POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

A knowledge-driven approach for supporting data preparation

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Enrico Staiano**

Student ID: 989729
Advisor: Prof. Cinzia Cappiello
Co-advisors: Camilla Sancricca
Academic Year: 2022-23

Abstract

Data-driven management is becoming more and more popular: organizations increasingly rely on collecting and analyzing large volumes of data to support their business decisions. However, the success of the decisions based on data depends greatly on the quality of the data itself. Working with poor quality data introduces the risk of unreliable and erroneous outcomes. Consequently, a thorough data preparation phase aimed to improve data quality is critical. It has been shown that data preparation can take a substantial portion - up to 80 percent - of a data scientist's workload. Data preparation is time-consuming, involving a large and heterogeneous variety of techniques and issues. Moreover, depending on the available data and the desired analyses to perform, different data preparation pipelines may be suitable. For these reasons, a user, especially a non-expert, may find it difficult to navigate the complex journey of data preparation. This thesis addresses this issue by proposing an approach that supports users, guiding them through the preparation process and offering appropriate suggestions for their needs. The methodology presented in this thesis allows users to upload their data and select a Machine Learning Application as the target of their analysis. Subsequently, the methodology guides the users, considering the selected context. A Knowledge Base, designed and implemented during this work, is central to this methodology, containing all the concepts needed for supporting data preparation. The Knowledge Base is constantly queried throughout the process to propose actions appropriate to the specific users' context. To provide tailored suggestions, the proposed methodology relies on classifiers: each classifier takes as input the data uploaded and the application selected by the users and predicts the best method to perform a certain data preparation technique in that context. This thesis focuses specifically on the Imputation technique: two classifiers were developed to predict the best imputation method to fill in the missing values of a dataset or column, considering both the data characteristics and the users' objective analysis. By leveraging the Knowledge Base and the implemented classifiers, the presented methodology is able to support users through the data preparation process, tailoring the proposed preparation pipeline according to the users' needs and offering contextually appropriate suggestions.

Keywords: Data Preparation, Data Quality, Knowledge Base, Imputation

Abstract in lingua italiana

Il data-driven management sta diventando sempre più popolare: le organizzazioni si affidano sempre più alla raccolta e all'analisi di grandi volumi di dati per supportare le loro decisioni. Tuttavia, il successo delle decisioni basate sui dati dipende in larga misura dalla qualità dei dati stessi. Utilizzare dati di scarsa qualità può portare a risultati inaffidabili o erronei. Di conseguenza, è essenziale un'accurata fase di preparazione dei dati volta a migliorarne la qualità. È stato dimostrato che la preparazione dei dati può occupare una considerevole porzione - fino all'80% - del lavoro di un data scientist. La preparazione dei dati è un processo lungo, costituito da un'ampia ed eterogenea varietà di attività e problematiche. Inoltre, a seconda dei dati che si hanno a disposizione e delle analisi che si desidera eseguire, diverse procedure di preparazione possono essere appropriate. Per questi motivi, un utente, soprattutto se non esperto, può trovare difficile orientarsi nel complesso procedimento di preparazione dei dati. Questa tesi affronta questo problema proponendo un approccio che supporta gli utenti guidandoli attraverso il processo di preparazione, offrendo suggerimenti opportuni per le loro esigenze specifiche. La metodologia presentata in questa tesi consente agli utenti di caricare i propri dati e di selezionare un'applicazione di machine learning come obiettivo della propria analisi. Da questo punto in poi, la metodologia guida gli utenti tenendo in considerazione il contesto selezionato. Alla base di questa metodologia c'è una Knowledge Base, progettata e implementata durante questo lavoro, contenente tutti i concetti necessari per supportare la preparazione dei dati. Durante la preparazione, la Knowledge Base è costantemente interrogata per proporre azioni appropriate al contesto specifico dell'utente. Per fornire suggerimenti personalizzati, la metodologia proposta fa uso di classificatori: ogni classificatore prende in input i dati caricati e l'applicazione selezionata dagli utenti e predice il metodo migliore per eseguire una certa tecnica di preparazione in quel contesto. Questa tesi si focalizza in particolare sulla tecnica di imputazione: due classificatori sono stati sviluppati per predire il migliore metodo di imputazione per sostituire i valori mancanti di un dataset o di una colonna, considerando sia le caratteristiche dei dati sia l'obiettivo di analisi degli utenti. Sfruttando la Knowledge Base e i classificatori implementati, la metodologia presentata riesce a supportare gli utenti durante il processo di preparazione

dei dati, variando la pipeline di preparazione proposta a seconda delle esigenze degli utenti e facendo suggerimenti appropriati.

Parole chiave: Preparazione dei dati, Qualità dei dati, Base di Conoscenza, Imputazione

Contents

Introduction

Nowadays, the influence of data has radically changed how organizations operate and make decisions. The unstoppable growth of fields like Internet of Things has caused an unprecedented availability of data. Organizations have recognized the power of data, and the concept of data-driven management has rapidly grown in popularity: the decision-making process is based on insights derived from the data collected.

However, the abundance of data sources often introduces challenges related to data quality. Real-world sources often have quality problems and it is likely to encounter data that is heterogeneous, non-standardized, and contains errors. The quality of the insights drawn from such data, and consequently, the decisions made, is closely linked to the underlying data quality. Analyzing data of poor quality can lead to flawed or unreliable results: data quality problems have a strong impact on the outcomes of data-driven analyses. It is therefore critical for a data-driven organization to continuously monitor the quality of the data it uses.

Therefore, the significance of an effective data preparation phase becomes evident. Properly preparing data can improve data quality and lead to more accurate analysis results. However, the data preparation process is complex and time-consuming, constituting a large share of the workload required to perform any analysis. Many possible preparation actions can be applied to the data, and depending on the specific case, different actions may be appropriate. Dealing with this complex preparation process can be difficult, especially for an inexperienced user.

This thesis aims to create an approach to support users throughout the entire data preparation process, directing them toward the preparation actions most suitable for the data they are working on and for the analyses they want to perform.

The methodology proposed in this work relies on a Knowledge Base, implemented with a graph database, containing all the concepts and knowledge needed to support users in the data preparation process. Furthermore, the methodology employs classifiers to predict which data preparation methods are best suited to the users' needs. Specifically, during the thesis, two classifiers were implemented to predict the optimal imputation method to

be used to fill in missing values in datasets and columns, considering the users' specific context.

Finally, to demonstrate a practical application of the knowledge base and classifiers implemented, they were integrated into an existing data preparation tool, which was enriched with new features leveraging the concepts developed in this thesis.

Structure of the document

Chapter 1 introduces the topics on which this thesis is based, and contains a comparison of some well-known frameworks for data preparation.

Chapter 2 describes the proposed methodology, explaining all its aspects in detail. This chapter contains the conceptual description of the designed Knowledge Base.

Chapter 3 focuses on how the Knowledge Base was implemented using a graph database.

Chapter 4 contains details regarding the implementation of the classifiers: the whole implementation process is described, starting from how the knowledge needed for their training and testing was generated to the results obtained.

Chapter 5 describes how the concepts developed in this thesis were integrated into a data preparation tool.

Chapter 6 concludes by summarizing the work done and proposing possible future works.

1 State of the art

This chapter describes the fundamental concepts and issues explored in this thesis. In particular, Section 1.1 introduces the general concept of data quality. Subsequently, Section 1.2 presents a more in-depth description of the data quality dimensions. Section 1.3 focuses on strategies to improve the quality level of data.

Section 1.4 contains a description of the main data quality issues and data preparation techniques within the context of machine learning.

Section 1.5 is dedicated specifically to the Completeness dimension, which is highly relevant for the research conducted in this thesis.

Section 1.6 provides an in-depth comparison of frameworks for automatic and semi-automatic data preparation.

Finally, Section 1.7 contains an introduction to the topic of graph databases, pertinent to the implementation part of the thesis.

1.1. Data Quality

Nowadays, the role of data is becoming more and more significant. Companies rely on collecting and analyzing large volumes of data to support their business decisions: data-driven management has become central.

However, as the quantity and utilization of data increase, so do the issues about the quality of the data collected. The success of the analyses and decisions based on data depends greatly on the quality of the data itself. Indeed, data quality issues significantly impact the outcomes of data-driven analyses [4].

The data quality concept can be defined in various ways. The traditional definition of data quality is the ability of data to meet user requirements, also expressed as "fitness for use" [37]. Another possible perspective, from an Information System point of view, defines data quality as the absence of contradictions between the data and the real-world [26].

There are many possible causes for poor data quality. The main ones are the following:

- **Data integration**: merging different data sources in a single comprehensive collection might cause data quality problems that need to be carefully addressed. Therefore, data integration is a common and often necessary operation, but it is sometimes dangerous from a data quality point of view.
- **Data usage**: changes in the usage of data can introduce errors and inaccuracies. For instance, data initially employed in an operational process may not prioritize certain details (and inaccuracies) that instead become crucial when used in a decision-making process.
- **Historical changes**: the importance of data for an organization might change over time, leading to issues with some data being ignored or badly collected until when data becomes necessary.
- **Data enrichment**: enriching data using external or untrusted sources can be dangerous, as it may introduce errors or other issues, such as duplicates or heterogeneous data formats.

