically only achievable with the involvement of experts.

The *Gold Standard* can be effectively applied in various scenarios. Today, a data set easily contains billions of entries. A human cannot feasibly capture, identify, assess, and correct data errors without technical assistance. However, for a subset of data with a few hundred or thousand entries, a human can effectively curate the entries. Once the subset is prepared, it can serve as a *Gold Standard* for the full data set to identify data errors systematically.

Data today are characterized not only by their volume but also by the speed at which they change, with new entries or attributes being added quickly. Comparing the current data set with an older version that has been prepared as a *Gold Standard* can reveal systematic errors in the newly added data.

These examples demonstrate that the *Gold Standard* is particularly suitable for uncovering systematic errors that can be traced from a reduced data set to a complete data set.

Not every data set is so large or changes so rapidly that it becomes unmanageable for a human. In such cases, the next verification method may be valuable.

Manual Verification by Domain Experts

There is the option to manually examine the data. Domain experts have the necessary expertise and knowledge to analyze and assess data quality [24].

While this verification method has the advantage of being very thorough and in-depth, it is also labor-intensive and time-consuming due to its manual nature.

It is particularly suitable when data errors could have serious consequences, such as when human lives are at stake. This is not only the case in medicine but can also occur under extreme conditions, such as deep-sea or space missions.

It is also appropriate for targeted investigations of specific problems. Even if a data set is large, targeted investigation of a particular error can be efficiently carried out by experts.

This verification method is very demanding but of high quality. It is suitable when the stakes are high or when the data problem is well-defined and manageable. In cases where this is not applicable, the following verification method may be helpful.

Verification with Rules

A domain expert can specify rules to verify data quality. These rules are generally feasible for humans to create and are applied automatically to the data set under observation to identify data errors. Depending on the domain, general rules or knowledge bases may already exist [24].

Table 1 Implementation of Metrics and Verification Methods in CheDDaR

Metric	Gold Standard	Manual verification	Rules	Automatic verification
Missing Values	✗	✗	✗	✗
Syntax Violation	–	✗	✗	–
Interval Violation	✗	✗	✗	–
Set Violation	✗	✗	✗	–
Wrong Data Type	✗	✗	✗	–
Uniqueness Value Violation	✗	✗	✗	–
Outlier	–	✗	✗	✗
Missing Attribute	✗	✗	✗	–
Additional Attribute	✗	✗	✗	–
Zero Variance	–	✗	✗	✗

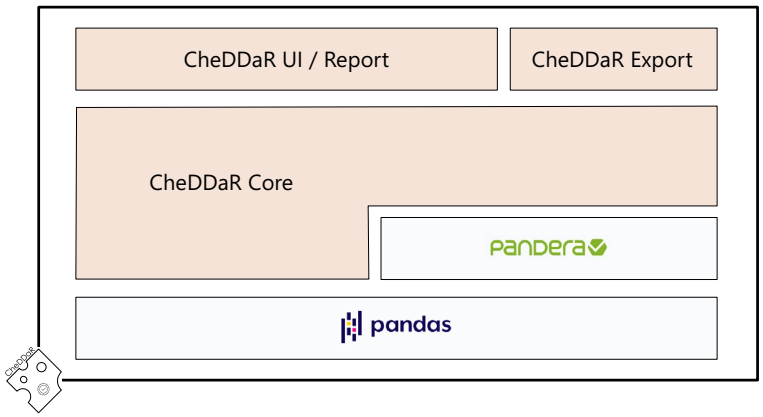


Fig. 5 Overview of CheDDaR's architecture

This method complements and speeds up purely manual verification by encapsulating quality metrics into rules that uncover data errors. For example, with this verification, an expert can easily detect violations of physical laws in faulty data. It also allows for systematic checking of domain-specific value ranges or data relationships. Overall, rules are a good choice when a domain expert wants to verify general, domain-specific data properties or their violations.

If the properties are not domain-specific, the final relevant verification method for this work may be useful.

Automatic Verification

There are several data errors that can be checked with quality metrics that do not require domain expertise or a reference data. Such metrics can be automatically applied to the data set under observation.

This verification method is particularly useful when the user lacks the necessary domain expertise or when the data are new and their properties are unknown. For example, automatic verification can identify missing values or outliers.