Regardless of the causes, poor data quality can occur in different and heterogeneous ways. In real-world data, some frequent issues include missing values, duplicates, non-uniform data formats, typos, outdated data, and many more. Addressing this wide variety of problems is time-consuming and expensive, and it has been demonstrated that data preparation can take up to 80% of the work of a data scientist [31].

A common strategy for addressing data quality issues, or in general for maintaining a high data quality level, involves the following four-phase approach: definition of suitable data quality dimensions, assessment of data quality level using these dimensions, analysis of data quality issues, execution of actions for quality improvement. This process not only helps monitor and control data quality issues but also contributes to improving the overall quality level. It is often referred to as Total Data Quality Management [36].

The following section provides the definition of data quality dimension and presents an overview of the most commonly used dimensions.

1.2. Data Quality Dimensions

As previously mentioned, data quality takes into account various issues.A data quality dimension captures a specific aspect of the general concept of data quality. By defining suitable dimensions, it is possible to simplify the quality assessment and the analysis of

the quality issues.

Over time, many data quality dimensions have been proposed in the literature.

A first distinction can be made between data dimensions and schema dimensions. The former refers to the quality of data values. The latter refers to the quality of the schema with which data was stored. This thesis only considers the data dimensions described in detail in the present and the following sections.

One of the most used classifications is the one presented in [37] and illustrated in Figure 1.1. This classification divides the data quality dimensions in four categories:

- **Intrinsic dimensions**: these dimensions capture the quality that data has intrinsically on its own.
- **Contextual dimensions**: dimensions that take in consideration the context where data is used. These dimensions are strictly related to the context of the process using the data.
- **Representational dimensions**: these dimensions capture aspects related to the quality of data representation.
- **Accessibility dimensions**: dimensions related to the accessibility of data.

Data Quality dimensions			
Intrinsic dimensions	**Contextual Dimensions**	**Representational Dimensions**	**Accessibility Dimensions**
Believability Accuracy Objectivity Reputation	Value-added Relevance Completeness Timeliness Appropriate amount of data	Interpretability Ease of understanding Representational Consistency Concise representation	Accessibility Access security

Figure 1.1: Data quality dimensions classification [37]

Furthermore, dimensions can be divided in objective and subjective dimensions. The subjective dimensions can be only evaluated by users, considering their specific needs and goals. The objective dimensions, on the other hand, can be associated to numeric metrics and objectively measured.

An objective dimension can be associated to one or more metrics, to quantitatively measure it. It may be appropriate to use different metrics for the same dimension, depending

on the specific cases.

Once the objective dimensions have been defined and assigned to their metrics, there is the assessment phase, during which the data quality level is precisely measured and evaluated.

In the following sections, there is a description of the most used objective dimensions.

1.2.1. Accuracy

Accuracy is defined as the closeness between a data value v and another value v', which is regarded as the correct representation of the real-world concept that the value v is representing. In other words, accuracy measures how much correct the data is [37].

Accuracy is often categorized into two types: syntactic accuracy and semantic accuracy [27].

Syntactic accuracy focuses on whether a value v belongs or not to the domain D to which it is assigned. For instance, if the value v appears as an address in a column of phone numbers, it is not considered syntactically accurate, because it does not belong to the domain of the column. If a value belongs to its designated domain, it is considered syntactically accurate, regardless of whether it is actually the correct value.

On the other hand, semantic accuracy considers the intrinsic correctness of value v, i.e., how close v is to the corresponding correct value v'. To assess semantic accuracy, additional knowledge is typically required, while assessing syntactic accuracy does not usually need additional information.

To measure the accuracy of a dataset, one of the most commonly used metrics is the ratio between the number of accurate values and the total number of values:

$$\text{Accuracy} = 1 - \frac{\text{Number of incorrect data values}}{\text{Total number of data}}$$

In this formula, the number of incorrect values can be calculated either considering the domain of the values (syntactic accuracy) or with respect to their true value (semantic accuracy).

1.2.2. Completeness

The completeness of a dataset indicates the extent to which that dataset represents the corresponding real-world [5]. It is an indication of the degree with which the dataset

is comprehensive, compared to the respective set of real-world objects. Typically, the completeness is associated with the number of missing values within the dataset. Note that it is often not enough to consider only the number of missing values, but it is necessary to also understand the reasons for their absence. For instance, there is a significant difference between a missing value indicating a non-existing concept (in which case, the value is not truly missing) and a missing value indicating an unknown or unrecorded value.

The most common and simple metric used to assess completeness is the ratio between not missing values and the total number of values:

$$\text{Completeness} = 1 - \frac{\text{Number of missing values}}{\text{Number of total values}}$$

It is possible to evaluate completeness at various granularities: at the single data sample level, at the column level, and at the dataset level.

This thesis gives particular attention to this dimension, and a more precise description of completeness issues and how to address them can be found in Section 1.5.

1.2.3. Consistency

The consistency dimension observes the violation of semantic rules defined over a set of data items [5]. Several kinds of semantic rules can be defined. The most common ones are integrity constraints and business rules.

Business rules are usually defined by experts in the specific data domain, who use their expertise and knowledge to define semantic rules.

The integrity constraints specify dependencies that must be respected by the values of one or more attributes of the data. Among integrity constraints, the main types of dependencies are key dependencies, inclusion dependencies, and functional dependencies. These types of dependencies can be briefly described in the following way: a key dependency indicates that an attribute (or a group of attributes) must have distinct values for every data element; an inclusion dependency indicates that every value assumed by an attribute (or a group of attributes) A must also exist in another attribute (or group of attributes) B; a functional dependency specifies that the value of an attribute (or group of attributes) A uniquely determines the value of another attribute (or group of attributes) B.

The most common metric used to assess consistency is the ratio between the number of consistency checks with positive results and the total number of checks performed:

$$\text{Consistency} = 1 - \frac{\text{Number of violations}}{\text{Total number of consistency checks performed}}$$

Therefore, a dataset has the highest level of consistency if all the semantic rules are satisfied.

1.2.4. Timeliness

Timeliness is a time-related dimension that measures the extent to which data are sufficiently up-to-date for a task [37]. It is important to monitor the temporal validity of data because, in many contexts, data can quickly become old and out of date. To calculate timeliness, two components are necessary [7]:

- Currency: the time interval passed between the last update of the considered data and the moment in which the data is read.
- Volatility: the average validity time of the considered data. It is an indication of how often data changes, i.e., it is the frequency of updates of that data.

It is possible to understand whether data is still valid using these two components. The timeliness can be measured using the following formula:

$$\text{Timeliness} = \max\{0, 1 - \frac{\text{Currency}}{\text{Volatility}}\}$$

The measure of timeliness can assume values between 0 and 1, with 0 indicating out-of-date data and 1 indicating perfectly up-to-date data.

1.2.5. Other dimensions

In addition to those described in the previous sections, other important dimensions are [4]:

- **Uniqueness**: this dimension refers to the absence of duplicates in the data, i.e., multiple data elements corresponding to the same real-world concept.
- **Accessibility**: this dimension measures the ability of the users to access the data from their own culture and background, physical status, and technologies available.
- **Readability**: this dimension refers to the ease with which users can understand and utilize the information contained in the data.

1.3. Data quality improvement

Once the data quality dimensions have been defined and assessed, several possible approaches exist for improving the data quality level. The two main strategies are the following:

- **Data-based approach**: this approach focuses on the data itself, aiming to identify and correct the errors present in the values without considering the process and context in which the data will be used. This direct and immediate approach works very well in the short term, but it is necessary to repeat the data quality improvement process on each new dataset and every time the data are updated.
- **Process-based approach**: this approach focuses on the process in which data are created and used. The aim is to analyze the process, in order to discover and eliminate the root cause of the data quality problems. This approach may be more complex, but it ensures that the same kinds of errors are not repeated in the long term.

This thesis is focused on the data-based approach.

In particular, the process considered in this work for data quality improvement is based on data cleaning and consists of the following steps:

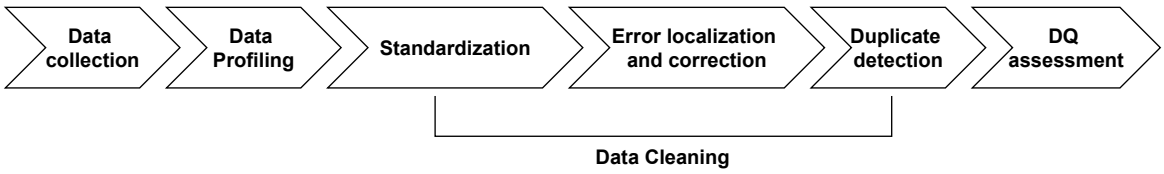

Figure 1.2: Data Quality Improvement Process

After collecting the data, the first step of the process is data profiling.
Data profiling is the set of activities and processes designed to examine a given dataset in order to collect statistics and, in general useful metadata of that dataset [1]. Data profiling aims to provide a better understanding of the data, also helping the following data preparation phases.

Some of the main tasks performed during the data profiling phase include:

- Calculation of cardinalities, e.g., number of rows and columns of the dataset, number

of null values, number of distinct values.