Thus, this verification method is especially helpful for exploring an unknown data set or when neither reference data nor domain expertise is available.

Table 1 provides an overview of the verification methods and the metrics implemented for each method. The rows represent the metrics, while the columns represent the verification methods. Crosses indicate that the metrics have been implemented for the corresponding method, or can be implemented for manual and rule-based verification. The minus signs indicate that these metrics could not be implemented due to limitations in the corresponding verification methods. The content of the table will be explained in more detail with each method. *Ground Truth* is not included in the table because it does not check values but rather the equality of individual attribute values.

Architecture

In the following, the architecture of CheDDaR and the libraries used are described. Figure 5 shows an overview.

CheDDaR leverages *pandas*, a common python library for data analysis and manipulation.[4] With a variety of functions and methods, pandas enables the efficient filtering, aggregation and transformation of data. Seamless integration with other python libraries such as *NumPy*, *scikit-learn* and *matplotlib* further extends the functionality of pandas. In addition, *pandera* is also based on the data structures of pandas.

Pandera is the next building block of CheDDaR. It is a python library that enables rule-based data validation for DataFrames (pandas and Spark).[5] It offers a declarative API for validating data, which makes it possible to define schemas for DataFrames and set rules for the attributes in the schema [51]. However, CheDDaR strives for a holistic approach that combines a variety of approaches in order to perform a comprehensive data quality analysis and using the optimal method in each case. This is not covered by the functional scope of pandera.

This is where the *CheDDaR Core* takes effect. It is the most important part of the CheDDaR framework and is responsible for several functionalities that are essential for data quality analysis: It enables the availability of several verification methods simultaneously and assigns the correct data types to the columns for the analysis. It also enables the creation of new data quality metrics and the adjustment of validity ranges for existing metrics. In addition, the metrics are assigned to the corresponding verification methods in the Core.

Based on this, there are two different analysis tools: *CheDDaR UI / Report* and *CheDDaR Export*. The CheDDaR UI / Report fulfills two important tasks. As mentioned, CheDDaR must offer the possibility to perform a manual data analysis with the desired metrics. In addition, it is important to summarize the results obtained using different verification methods and present them in a common report. Jupyter Notebooks[6] are used for this purpose. Ipywidgets[7] further simplify interactions for users.

Reports embedded in Jupyter Notebooks are useful to get a quick overview of the results. However, it can be difficult and cumbersome to compare the results from the validation of different data sets. For some use cases, it makes more sense to not only display the analysis results as an interactive report in a Jupyter Notebook, but to save

[4] https://pandas.pydata.org/.

[5] https://pandera.readthedocs.io/en/stable/index.html.

[6] https://jupyter.org/.

[7] https://pypi.org/project/ipywidgets/.

Table 2 Exemplary *cheese data set*, errors are marked in orange

ID	CheeseName	CheeseType	CheeseAge	PricePerKilo
1	Orkney	Blue	8	40.41
2	Mozzarella		20	
3	Pecorino	Flavoured	68	64.31
4	Brie	Grated	623	
5	Emmental		54	27.70
6	Camembert	Semi-soft	85	9.32
7	Livarot	Semi-hard	93	412.42
8	Raclette		517	34.80
9	Kasseri	Grated	89	
10	Queso de Bola	Smoked	23	49.32

Attribute	Metric	Level	Amount	Percent	Potential outliers
CheeseAge	Outlier	Level 2	2	20.0	[517, 623]
CheeseName					
CheeseType	Missing Values	Level 1	3	30.0	
PricePerKilo	Missing Values	Level 1	3	30.0	
	Outlier	Level 2	1	10.0	[412.42]

Fig. 6 CheDDaR UI / Report for *cheese data set*

the results in a serialized file format and then compare them with each other. For this reason, CheDDaR Export allows the report to be saved as a JSON file. This makes it easier to extract and compare the results in order to draw well-founded conclusions. In the context of FONDUE, which aims to make the application as automated as possible, Export and the JSON files generated in the process are used as the error profile.