- Analysis of value distributions, which involves assessing the distribution of data in the columns, including calculating extremes in numeric columns, quartiles, and constancy.
- Examination of data types, formats, and patterns: this task involves examining the column data types and scanning the dataset to identify recurring patterns and formats.
- Dependency discovery: during this task key, inclusion, and functional dependencies are searched within the data. These dependencies are very useful for individuating (and subsequently correcting) errors in the dataset.

After data profiling is finished, the data cleaning phase begins. During the data profiling phase, the dataset is never modified or altered, while the data cleaning procedure modifies the values, correcting the errors. More precisely, data cleaning is defined in [25] as detecting and removing inconsistencies and errors in data to improve quality. Data cleaning consists of three main steps: standardization, error correction, and duplicate detection.

The objective of the standardization step is to achieve uniformity in the formats of the data. This phase may involve various operations such as data type conversions, merging or splitting of columns, and discretization of numerical values. Standardizing data formats may necessitate actions like, for instance, altering formats of dates, addresses, and telephone numbers or converting different currencies [3]. In some cases, domain-specific transformations might be needed, such as expanding acronyms; however, these transformations often require external knowledge [15].

At the end of the standardization step, the obtained data should be homogeneous and have uniform formats. At this point, the error localization and correction starts. During this central section of data cleaning, errors in the data are actually localized and corrected, also exploiting the results of the data profiling phase. Some of the activities performed during this step are the localization and correction of inconsistencies and of incomplete data. Note that these two are only examples of the many activities that can be included in the error localization and correction step.

To individuate inconsistencies, the semantic rules previously defined and the dependencies found in the data profiling phase are checked. The records in which these constraints are not verified are considered erroneous [4]. Once the inconsistent data elements are found, there are several possible strategies: a possible option is to delete the erroneous elements, while another option is to correct the errors, substituting the erroneous values with correct

ones. Modifying or inserting values to correct erroneous fields is also called imputation.

Details on how to handle incomplete data can be found in Section 1.5.

After the error localization and correction, the last step of data cleaning is duplicate detection. Duplicate detection can be defined as the discovery of multiple representations of the same real-world object [22]. The main challenges of duplicate detection are the research of suitable similarity measures and efficient algorithms to individuate duplicates. Similarity measures are fundamental to find representations that are not identical but that refer to the same real-world object. The utilization of efficient algorithms is needed because the datasets have often a large volume, making it not possible to compare all the data elements.

Some of the most common techniques used to individuate duplicates reducing the search space are the following:

- Blocking: the data is partitioned in separate blocks and the comparisons are limited to records within the same block. Blocking is implemented by choosing a blocking key and grouping into a block all records with the same value of that key.
- Sorted neighborhood: data is first sorted using a key (a significant attribute, or even multiple attributes). Then a window of fixed size is shifted over the sorted data, and only the records within the window are compared.
- Pruning (or filtering): this method has the objective removing from the search space all records that cannot match each other, without actually comparing them.

After duplicates referring to the same concept are detected in the data, they are typically deleted.

At the end of the data cleaning phase, a new data quality assessment is conducted on the dataset, to determine whether the performed actions positively impacted the quality level.

1.4. Data Quality for Machine Learning

When using a dataset as input of a Machine Learning (ML) algorithm, it is of fundamental importance to consider the level of quality of the dataset and, in general, how that dataset has been prepared. If the data on which the ML algorithm works has data quality problems, the performance of the algorithm will be compromised [8]. Therefore, data preparation is one of the most important tasks for the success of any ML algorithm.

In this section, some of the most commonly performed operations to prepare data for machine learning are described.

While some ML algorithms can deal with non-numeric data types directly, other algorithms necessitate numerical attributes as input. In the last case, it is required to convert all the non-numeric attributes of the dataset into a numerical format [8].

Regarding non-numeric ordinal attributes, the conversion must preserve the order of the original values. Therefore, the assignation of the numerical values must reflect the original ordering.

Regarding instead unordered nominal attributes, a commonly adopted solution is one-hot encoding [9]. In this method, the original nominal attribute is transformed into a set of binary columns. Each column refers to a distinct value of the original attribute. For a given sample, the column associated with the original value present in the sample is assigned a value of 1, while all the other columns are set to 0. The main advantage of one-hot encoding is that it does not impose an implicit order or hierarchy on attributes. However, when dealing with nominal attributes that can assume many distinct values, this method could increase significantly the dimensionality of the dataset.

Another operation that, in some cases, can be useful in data preparation for machine learning is discretization. This operation converts a continuous variable into a discrete ordinal variable. Discretization divides the range of a continuous variable in a set of intervals, called bins, that are used to obtain a discrete variable [12]. Discretization can be used for several possible purposes. For instance, some ML algorithms work better with discrete values, like Naive Bayes classifiers. Discretization can also be applied to reduce the data volume.

For many machine learning algorithms, in particular, for distance-based methods, it is fundamental, during the preprocessing, to normalize the data. Normalization is a data preprocessing technique that rescales the attributes of the dataset. It is aimed to scale all the attributes to a similar range. Without normalization, a possible issue is that attributes with large ranges might out-weight attributes with small ranges. Two well-known normalization methods are Min-Max scaling and z-score normalization [16].
Min-Max scaling scales the values of an attribute to a desired range (usually [0,1] or [-1,1]), by subtracting from each value the minimum of the attribute and dividing by the range of the attribute.
Z-score normalization subtracts from each value the mean of the attribute and divides it by the standard deviation of the attribute. After this operation, the normalized attribute will have a mean of 0 and a standard deviation of 1.

Before utilizing a dataset as input for a ML algorithm, handling the missing values within that dataset is recommended. Many ML algorithms do not function properly when faced with missing data, while others may handle them in unpredictable ways. Therefore, it is a good practice to address this problem during preprocessing, to avoid undesirable behaviors. Various approaches can be used to handle missing values in a dataset. For more information on the available strategies, see Section 1.5.

Sometimes, datasets may contain columns where every sample assumes the same value. These columns, often called zero-variance predictors, are typically useless for modeling. Therefore, in the majority of cases, these attributes are deleted from the dataset before applying the ML algorithm [8].

A similar situation arises when a dataset contains columns with few unique values. These columns are often referred to as near-zero variance predictors. Near-zero variance predictors may or may not be helpful for modeling purposes. Hence, evaluating the specific case before removing or keeping the column is advisable.

Another type of column often useless for ML algorithms is ID-like fields. In this kind of attribute, each sample assumes a unique value. ID-like fields typically do not contain meaningful information for machine learning models and are often removed before applying the ML algorithm.

An important operation to perform during preprocessing, is the detection and deletion of unwanted duplicates present within the dataset. Duplicates not only could be useless to the modeling process, but, more importantly, they could also be misleading during the model evaluation and testing phases.

A frequent issue in machine learning, particularly for classification algorithms, is represented by unbalanced datasets [20]. An unbalanced dataset contains an unequal class frequency: some classes appear in the samples much more frequently than others. In these cases, the ML algorithm receives many more examples from one class (or a small number of classes), and this biases it towards that specific class. An approach to address this problem is to collect more data from the minority classes. However, this approach is often not feasible. Some commonly used strategies are undersampling and oversampling. Considering a binary classification problem with an unbalanced dataset, undersampling aims to balance the dataset by reducing the number of instances of the majority class. Only some samples of the majority class are included in the dataset, to balance the class distribution. A disadvantage of this method is the possible loss of valuable data.
On the other hand, oversampling aims to balance the dataset by increasing the number of samples of the minority class. To reach this goal, this method replicates some samples

of the minority class, or creates new synthetic data points.

The described operations are only some of the many possible data preparation actions that can be used to preprocess a dataset for machine learning. When working on dataset preparation, it is always important to consider the specific characteristics of the dataset and the needs of the ML algorithm to be employed.

1.5. Addressing Completeness: Considerations and Strategies

This thesis is particularly focused on completeness, therefore, this section discusses more details about this dimension, along with a description of the most used methods for handling missing values in data.

As described in Section 1.2.2, completeness indicates the degree to which a data collection contains all the data describing the corresponding set of real-world objects. Completeness is strictly related to the presence of missing values in the dataset, however only counting the number of null values is sometimes not enough to properly assess the level of completeness of the data.

When assessing the level of completeness of a dataset, one of two assumptions can be made: the Closed World Assumption (CWA) or the Open World Assumption (OWA) [4]. The CWA states that only the data elements present in the dataset are the ones of interest. Note that these data elements can still have missing values. The OWA, on the other hand, states that there could be elements of interest in the real-world that are not represented in the dataset. Consequently, even if the dataset contains no missing values, completeness may still not be optimal.

Another aspect to consider when assessing completeness is the potential presence of truncated and censored data [14].

As briefly mentioned in Section 1.2.2, it is important to consider not only the number of null values present within a dataset but also their meaning. A null value does not necessarily indicate a missing value, but it may sometimes indicate a default value for a particular attribute. Furthermore, a null value can also indicate a non-existent real-world concept: for instance, in an attribute containing telephone numbers, a null value could denote that a person does not own a telephone. In both cases, the null values do not represent actual missing values, and the dataset can be considered complete. Therefore, it is important to distinguish between null values representing values existing in the real-

world but missing in the data, thereby causing the dataset to be incomplete, and those representing other cases.

Once the missing values are individuated, several possible strategies can be used to handle them. These strategies can be divided into two categories: deletion and imputation. Deletion aims to eliminate the sections of data containing missing values. On the other hand, imputation replaces the missing data with substitute values. In the following paragraphs, there is a description of several methods for both deletion and imputation.

Considering a dataset, the two main deletion methods that can be applied are: deletion by row and deletion by column [17].

Deletion by row identifies all the tuples of the dataset containing missing values and deletes them. This method is very simple and fast, but has some disadvantages. Firstly, if the number of missing values is large, this method could lead to the deletion of most of the entire dataset, drastically decreasing the effectiveness of subsequent analyses performed on the data. Secondly, the missing values may not be uniformly distributed among the tuples, but may be concentrated in particular segments of the data, and deleting such tuples would result in the loss of valuable information and in obtaining a biased representation of the data.

Deletion by column deletes all the columns of the dataset containing missing values. Similarly to deletion by row, this method is simple and computationally not expensive, however, it also shares some of the problems of the previous method. In particular, the main disadvantage is the loss of data, which can be quite significant. Consequently, the application of this method is usually suitable only when the number of missing values is very low, or when these missing values are highly concentrated within specific columns.

Regarding the imputation strategy, there is a large number of possible methods, ranging from more straightforward approaches to more complex ones, some of which also exploit machine learning algorithms.

Deterministic Imputation: this method imputes a missing value by using logical relations and dependencies between the attributes of the dataset. It derives the correct value of the missing item using a deterministic approach. For instance, a possible deterministic approach is to exploit a functional dependency present in the dataset to uniquely determine a missing value from other attributes' values. In general, any dependency or semantic rule that can determine the correct value of the missing item can be applied. The clear advantage of this method is that it imputes the correct value of the missing item. However, a useful rule to impute missing values is not always available; therefore

is often not possible to apply deterministic imputation for all the missing values of a dataset. In general, when dealing with missing values, it is always wise to check whether deterministic imputation can be applied, before trying different methods.

Standard Value Imputation [32]: this method imputes every missing value using a constant standard value. Any desired value may be used as an imputation value. Some popular choices are the value 0 for numeric-type columns and the value "Missing" for string-type columns. The main advantage of this method is its simplicity. However, the main disadvantage is that since it imputes the same value for all the attributes, the imputed values in an attribute are typically totally unrelated to the other values within that attribute.

Mean Imputation [2]: this method replaces the missing values of an attribute with the mean of the observed values of that attribute. For its nature, this method can only be applied to numerical attributes. An advantage of Mean Imputation is that it preserves the mean of the observed data.

Median Imputation [2]: this method uses the same approach of Mean Imputation, but, instead of the mean, it uses the median of the attribute as imputation value. Similarly to the previous method, it can only be applied for numeric-type attributes. In datasets containing outliers, Median Imputation is a more robust method than Mean Imputation, because it mitigates the effect the outliers have on the imputation value.

Mode Imputation [17]: this method replaces the missing values of an attribute with the mode of the observed values of that attribute. Differently from Mean and Median Imputation, this method can be applied for every kind of attribute. However, this method is particularly appropriate for categorical data, while it may not be the best choice for continuous attributes.

Forward-Fill Imputation and Backward-Fill Imputation [28]: these two methods replace the missing values with previous and subsequent non-missing values present within the dataset. These methods may be valid when the order of the tuples in the dataset is relevant, for instance, in the case of time-ordered datasets.

Random Imputation [32]: this method replaces the missing values of an attribute with random values drawn from the observed values of that attribute.

While the previously described methods are simple, the following imputation methods rely on more complex techniques.

Linear Regression Imputation [29]: this method imputes the missing values of an attribute A of a dataset by building a linear regression model. The model is trained using

the observed values of A as the target variable and the other attributes of the dataset as features. Once the model has been trained, it is used to predict and fill in the missing values present in attribute A. Differently from Mean Imputation and similar methods, Linear Regression Imputation does not exploit only the information within the attribute containing the missing values. This approach leverages all the information available in the entire dataset, taking into account also the relationships between the attributes. A disadvantage of this method (and similar methods) is that it is more computationally expensive in comparison to simpler approaches. This imputation method can be applied only to impute missing values of numerical columns.

Logistic Regression Imputation [29]: this method is similar to Linear Regression Imputation. The main difference is that, instead of using a linear regression model, it builds a logistic regression model to predict the missing values of an attribute. Using this method to only impute missing values of categorical attributes is appropriate.

K-Nearest Neighbors Imputation [2]: this method imputes the missing values of a sample S by looking at similar samples within the dataset. This method uses a similarity metric to identify and retrieve the K most similar samples to S, called neighbors. The imputed values for categorical variables are determined by the most common category among its neighbors to impute the missing values of S. For numerical variables, the imputed values are calculated as the mean of the corresponding values of the neighbors. KNN Imputation is a particularly valid method when the similarity between samples is relevant in the dataset. One of the disadvantages of this method is its computational cost, which can be high in the case of large datasets. Moreover, it may require a careful parameter selection for the choice of K and similarity metric.

Multiple Imputation by Chained Equations (MICE) [40]: this well-known imputation method employs an iterative approach to impute all the missing values of an entire dataset. A series of predictive models are built in order to predict and then impute the missing values in the dataset's columns. In particular, the method works as follows. Initially, the missing values are temporarily filled in randomly. Then, a sequence of iterations is performed. In each iteration, a column is considered as target variable, and a model is built to predict that column using the other columns of the dataset. This process is repeated for each column of the dataset. MICE is a powerful method that imputes all the missing values of a dataset, considering the relationships between the attributes.

SoftImpute [34]: this method takes a different approach with respect to the previous methods. It relies on the concept of matrix completion. In summary, this method decomposes the dataset (considered as a matrix) using Singular Value Decomposition (SVD)

and then reconstructs it, filling in the missing values in the process. SoftImpute also works well for large datasets with high dimensionality.

The described methods are only some of the available choices for imputing the missing values of a dataset.

Furthermore, it is possible to combine these methods. For example, in a dataset containing both categorical and numerical attributes, a possible choice is to impute the categorical columns using Mode Imputation and the numerical columns with Mean Imputation.

1.6. Approaches for automatic and semi-automatic design of data preparation pipelines

With the increase in awareness regarding the importance of data preparation and quality, numerous approaches and frameworks have emerged to support users in preparing their data. Some of these approaches assist users through the preparation process step by step, suggesting individual actions, while others automatically propose complete preparation pipelines. These methods differ significantly for the kind of actions suggested and the strategies utilized. This section describes some common aspects and strategies employed in these frameworks.

To determine the optimal data preparation pipeline to apply, a possible strategy is to consider the application to be executed on the data and to search for the optimal sequence of tasks such that the quality of the application results is maximized. This strategy is adopted in works such as [6] and [19]. In these frameworks, a machine learning application is selected, and then, using various approaches, a pipeline is constructed containing preparation actions aimed to optimize the performance of that application. In particular, [6] utilizes a Q-Learning-based approach, while [19] relies on ensemble methods.
This strategy is highly effective when users prepare data for a specific objective, as the suggestions are tailored to that particular situation.

Other works suggest optimal data preparation actions by examining past preparation pipelines. This strategy involves studying cleaning tasks used previously and learning from them to propose promising actions for new datasets. This strategy is evident in [39] and [21]. In particular, in [39], information present in past data science notebooks is gathered. The information is then leveraged to suggest data preparation actions for new problems. The approach used in [21], on the other hand, learns from how past datasets were cleaned. When a new dataset has to be prepared, its profile is extracted, containing metadata summarizing its characteristics. Subsequently, the new dataset is compared to

already cleaned datasets, and considering the similarity between their profiles, appropriate error detection strategies are proposed.
This strategy facilitates the reuse of past work, and is particularly effective when the profile of the dataset to be prepared is similar to profiles of previously considered datasets.

Many frameworks designed for data preparation do not operate as fully automated systems but involve users. This approach is called Human-In-The-Loop (HIL). The degree of user involvement may vary depending on the approach considered. For instance, in [31], the presented framework suggests multiple potential operations, leaving the user to choose which one to execute. In [21], probable errors in the dataset are identified automatically, but users are asked to confirm or reject these identifications. Not all the works involve the users along the entire process: some approaches initially gather users' preferences and then proceed automatically. In [18], for instance, users are required to provide a data quality metric to optimize, after which the framework automatically generates a data cleaning pipeline considering the metric provided. Certain methods, like [13], opt for crowdsourcing instead of seeking feedback from individual users.
The Human-In-The-Loop approach offers several advantages. Certain data preparation tasks are challenging to execute in a fully automated way, and HIL addresses these complexities. User involvement allows users to tailor preparation to their preferences, resulting in more suitable results. Furthermore, HIL ensures that users are always monitoring their data, keeping track of all the modifications applied.

To semi-automate dataset preparation, a viable strategy is to conduct a series of checks on the dataset and advise users on which data preparation actions to execute based on the results of these checks. This strategy is adopted in [31]. This work employs an acyclic graph to control the flow of the operations. Following this graph, the system performs data quality checks on the considered dataset, presenting the result to the users, along with the suggested actions.
One of the main advantages of this strategy is that the suggested actions are selected specifically for the data quality problems of the dataset.