Application Example

To illustrate the functionality of CheDDaR, automatic verification is shown below for a small data set (shown in Table 2) as an example. It contains information on various cheeses and comprises five attributes: `ID`, `CheeseName`, `CheeseType`, `CheeseAge`, `PricePerKilo` – which indicate the id, name, type, age and price per kilo of the cheese. The following errors can be found in the data (marked in orange in Table 2):

- `CheeseType`: 3x *Missing Values*
- `CheeseAge`: 2x *Outlier* (`623` and `517`)
- `PricePerKilo`: 3x *Missing Values* and 1x *Outlier* (`412.42`)

The automatic verification of CheDDaR is now illustrated with this exemplary data set. As described in Sect. "Verification Methods", neither domain expertise nor reference data is required for this verification method. Therefore, only the data set is used for evaluation and no further information is needed. Two different outputs are possible, as specified in Sect. "Architecture": CheDDaR UI / Report, which presents the results in a Jupyter report and CheDDaR Export, which saves the results in a JSON report.

Figure 6 shows the output of the CheDDaR UI / Report. This includes the following information:

- *Attribute* The respective attribute
- *Metric* The error type
- *Level* The level of the error type, as classified in [37]
- *Amount* The number of the respective errors
- *Percent* The percentage to which this amount corresponds in the data set
- *Potential outliers* Specifically for the error type *outlier*, the potential candidates are also output

As can be seen, all errors were detected correctly.

In addition, the results generated by CheDDaR Export are shown in Listing 1. For a better overview, only the excerpt for the `CheeseType` attribute is shown. The information from the CheDDaR UI / Report can also be found here.

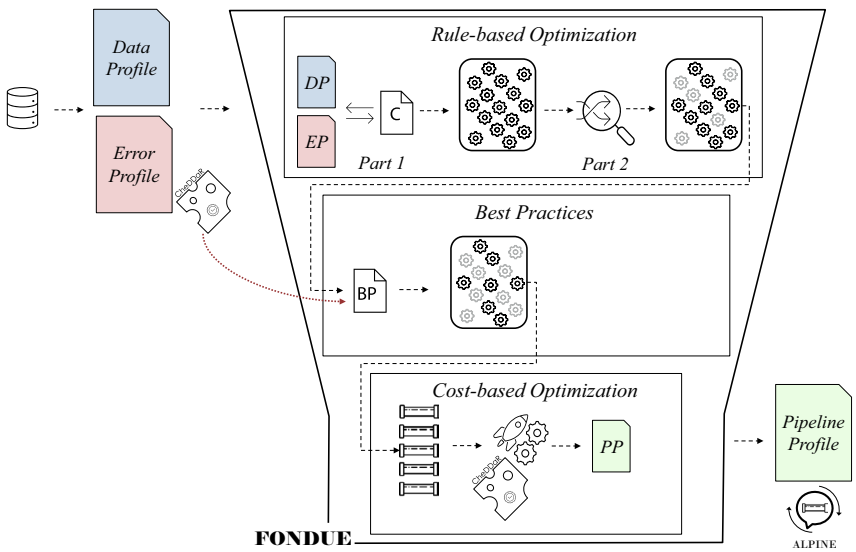


Fig. 7 Detailed process of FONDUE and the associated input/output to maximize data quality EP = Error Profile, DP = Data Profile, PP = Pipeline Profile, C = Constraints, BP = Best Practices

Table 3 Algorithms considered in the application example per algorithm class

Missing Value Imputation	Outlier Handling
(1) MVI_{mean}: Replacing by mean	(1) OH_{mean}: Replacing by mean
(2) MVI_{median}: Replacing by median	(2) OH_{median}: Replacing by median
(3) MVI_{mode}: Replacing by mode	(3) OH_0: Replacing by 0
(4) MVI_0: Replacing by 0	(4) OH_{delete}: Delete row
(5) MVI_U: Replacing by "Unknown"	
(6) MVI_{delete}: Delete row	

```
{
    "properties": {
        "CheeseType": {
            "checks": {
                "Missing Values": {
                    "level": "Level 1",
                    "amount": 3,
                    "percent": 30.0,
                    "indices": [
                        2,
                        5,
                        8
                    ]
                }
            }
        }, ...
    }
}
```

Listing 1 JSON report for cheese data set – for a better overview, only the excerpt for CheeseType is shown. This can be used as an error profile in the context of FONDUE

To conclude, all important information can be seen at a glance in the CheDDaR UI/Report. For example, a domain expert could check the potential outliers and decide whether this is actually an error or whether it is in fact a very expensive cheese, for example. In addition, a JSON report generated by CheDDaR Export is available. For example, the results can be compared with those of another data set or with those of the same data set after an error correction. This flexible handling makes CheDDaR suitable for a wide variety of applications.