Some recent approaches are starting to employ innovative technologies for data preparation. For instance, the approach presented in [33] makes use of transformers. In particular, this work employs a pre-trained autoencoder to correct missing values and/or errors within tuples. Furthermore, the autoencoder can be fine-tuned to perform more specific data preparation tasks, like normalization and data format transformation.

The previous paragraphs show that the approaches found in the literature for supporting users in data preparation are highly heterogeneous. Not only the adopted strategies are

entirely different from each other, but also the actions proposed are often of different kinds.

In this thesis, nine approaches were analyzed and confronted. Table 1.1 summarizes the main aspects of these approaches. In particular, the aspects considered are the following:

- **Human-In-The-Loop** indicates whether the user is involved in the approach and at what level.
- **Recommendation Granularity** specifies the kind of recommendations the approach considers: whether it makes high-level suggestions or recommends specific actions and methods.
- **Recommended Actions** specifies which actions can be suggested by the approach.
- **Recommendation Criteria** indicates what criteria the approach considers to make its suggestions.
- **DQ dimensions** are the data quality dimensions considered in the approach.
- **Strategy Used** indicates on what particular strategy or idea the approach is based.

Paper	Human-In-The-Loop	Recommendation Granularity	Recommended Actions	Recommendation Criteria	DQ dimensions	Strategy Used
Learn2Clean (2019) [6]	No.	Data preparation pipeline, specifying the methods to use.	Normalization, feature selection, imputation, outlier detection, deduplication, consistency checking.	ML model performance (accuracy, MSE, silhouette).	Not explicitly considered.	Reinforcement learning (Q-learning).
BoostClean (2017) [19]	No. However, the user can provide custom error detection and repair libraries.	Automatic error detection and repair.	Operators in user-specified libraries, and simple methods of imputation/discard record.	ML model performance.	Focus on domain value violations, hence mainly completeness and accuracy.	Ensemble method (boosting).
RPT (2021) [33]	No.	Actions automatically executed.	Reconstruction of tuples with missing values. With fine-tuning also standardization and data transformation.	RPT is trained to perform tuple reconstruction tasks.	In the basic version, mainly completeness.	Encoder-decoder transformer.
Auto-Suggest (2020) [39]	No.	User can select an action and the tool suggests how to perform it, or the tool can suggest the next best action.	Join, group-by, pivot, unpivot.	Based on past data science notebooks.	Not explicitly considered.	Learning from data preparation pipelines present in past notebooks.
DQA (2019) [31]	Yes. To select the next operation among some options; to verify the output of the validator; to add custom checker functions.	The tool suggests to the user a specific action to solve a problem (the execution of the action is not automatic).	The operations mentioned are: No operation, Data Imputation, Column Filter, Row Filter, Perform deeper check.	Output of data quality checks.	The tool doesn't consider explicitly DQ dimensions, some of the DQ checks mentioned are: missing values, duplicates, anomaly detection, distribution of values.	The tool uses a DAG that performs DQ checks and considering the output of these checks suggests actions to perform (with the help of the user).

Table 1.1: Approaches Confrontation Table - Part 1

Paper	Human-In-The-Loop	Recommendation Granularity	Recommended Actions	Recommendation Criteria	DQ dimensions	Strategy Used
Simultaneous Improvement of ML Model Fairness and Performance (2021) [11]	No.	Automatic detection and elimination of bias inducing samples.	Elimination of bias-inducing samples.	Analysis of samples and model output.	Bias is expressed considering the definitions of independence, separation e sufficiency.	Search of similar tuples with different values of protected attributes and different outcomes. The found tuples are flagged as bias-inducing.
AlphaClean (2019) [18]	Yes. The user has to provide data cleaning libraries and has to define the DQ metric to optimize.	AlphaClean suggests the optimal detailed data cleaning pipeline.	Operations in the libreries provided by the user.	Optimization of DQ metric provided by the user.	DQ metric provided by the user.	Data cleaning as an hyperparameter tuning problem.
Towards Automated Data Cleaning Workflows (2019) [21]	Yes, the user has to annotate samples of the detected errors.	The tool suggests a data cleaning pipeline, but in the paper the level of granularity is not specified.	Not specified.	Similarity of the new dataset with already cleaned datasets, output of error detection strategies, user annotations.	Not specified.	Similarity of new dataset with past, already cleaned, datasets.
KATARA (2015) [13]	Crowdsourcing. Furthermore, the user can pick a repair (among k possible repairs) for each incorrect tuple.	The tool annotates each tuple as correct/incorrect, and it suggests k possible repairs for each incorrect tuple.	Substitutions of some values in each incorrect tuple.	Knowledge base and crowdsourcing to detect incorrect tuples, least possible number of changes to suggest repairs.	Not explicitly considered.	Knowledge bases and crowdsourcing.

Table 1.1: Approaches Confrontation Table - Part 2

1.7. Graph Databases

Since a graph database was implemented during this thesis, this section briefly introduces graph databases.

As explained in [24], a graph database is a database that uses graph structures to store data. In particular, data is represented using nodes and edges (called relationships). Nodes, serving as entities, can be tagged with labels that can be considered as the types of the nodes and can have properties, the "attributes" of the nodes. Relationships are connections between two nodes. Each relationship has a direction, a start node, an end node, and a type, and can have properties (like nodes). In a graph database, the edges usually hold the most important information: the connections between the concepts.

An important property of graph databases is to provide index-free adjacency: each node is a pointer to its adjacent elements. Therefore, without global indexes, each node directly references its adjacent nodes.

When the primary goal is to analyze the relationships within the data, graph databases are an excellent storage solution due to the nature of their data structure. Therefore, graph databases are well suited for processing highly connected data. For these reasons, this kind of databases is also very fast when dealing with associative datasets, like in the case of social networks.

Another important advantage of graph databases is their more direct mapping, compared to other types of databases, to object-oriented applications.

To query a graph database, the most common approach is graph matching. Each query specifies a subgraph the users want to look for within the database. The database is explored, searching for subgraphs that match the query. The matching subgraphs are then retrieved and presented to the users as results.

The most popular graph database, as well as the one employed in this thesis, is Neo4j [23] [35] [38].

Neo4j is an open-source graph database, implemented in Java. Neo4j is schema-free: data is not required to respect predefined conventions. This property offers flexibility by allowing nodes and relationships to be created and modified freely, without having to consider the structure of already stored data. Neo4j is fully ACID-compliant.

Neo4j stores data in edges and nodes, as described before. Both nodes and relationships are identified by unique IDs. The primary purpose of Neo4j is to be an operational database, it was not designed specifically for analytics.

Cypher is the query language of Neo4j. It is used both for querying a Neo4j graph and for updating it. Cypher is a declarative language loosely inspired by SQL, but optimized for graph databases. It prioritizes the clarity of expressing what to retrieve from the graph and the readability of the queries, rather than focusing on the specific details of how the information is to be retrieved.

2 | Methodology

Data preparation is a complex process, involving a large variety of techniques and issues, and it may be hard for a user, especially for a non-expert, to navigate through this heterogeneous and usually long path. Furthermore, many alternative data preparation actions can be selected, often also executable in various ways, and an inexperienced user may have difficulty choosing what to do. The methodology presented in this thesis aims to guide the users through the data preparation process and help them select the best actions to satisfy their specific needs.

It is important to consider the particular user's needs because, depending on them, different data preparation actions might be appropriate. Therefore, the methodology considers the users' context (i.e., the data they have and the kind of analysis they want to perform) and suggests a data preparation pipeline appropriate for that scenario.

In Section 2.1, the architecture on which this methodology is based is described in detail.

2.1. Architecture

The architecture's main "use case scenario" is simple: the users have a dataset and want to use it for analysis purposes, and the architecture guides them on a data preparation pipeline optimal for that dataset and for the kind of analysis they want to perform.

A high-level representation of the overall architecture is shown in Figure 2.1. Note that this thesis focuses on the Data Preprocessing section of the schema.

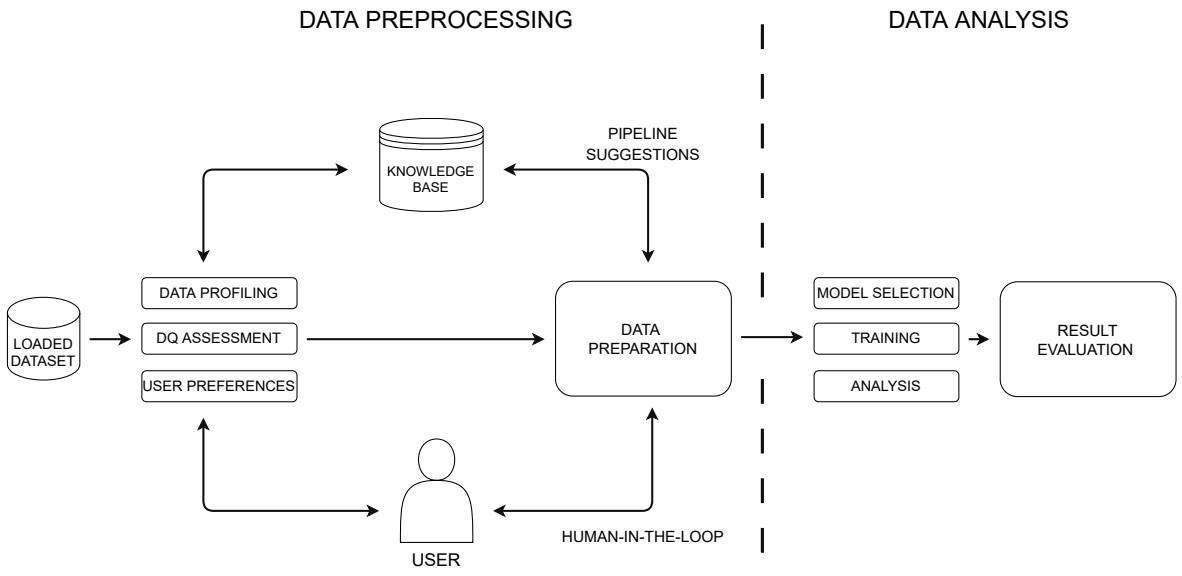

Figure 2.1: Architecture

The architecture relies on a Knowledge Base (KB), that contains the information needed to support the data preparation process. A detailed description of the KB can be found in Section 2.2.

At the beginning of the process, the users load the Dataset they want to prepare and select a Machine Learning Application they want to execute as a goal of their analysis of the dataset. As mentioned, the objective of the architecture from this moment on is to suggest actions to the users to prepare that dataset to optimize the performance of the selected application. Once the dataset is loaded, its Data Profile is computed. Data Profile contains a set of characteristics of the dataset (e.g., number of tuples, number of attributes, percentage of missing values, etc.) called Data Profile Features, which are relevant during the preparation phase. To compute the Data Profile, a series of analyses, called Data Profiling Techniques, are performed on the dataset, each of them calculating one or more Data Profile Features. For example, a certain Data Profiling Technique will calculate the percentage of missing values in the dataset. The set of all the Data Profile Features of a dataset, constitutes the Data Profile of that dataset.

During the profiling process, a first assessment of the Data Quality Dimensions is conducted. One quality dimension at a time is considered individually and its level of quality is measured. Taking into account the results of the quality assessment (i.e., the level of quality of each dimension), the selected ML application, and the characteristics of the loaded dataset, a ranking of the data quality dimensions is produced. This ranking indi-

cates the optimal order in which it is best to improve the quality dimensions in the users' context. Improving the quality dimensions following the obtained ranking, rather than a different order, leads to the achievement of better results, i.e., higher performance of the ML application. Note that the process in which this ranking is generated is not the primary focus of this thesis, but it is instead the result of a previous study [30].

At this point, the actual data preparation phase begins. The data preparation process can be divided into several segments, each of them focusing on a different set of actions. In particular, for each Data Quality Dimension considered, there is one segment that focuses on the improvement of that dimension, and then there is an additional segment (called "ML Application-oriented" segment) that contains actions specifically aimed to improve the execution of the ML Application.
Note that the order in which the segments are presented to users follows a specific sequence: initially, the segments focused on the improvement of the quality dimensions are presented, in an order respecting the previously established dimensions' ranking; subsequently, the ML Application-oriented segment is presented as the concluding one.

In each segment, a list of possible actions is proposed: these actions are called Data Preparation Techniques and represent possible operations that can be applied to the data. Often there is not a unique way to execute in practice a technique, but they can be implemented in several ways, called Data Preparation Methods. Therefore, it has to be decided what techniques to perform and with which methods.

The execution of a technique and/or of a specific method may be subject to one or more conditions on Data Profile Features: this happens when that technique (or method) can be applied only if some conditions on the dataset are verified. Before proposing techniques (and methods) to the users, these conditions are checked, and only the techniques/methods that pass these checks are actually suggested. Therefore, it is evident that, also during the data preparation phase, the data profile is constantly consulted.

Considering the ML Application-oriented segment, the techniques proposed in this segment are specifically selected for the particular ML Application chosen at the beginning by the users.

Furthermore, every time a technique is selected, the architecture can give the user an indication about the best method to implement that technique with, considering the user's specific context (dataset + ML application selected). The details of how this works will be explained afterward (see Section 2.3).

Because of these architecture functionalities, the suggestions made to the users are cus-

tomized to their needs and goals.

This architecture uses a Human-In-The-Loop approach. During the course of the entire process of data preparation, the users are constantly involved, at various levels. First of all, the users can, of course, accept the suggestions they receive, but they can also reject them. None of the recommended techniques and methods are mandatory: the users, if not satisfied with them, can choose independently the data preparation pipeline to perform. Complete freedom is always guaranteed.
Moreover, the users are involved also at execution time: during the execution of some techniques, the architecture can ask the users feedback and input, if needed.

The users' choices, which, as mentioned, may be different from the actions suggested, are stored in the Knowledge Base. In this way, it is possible to create a history of users' preferences to understand the most common choices and their differences from the suggested actions. This data can also be leveraged to adjust the suggestions to align them with past choices made by users in similar contexts: in particular, if a user rejects a suggestion several times, that suggestion will be adjusted and aligned to the actions the user is choosing instead. This basic learning mechanism will be refined in future work.

Note that the Knowledge Base plays a fundamental role in the entire process, from data profiling until the end of data preparation. All the concepts discussed and the relationships between them are stored in the KB and retrieved when needed. Therefore, all the data preparation techniques and methods, their dependencies with the data profile features, etc., are all part of the stored knowledge. During the data preparation phase, the KB is continuously queried to propose the appropriate techniques and methods for the specific context of the users.

2.2. Knowledge Base

The Knowledge Base (KB) contains knowledge about the various aspects of data preparation; in this way, it is possible to guide the users during the data preparation process. Furthermore, the KB contains information about the loaded datasets and their features, and it also stores the data preparation actions actually chosen by the users in the past to maintain a history of the users' choices.

Figure 2.2 shows the knowledge base schema at the conceptual level.

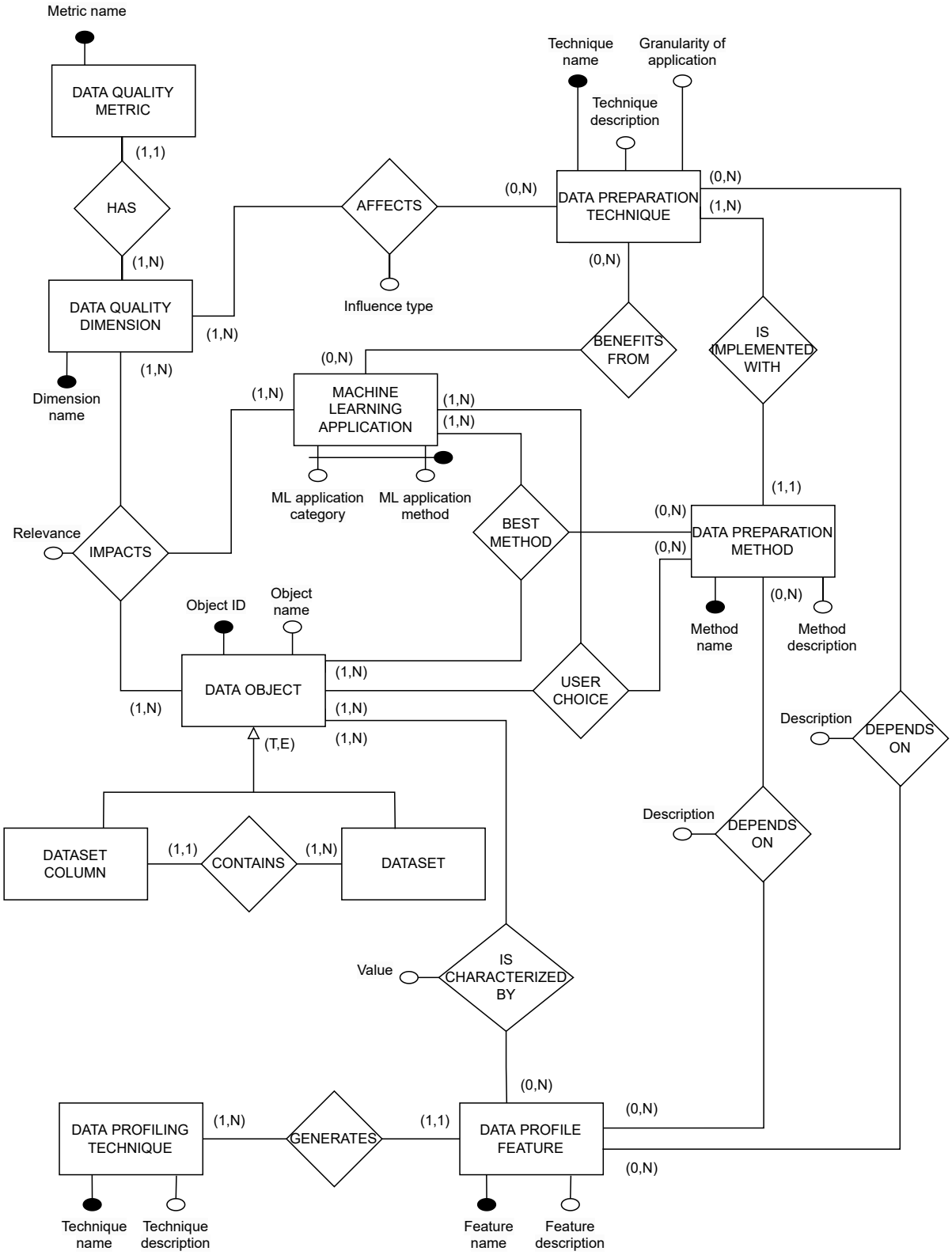

Figure 2.2: Knowledge Base Conceptual Schema

In the following paragraphs, the contents of the KB are listed and explained, along with their corresponding formal description.