Application Example for FONDUE

This section illustrates the application of the concepts presented and the integration of FONDUE and CheDDaR using a simple example. Figure 7 shows a detailed representation of the high-level representation from Fig. 1, which was presented at the beginning in Sect. "FONDUE". It shows the entire process, starting with the data to be cleaned as input in FONDUE followed by the optimization and ending with the output of the pipeline in the form of a pipeline profile. The use of CheDDaR in the context of FONDUE is also highlighted here. The overall process and the individual profiles and steps are illustrated below using a simple application example. The *cheese data set* from Sect. "Application Example" (see Table 2) is used for this purpose.

Data Cleaning Pipeline Operators

As described in Sect. "FONDUE", a pipeline consists of various operators. For the pipeline of the application example, the following operators are required:

- O_{mvi}^{ct}: Handling of missing values in `CheeseType`
- O_{oh}^{ca}: Handling of outliers in `CheeseAge`
- O_{mvi}^{ppk}: Handling of missing values in `PricePerKilo`
- O_{oh}^{ppk}: Handling of outliers in `PricePerKilo`

A large number of algorithms of the respective algorithm class are available for each of these operators. For a better understanding, only a selection of simpler algorithms are considered for the application example. These are shown in Table 3.

This represents the initial search space. There are hence six different algorithms to choose from for O_{mvi}^{ct} and O_{mvi}^{ppk} and four each for O_{oh}^{ca} and O_{oh}^{ppk}. This already leads to $6 * 4 * 6 * 4 = 576$ different possible pipelines. Now these operators can also be executed in different orders, which would lead to different outcomes (e.g. outlier handling can influence missing value imputation with mean values). For the four operators, this results in $4! = 24$ different possible combinations. The initial search space consequently consists of $576 * 24 = 13,824$ possible pipelines.

Naive Approaches

The first naive approach would be to try out all possibilities and evaluate what leads to the best data quality. However, as can be seen from the minimal example, the search space quickly becomes very large. It therefore quickly becomes unrealistic to evaluate all possibilities.

The second naive approach would thus be to randomly select operators and combine them in a random order. We have illustrated this for the application example. The following operators were selected in the subsequent order:

1. O_{mvi}^{ct}: MVI_{mode} (Replace missing values in `CheeseType` with **Mode**)
2. O_{oh}^{ca}: OH_0 (Replace outliers in `CheeseAge` with **0**)
3. O_{mvi}^{ppk}: MVI_{mean} (Replace missing values in `PricePerKilo` with **Mean**)
4. O_{oh}^{ppk}: OH_{median} (Replace outliers in `PricePerKilo` with **Median**)

The result of this pipeline is shown in Table 4. The changed values are marked in blue. It can be seen that the randomly selected pipeline leads to an impairment of the data quality: Due to the mode imputation of `CheeseType`, *Emmental*, for example, was assigned the `CheeseType` *Grated*. However, this is not correct; the correct `CheeseType` here would be *Flavoured*. An impact can also be seen for `CheeseAge`. The outliers were replaced by 0, which is obviously not correct and could have a negative impact on further analyses. A further limitation results from the randomly selected order. There are two operators that work on `PricePerKilo` – O_{mvi}^{ppk} and O_{oh}^{ppk}. For O_{mvi}^{ppk}, replacement by the mean value was selected. This is strongly influenced by outliers, which is why these should have been

corrected first (with O_{oh}^{ppk}). But this is not the case here. The effects can be seen in the cheese types *Mozzarella*, *Brie* and *Kasseri*. Here the `PricePerKilo` is now distorted by the outlier.

Optimization with FONDUE

It is thus evident that these approaches are not suitable. They are either too time-consuming or impair data quality. For this reason, the optimization with FONDUE will now be presented and the results compared. As shown in Fig. 7, the data profile and the error profile are first extracted from the data.