Definition 1. A ***Data Object*** $do \in DO$ represents a set of data loaded by the users. Each do is characterized by a name and an id. The id is needed to give the possibility of storing different data objects with the same name. A data object can represent a Dataset (DS) or a Dataset Column (DSC), and this is expressed in the conceptual schema with an ISA hierarchy, exclusive and total (because a data object can only be a column or a dataset, not anything else and not both).
A dataset $ds \in DS$ contains one or more dataset columns $dsc \in DSC$, while each column belongs to one and only one dataset.

Datasets and Dataset Columns mapping:
$\forall ds \in DS \; \exists DSC' \subseteq DSC \mid DSC' \neq \varnothing \wedge ds \rightarrow DSC'$
$\forall dsc \in DSC \; \exists! ds \in DS \mid dsc \rightarrow ds$

Definition 2. A ***Data Profile Feature*** $dpf \in DPF$ indicates a feature that a data object can have. A data object is characterized by one or more Data Profile Features. Each data profile feature has a name and a description. Possible examples of Data Profile Features are the number of tuples of a dataset, or the number of distinct values in a column. As evident in the conceptual schema in Figure 2.2, the value that feature dpf takes in the specific case of data object do is not a property of the feature alone, but it is a property of the relationship between the two concepts do and dpf. Each data profile feature characterizes zero, one or more data objects.

The set of data profile features that characterize a data object constitutes the data profile of that object.

Data Objects and Data Profile Features mapping:
$\forall do \in DO \; \exists DPF' \subseteq DPF \mid DPF' \neq \emptyset \wedge do \rightarrow DPF'$
$\forall dpf \in DPF \; \exists DO' \subseteq DO \mid dpf \rightarrow DO'$

Definition 3. A ***Data Profiling Technique*** $dpft \in DPFT$ is a technique that performs the profiling of data objects and returns as output the values of some data profile features. Every data profile feature is generated by one and only one data profiling technique. Each $dpft$ is characterized by a name and a description, and generates one or more data profile features.

Data Profiling Techniques and Data Profile Features mapping:
$\forall dpft \in DPFT \; \exists DPF' \subseteq DPF \mid DPF' \neq \varnothing \wedge dpft \rightarrow DPF'$
$\forall dpf \in DPF \; \exists! dpft \in DPFT \mid dpf \rightarrow dpft$

Definition 4. A ***Data Preparation Technique*** $dpt \in DPT$ is a possible technique that the users can apply to their data during data preparation. Each *dpt* is characterized by a name, a description, and a granularity of application, indicating whether this technique must be applied on a whole dataset, a single column, or can be applied in both cases.

Definition 5. A ***Data Quality Dimension*** $dqd \in DQD$ is a concept that captures a specific data quality aspect. A data preparation technique can affect one or more Data Quality Dimensions. The influence of a *dpt* towards a *dqd* can be of various types: a *dpt* can, of course, bring an improvement to a *dqd*; however, in some cases, it can also bring a worsening (usually as a side effect). Note that in the presented architecture it is assumed that, although a data preparation technique can affect multiple data quality dimensions, the main improvement provided is toward a single dimension.
A data preparation technique may also not affect any data quality dimension, while each data quality dimension is affected by one or more techniques.

Data Preparation Techniques and Data Quality Dimensions mapping:
$\forall dpt \in DPT \; \exists DQD' \subseteq DQD \mid dpt \rightarrow DQD'$
$\forall dqd \in DQD \; \exists DPT' \subseteq DPT \mid DPT' \neq \varnothing \wedge dqd \rightarrow DPT'$

(Note that in the first formula, since DQD' is a subset of DQD and there is no specific condition, DQD' can also be empty. This also applies for the following similar formulas.)

Definition 6. A ***Data Quality Metric*** $dqm \in DQM$ expresses how a data quality dimension *dqd* can be assessed. Every *dqd* has one or more data quality metrics. Each data quality metric *dqm* is obviously related to one *dqd*.

Data Quality Dimensions and Data Quality Metrics mapping:
$\forall dqd \in DQD \; \exists DQM' \subseteq DQM \mid DQM' \neq \varnothing \wedge dqd \rightarrow DQM'$
$\forall dqm \in DQM \; \exists! dqd \in DQD \mid dqm \rightarrow dqd$

Definition 7. A ***Machine Learning Application*** $mla \in MLA$ is an application that the users can choose as their goal analysis. A data preparation technique can benefit zero, one or more ML Applications. Every *mla* is characterized by a macrocategory, which indicates the kind of ML application considered (e.g., clustering, classification or regression), and by a ML application method, the method that actually implements that application.
A *mla* can benefit from zero, one or more *dpt*.

Data Preparation Techniques and ML Applications mapping:
$\forall dqt \in DQT \; \exists MLA' \subseteq MLA \mid dqt \rightarrow MLA'$
$\forall mla \in MLA \; \exists DQT' \subseteq DQT \mid mla \rightarrow DQT'$

Definition 8. A ***Data Preparation Method*** $dpm \in DPM$ indicates a possible way by which a data preparation technique can be performed in practice. Every data preparation technique is implemented with one or more data preparation methods. Each data preparation method is characterized by a name and a description, and implements one and only one data preparation technique.

Data Preparation Techniques and Data Preparation Methods mapping:
$\forall dpt \in DPT \; \exists DPM' \subseteq DPM \mid DPM' \neq \varnothing \wedge dpt \rightarrow DPM'$
$\forall dpm \in DPM \; \exists! dpt \in DPT \mid dpm \rightarrow dpt$

A data preparation technique *dpt*, or a data preparation method *dpm*, can depend on one or more data profile features. This means that the execution of that technique, or method, can only take place if the considered *dpf* meets a certain condition. The description of this condition is stored in the KB as a property of the relationship between the technique, or method, and the data profile feature.
A *dpt*, or *dpm*, may also not depend on any feature. A *dpf* can be related to zero, one or more techniques, and/or methods.

Data Preparation Techniques / Methods and Data Profile Features mapping:
$\forall dpt \in DPT \; \exists DPF' \subseteq DPF \mid dpt \rightarrow DPF'$
$\forall dpf \in DPF \; \exists DPT' \subseteq DPT \mid dpf \rightarrow DPT'$
$\forall dpm \in DPM \; \exists DPF' \subseteq DPF \mid dpm \rightarrow DPF'$
$\forall dpf \in DPF \; \exists DPM' \subseteq DPM \mid dpf \rightarrow DPM'$

Dynamic Data

Among the concepts listed in the previous section, those needed for the basic functioning of the architecture can be called Static Data. For instance, all the data preparation techniques, methods, data quality dimensions and machine learning applications are static data. This data is the core of the whole architecture, and without it, it is not possible to make any suggestions to the users. Static data can be updated from time to time to add, for example, the latest preparation methods or a new machine learning application. However, it is not meant to be changed often.

There is also another kind of data stored in the knowledge base that can be called Dynamic Data. Unlike static data, this data is created through experiments and through the usage of the architecture. Because of the nature of this kind of data, dynamic data is much more frequently added and updated in comparison with static data. The dynamic data stored in the KB consists of the loaded data objects, along with the values of their profile features, and the data relative to the three ternary relationships present in the KB's

conceptual schema: Impacts, Best Method and User Choice.

The data objects and the values of their features are dynamic data because they are created through the usage of the architecture: in particular, the data objects are loaded by the users, while the value of the features is calculated by the data profiling techniques once the object is loaded.

The three ternary relationships Impacts, Best Method and User Choice enable the storage of additional knowledge in the KB and are described in detail in the following paragraphs.

The first ternary relationship, Impacts, is needed to store knowledge about the impact that each data quality dimension has in a certain context, i.e., with a certain ML application and data object. In this way, it is possible to store how relevant each data quality dimension is in a given situation. The Relevance property of the relationship is a measure of the importance of that data quality dimension in that context.
Note that since a data object can either be a whole dataset or a single column, it is possible to store the impact of a data quality dimension both at the dataset level and column level.
This knowledge regarding the impact that quality dimensions have on different contexts represents dynamic data, because has been extracted through experiments conducted in a previous work [30].

The second ternary relationship, Best Method, is used to store knowledge about which data preparation method is better to use with a given data object and ML application. Since, as previously mentioned, a data preparation technique can be implemented with various methods, with this ternary relationship, it is possible to save which method is the best for that technique in a specific context.
Note that the cardinality of the relationship is (0,N) near data preparation method because there could be a method that is never the best in all the contexts stored in the KB. Furthermore, because of this cardinality, it is also possible to indicate that a method is the best for a given data object regardless of the ML application selected.
As in Impacts, also in this relationship a data object can represent a dataset or a single column, therefore a method can be appointed as best method either for a column or a dataset.
Also this knowledge represents dynamic data, and was extracted through experiments. In particular, the knowledge regarding the best Outlier Detection method to use with different data objects was extracted in [10]. Instead, the knowledge regarding the best Imputation method to use in different contexts (data object + ML application) was generated in this thesis, details in Section 4.1.