Data Profile

As described in Sect. "Rule-based Optimization", the data profile comprises a concrete set of metadata and is required to extract the characteristics of the input data. Listing 2 shows an example excerpt from the data profile of the *cheese data set*.

```
"CheeseType": {
        "datatype": "categorical",
        ...
        },
"CheeseAge": {
        "datatype": "float64",
        "num_descr_stats": {
            "count": 10,
            "mean": 158.0,
            "std": 220.632,
            "min": 8.0,
            "25%": 30.75,
            "50%": 76.5,
            "75%": 92.0,
            "max": 623.0
        }, ...
```

Listing 2 Data profile of the cheese data set – for a better overview, only an excerpt for CheeseType and CheeseAge is shown

Table 4 Result after cleaning the *cheese data set* with a random pipeline, changed values are marked in bold

ID	CheeseName	CheeseType	CheeseAge	PricePerKilo
1	Orkney	Blue	8	40.41
2	Mozzarella	**Grated**	20	**91.18**
3	Pecorino	Flavoured	68	64.31
4	Brie	Grated	**0**	**91.18**
5	Emmental	**Grated**	54	27.70
6	Camembert	Semi-soft	85	9.32
7	Livarot	Semi-hard	93	**49.32**
8	Raclette	**Grated**	**0**	34.80
9	Kasseri	Grated	89	**91.18**
10	Queso de Bola	Smoked	23	49.32

Error Profile

The generated JSON report from CheDDaR (see Listing 1) can be used as an error profile for initial error detection as described. This determines which errors need to be handled by the data cleaning pipeline.

Rule-based Optimization

Once the data profile and error profile have been extracted, they are used in the next step to reduce the search space. As shown in Fig. 7, in *Part 1*, they are compared with the constraints. The data profile (Listing 2), for example, shows that the data type of `CheeseType` is categorical. The error profile (Listing 1) shows that this property has missing values. As described in Sect. "Rule-based Optimization", an exemplary constraint would be that for non-numerical values, replacement by the mean (MVI_{mean}) is not suitable. The same applies to the median (MVI_{median}) and replacement by 0 (MVI_0). This means that for O^{ppk}_{mvi} of the six algorithms for the algorithm class *missing value imputation* (see Table 3), only three remain: MVI_{mode} (*Replacing by mode*), MVI_U(*replacing by "Unknown"*), MVI_{delete} (*deletion of the row*). Of course, this is a very simple constraint. Checking the data type for suitable algorithms could also still be done for the naive approach. However, the constraints can also be more complex (see Sect. 3.1). Nevertheless, it should be emphasized that the search space can already be reduced in the first step by the constraints in combination with the profiles. In the example described here, the search space can be reduced from its initial size of 576 operators (without taking the order into account) to $3 * 4 * 4 * 3 = 144$ operators by the constraints of the data types alone. Taking the order into account, the search space is thus reduced from the original 13, 824 to $144 * 24 = 3,456$ possible pipelines.

In *Part 2* it is now checked which operators are independent of each other. This reduces the possible combinations in terms of sequence. In the application example, operators O^{ct}_{mvi} and O^{ca}_{oh} can be executed at any time. Only the operators O^{ppk}_{mvi} and O^{ppk}_{oh} are dependent upon each other. This reduces the possibility of orders from $4! = 24$ to $2! = 2$. This leads to a reduction of the search space to only $144 * 2 = 288$ possible pipelines. Moreover, this approach results in a further advantage. The independent operators identified are candidates that can be executed in parallel. This leads to a more efficient execution of the data cleaning pipeline.

Best Practices

After reducing the search space using rule-based optimization, a further reduction is achieved by introducing best practices (Sect. "Best Practice"). The naive approach has already shown that when replacing the missing values of `PricePerKilo` with the mean value, the outliers should be cleaned first. This can be incorporated here as best practices. By doing so, part of the order is fixed and thus the search space is reduced. While a generally valid specification was used in this example, these best practices can also depend heavily on the use case, as described in Sect. "Best Practice". This step poses another possible use of CheDDaR. As described, the involvement of domain experts is advantageous for the best possible analysis of errors in the error profile. The domain knowledge collected there and the rules can be incorporated directly into the

optimization in the form of best practices. This is indicated by the red dotted line in Fig. 7.

Cost-based Optimization

Once the search space has been reduced as far as possible, cost-based optimization is used to determine the pipeline that leads to high quality data. For example, possible pipelines could be encoded in the form of a matrix and suitable candidates could be identified using the genetic algorithm or harmony search. Alternatively, agents could be used to find the best possible pipelines using simulated annealing or particle swarm optimization. The possible implementations are described in more detail Sect. "Cost-based Optimization". As explained in Sect. "CheDDaR", CheDDaR is used at this point to evaluate the data quality. We are currently still analyzing the applicability of the various algorithms. One particular challenge is to find a suitable encoding for the pipeline candidates.

Result

As shown in Fig. 7, the output of FONDUE is the abstract description of the pipeline in the form of a pipeline profile. For the application example, the profile is shown in Listing 3. As described, to this end we use ALPINE [34]. For presentation purposes, only an excerpt is shown. The following operators were selected in the subsequent order:

1. O^{ct}_{mvi}: MVI_{U} (Replace missing values in `CheeseType` with **"Unknown"**)
2. O^{ca}_{oh}: OH_{mean} (Replace outliers in `CheeseAge` with **Mean**)
3. O^{ppk}_{oh}: OH_{median} (Replace outliers in `PricePerKilo` with **Median**)
4. O^{ppk}_{mvi}: MVI_{mean} (Replace missing values in `PricePerKilo` with **Mean**)

With the help of this profile, the pipeline can now be executed in the technology of choice. For the *cheese data set*, we have used the prototypical Python adapter[8] as an example. The result is shown in Table 5.

Table 5 Result after cleaning the *cheese data set* with an optimized pipeline, changed values are marked in bold

ID	CheeseName	CheeseType	CheeseAge	PricePerKilo
1	Orkney	Blue	8	40.41
2	Mozzarella	**Unknown**	20	**37.64**
3	Pecorino	Flavoured	68	64.31
4	Brie	Grated	**55**	**37.64**
5	Emmental	**Unknown**	54	27.70
6	Camembert	Semi-soft	85	9.32
7	Livarot	Semi-hard	93	**37.6**
8	Raclette	**Unknown**	**55**	34.80
9	Kasseri	Grated	89	**37.64**
10	Queso de Bola	Smoked	23	49.32

[8] https://gitlab.com/d6745/alpine.

```
{"pipeline" : {
    "general": {
      "pipeline_id": "P1",
      "created": "2025-03-03 11:11:11",
      "dataset": "cheese.csv"
    },
    "operators": [
      {
        "id": "0_oh-ppk",
        "properties": ["PricePerKilo"],
        "algorithm_class": "outlier-handling",
        "position": {"before": ["0_mvi-ppk"]},
        "parameters": {"strategy": "replace", "fill_value": "median"}
      }, ...
    ]
  }
}
```

Listing 3 Pipeline profile – for a better overview, only an excerpt is shown

For `CheeseType`, the missing values have been replaced by "Unknown". Even if the correct values are not available here, at least no incorrect values have been inserted. By replacing `CheeseAge` with the mean value instead of 0, more realistic values could be inserted. Consequently, analyses are not negatively influenced here either. The missing values for `PricePerKilo` are now also more realistic as they have not been distorted by the outlier.

In summary, it was shown that the first naive approach could not be realized as it was too time-consuming. The second naive approach lead to an impairment of the data quality. By using FONDUE, however, the search space could be reduced and consequently better data quality was achieved.

Conclusion and Outlook

In this paper, we have presented our vision of FONDUE, an end-to-end data quality optimizer, along with the corresponding research challenges. Our approach consists of a rule-based optimization, best practices, and a cost-based optimization. In contrast to other work that focuses on cleaning for machine learning, our solution can be used in various scenarios. This includes cases where analysis is performed separately from preprocessing. To achieve this, we presented a data quality framework called CheDDaR. CheDDaR is an important building block for FONDUE. First, it detects errors in a data set. In addition, it can be used to measure the quality of the data. Both tasks are crucial to improving the quality of the data with FONDUE.

Another distinctive factor of our proposed solution is that different optimization goals can be pursued with this approach. This makes it flexible and enables good performance to be achieved in various applications in terms of data quality, resource usage, and human involvement.

The next step, of course, is to implement the optimization, which is the subject of our current research. To this end, we investigate various optimization algorithms. This will

be followed by a comprehensive evaluation using common benchmarks and real-world data sets. Performance measurements will also be part of the evaluation.

In [12], as described above, we have already analyzed the particular challenges that streaming data poses. In our future research, we will investigate in more detail how these challenges can also be addressed with FONDUE.

Acknowledgements
Many thanks to Dennes Krebs and Kevin Kramer for the engaging and helpful discussions on cost-based optimization.

Author contributions
VR, MK and US made substantial contributions to the conception of FONDUE, with VR having the main responsibility for the concrete elaboration of the concept. All authors made substantial contributions to the conception of CheDDaR. ID was mainly responsible for implementing CheDDaR. VR wrote the main manuscript text. ID wrote the main part of the CheDDaR section. All authors reviewed and approved the manuscript.

Funding
Open Access funding enabled and organized by Projekt DEAL. No funds, grants, or other support was received.

Availability of data and materials
The datasets used and/or analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Ethics approval and consent to participate
Not applicable

Consent for publication
Not applicable

Competing interests
The authors declare that they have no Conflict of interest.

Received: 30 September 2024 Accepted: 10 April 2025
Published online: 23 May 2025

References

1. Abedjan Z, et al. Detecting data errors: where are we and what needs to be done? Proc VLDB Endow. 2016;9(12):993–1004.
2. Neutatz F, et al. From Cleaning before ML to Cleaning for ML. IEEE Data Eng Bull. 2021;44(1):24–41.
3. Boehm M, et al. SystemDS: A Declarative Machine Learning System for the End-to-End Data Science Lifecycle. , Amsterdam (2020).
4. Krishnan S, Wu E. AlphaClean: Automatic Generation of Data Cleaning Pipelines. CoRR **abs/1904.11827** (2019).
5. Krishnan S, et al. Towards reliable interactive data cleaning: a user survey and recommendations. New York, NY: ACM; 2016.
6. Restat V, Klettke M, Störl U. Towards an End-to-End Data Quality Optimizer. New York, NY: IEEE; 2024.
7. Yun H, Jeong S, Kim K. Advanced harmony search with ant colony optimization for solving the traveling salesman problem. J Appl Math. 2013;2013:123738–11237388.
8. Kiran MS, Iscan H, Gündüz M. The analysis of discrete artificial bee colony algorithm with neighborhood operator on traveling salesman problem. Neural Comput Appl. 2013;23(1):9–21.
9. Dowlatshahi MB, Nezamabadi-pour H, Mashinchi M. A discrete gravitational search algorithm for solving combinatorial optimization problems. Inf Sci. 2014;258:94–107.
10. Strasser S, et al. A New Discrete Particle Swarm Optimization Algorithm. NY: ACML New York; 2016.
11. Räth T, Onah N, Sattler K. Interactive data cleaning for real-time streaming applications. New York, NY: ACM; 2023.
12. Restat V, et al. Data Cleaning of Data Streams. Heidelberg: Springer; 2025.
13. Ding X, et al. Clean4TSDB: a data cleaning tool for time series databases. Proc VLDB Endow. 2024;17(12):4377–80.
14. Wang RY, Kon HB, Madnick SE. Data quality requirements analysis and modeling. New York, NY: IEEE Computer Society; 1993.
15. Wang RY, Storey VC, Firth CP. A framework for analysis of data quality research. IEEE Trans Knowl Data Eng. 1995;7(4):623–40.
16. Heinrich B, Hristova D. A quantitative approach for modelling the influence of currency of information on decision-making under uncertainty. J Decis Syst. 2016;25(1):16–41.
17. Blake RH, Mangiameli P. The effects and interactions of data quality and problem complexity on classification. ACM J Data Inf Qual. 2011;2(2):8–1828.
18. Schelter S, et al. Automating large-scale data quality verification. Proc VLDB Endow. 2018;11(12):1781–94.
19. Elouataoui W, et al. An advanced big data quality framework based on weighted metrics. Big Data Cogn Comput. 2022;6(4):153.

20. Mahdavi M, et al. Raha: a configuration-free error detection system. New York, NY: ACM; 2019.
21. Heidari A, et al. HoloDetect: few-shot learning for error detection. New York, NY: ACM; 2019.
22. Bors C, et al. Visual interactive creation, customization, and analysis of data quality metrics. ACM J Data Inf Qual. 2018;10(1):3–1326.
23. Shrestha R, et al. Exploratory training: when annonators learn about data. Proc ACM Manag Data. 2023;1(2):135–113525.
24. Restat V, Klettke M, Störl U. FAIR is not enough - A Metrics Framework to ensure Data Quality through Data Preparation. In: BTW. LNI, vol. P-331, pp. 917–929. Gesellschaft für Informatik e.V., Bonn (2023).
25. Ilyas IF, Chu X. Data Cleaning. New York, NY: ACM; 2019.
26. Mahdavi M, Abedjan Z. Semi-Supervised Data Cleaning with Raha and Baran. In: CIDR. www.cidrdb.org, online (2021).
27. Hameed M, Naumann F. Data preparation: a survey of commercial tools. SIGMOD Rec. 2020;49(3):18–29.
28. Klettke M, Störl U. Four Generations in Data Engineering for Data Science: The Past, Presence and Future of a Field of Science. Datenbank-Spektrum, 59–66 (2021).
29. Li P, et al. CleanML: a study for evaluating the impact of data cleaning on ml classification tasks. New York, NY: IEEE; 2021.
30. Freytag JC. A rule-based view of query optimization. New York, NY: ACM Press; 1987.
31. Neumann T. Query Optimization (in Relational Databases). Heidelberg: Springer; 2018.
32. Abedjan Z. Data Profiling. Heidelberg: Springer; 2019.
33. Clemente F, et al. ydata-profiling: Accelerating data-centric AI with high-quality data. Neurocomputing. 2023;554: 126585.
34. Restat V, Störl U. ALPINE: Abstract Language for Pipeline Integration and Execution. In: BTW. LNI. Gesellschaft für Informatik e.V., Bonn (2025).
35. Woltmann L, et al. FASTgres: making learned query optimizer hinting effective. Proc VLDB Endow. 2023;16(11):3310–22.
36. Tsai C, Hu Y. Empirical comparison of supervised learning techniques for missing value imputation. Knowl Inf Syst. 2022;64(4):1047–75.
37. Restat V, et al. GouDa - generation of universal data sets: improving analysis and evaluation of data preparation pipelines. New York, NY: ACM; 2022.
38. Giovanelli J, Pisano G. Towards Human-centric AutoML via Logic and Argumentation. In: EDBT/ICDT Workshops. CEUR Workshop Proceedings, vol. 3135. CEUR-WS.org, Aachen (2022).
39. Hasan K, et al. Missing value imputation affects the performance of machine learning: a review and analysis of the literature (2010–2021). Inf Med Unloc. 2021;27: 100799.
40. Ilyas IF, et al. Estimating compilation time of a query optimizer. New York, NY: ACM; 2003.
41. Dalvi NN, Suciu D. Efficient query evaluation on probabilistic databases. VLDB J. 2007;16(4):523–44.
42. Haas PJ. Monte carlo methods for uncertain data. Heidelberg: Springer; 2018.
43. Mandal PK. A review of classical methods and Nature-Inspired Algorithms (NIAs) for optimization problems. Results Control Optimiz. 2023;13: 100315.
44. Holland JH. Genetic algorithms. Scholarpedia. 2012;7(12):1482.
45. Dowsland KA, Thompson JM. Simulated Annealing. Cham: Springer; 2012.
46. Markl V, Lohman GM, Raman V. LEO: An autonomic query optimizer for DB2. IBM Syst J. 2003;42(1):98–106.
47. Grafberger S, et al. MLINSPECT: a data distribution debugger for machine learning pipelines. New York, NY: ACM; 2021.
48. Codd EF. A relational model of data for large shared data banks. Commun ACM. 1970;13(6):377–87.
49. Boukerche A, Zheng L, Alfandi O. Outlier detection: methods, models, and classification. ACM Comput Surv. 2021;53(3):55–15537.
50. Wang H, Bah MJ, Hammad M. Progress in outlier detection techniques: a survey. IEEE Access. 2019;7:107964–8000.
51. Bantilan N. pandera: Statistical Data Validation of Pandas Dataframes. In: SciPy, pp. 116–124. scipy.org, online (2020).

Publisher's Note