The third ternary relationship, User Choice, is similar to the previous relationship, but while Best Method is used to store the best method for a given context, User Choice allows to store what methods were actually selected by the users in that context. Given a selected ML application and data object, the users will perform a series of operations on the data, and with this ternary relationship it is possible to store all the data preparation methods they used. Since a data preparation method implements one and only one data preparation technique, from this knowledge, it is of course easy to identify the used techniques too. This relationship allows to build the history of the methods and techniques selected by past users in various contexts, and it gives the possibility of extracting useful information, like the most common users' choices.
Note that also in this relationship there is cardinality (0,N) near data preparation method, because there could be a method that users have never chosen before.
This knowledge represents dynamic data because it is created through the use of the architecture by the users.

2.3. Classifier

As previously mentioned, in the Knowledge Base there is information about the best method to use in a particular context, i.e., with a certain data object and ML application. In particular this knowledge is represented using the ternary relationship Best Method.
One of the goals of this thesis was to find a way to exploit this knowledge to suggest to the users the best methods to use in their specific analysis context, in order to have more reliable results. Considering a single data preparation technique, the problem can be more precisely specified in the following way: the user selects a data object, that can either be a whole dataset or a single column, and a ML application, and the objective is to return the best data preparation method with which to implement that technique in the selected context.

To reach this goal, in this thesis the approach followed is to use supervised learning, a branch of Machine Learning. More specifically, the problem has been moved to multi-class classification. The target variable, i.e., the class to be predicted, is the best data preparation method to use in a given context. Therefore, in this thesis, a classifier has been created, and trained on the data stored in the KB. In particular, it is possible to obtain a training set for the classifier from the KB using the ternary relationship Best Method, considering the data profile features of the data object and the ML application as features of the training set and the best data preparation methods as labels. Once trained, the classifier can predict the best data preparation method to use in a new context, taking

in input the data profile features of a new data object and a chosen ML application. Thereby, the process of suggesting what method to use to the user (still considering a single data preparation technique) can be summarized in this way: the user loads a new data object and selects a ML application, then from the data object are extracted its data profile features, and at this point the data profile of the object and the ML application are given in input to the trained classifier, that returns in output the best method to implement the technique in that specific situation.
This process is illustrated in Figure 2.3.

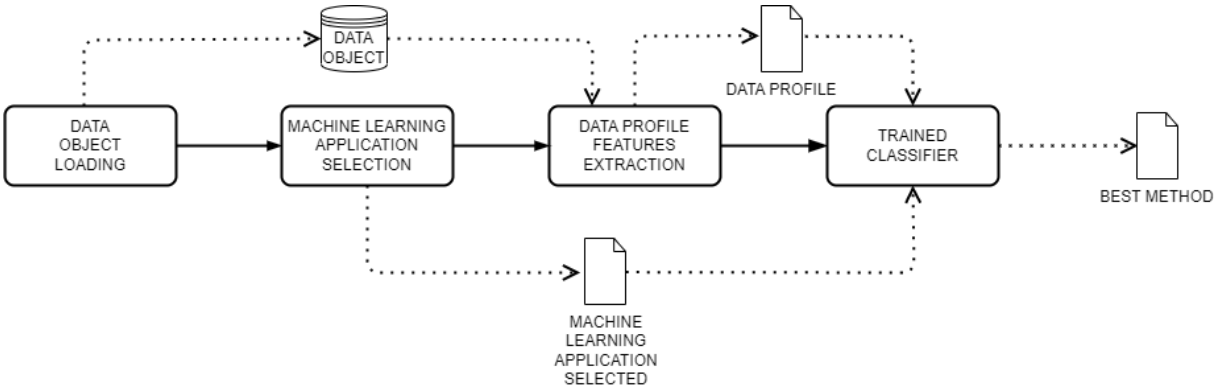

Figure 2.3: Classifier Schema

Since a data object can be either a whole dataset or just a single column, in this work it was opted to build two different classifiers, one for each case, to try to achieve more fitting suggestions. In the whole dataset case, the classifier takes in input the data profile of that dataset, while in the single column case, it takes in input not only the data profile of that column, but also part of the profile of the dataset to which the column belongs. This last design choice was made because, to suggest a method in the case of a single column, it might also be important to consider how the dataset from which the column comes is made.

Regarding the practical implementation part of this concepts, this work is focused on one data preparation technique: Imputation. Two classifiers have been actually implemented, one for the single-column case and one for the whole dataset case. These classifiers work as previously described: they take the data profile of a data object (column or dataset) and an ML application, and return in output the best imputation method to use to impute the missing values in that context.
For more details on the implementation of these classifiers, see Chapter 4.

3 | Knowledge Base Implementation

This chapter describes the implementation of the Knowledge Base presented in Section 2.2. This implementation aims to be as close as possible to the conceptual schema shown in Figure 2.2. To implement the KB it was decided to use a graph database, in particular, Neo4j.

The following paragraphs give some examples of how the knowledge is stored in the database. The data showed in the examples has been retrieved from the database using the Neo4j query language, Cypher.

Some of the fundamental concepts of the KB are data preparation techniques and methods. In Figure 3.1, it is possible to see how these concepts and the relationships among them are stored in the database.

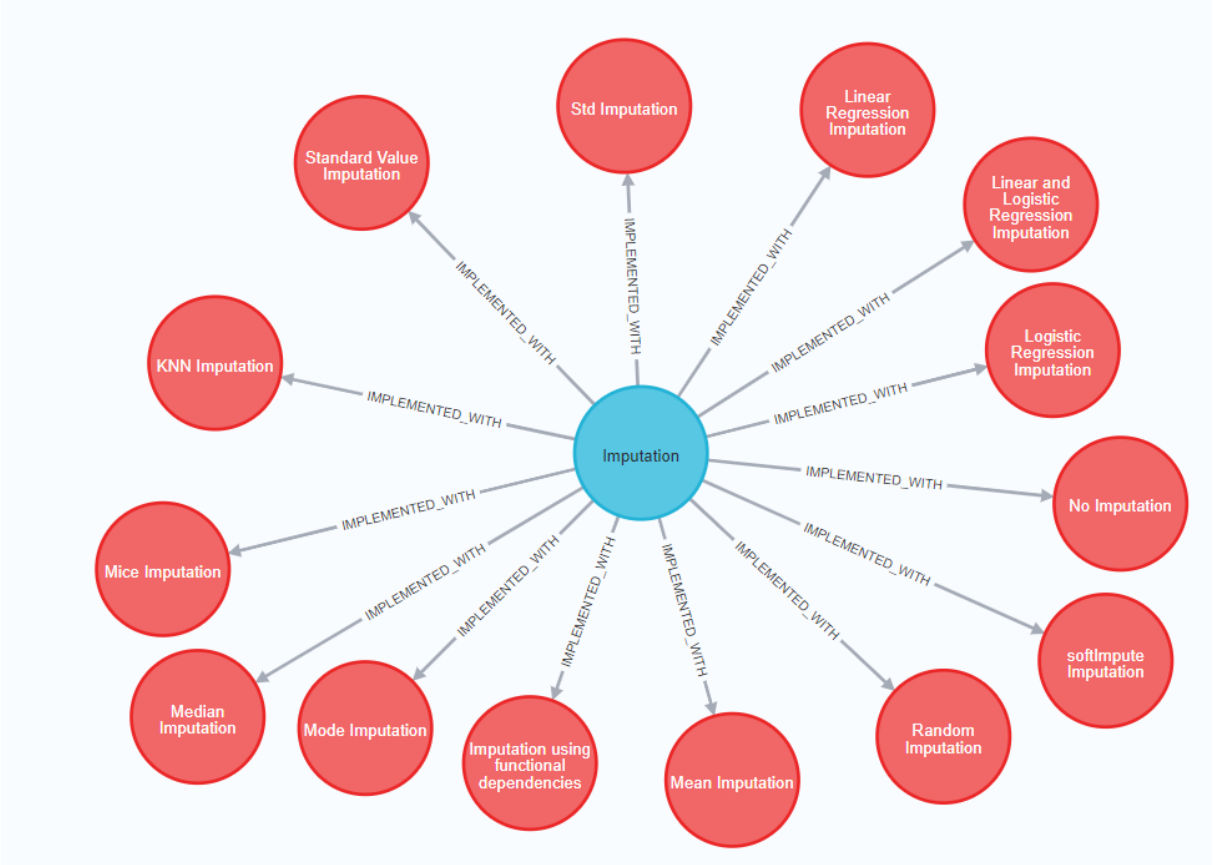

Figure 3.1: Imputation technique and methods

In Neo4j, a different color for the node means a different label (i.e., the type of the node), in this example light blue is used for the techniques and red for the methods. The type of a relationship is clearly written on the edge that represents that relationship.
In this example, it is possible to see a data preparation technique (here Imputation has been selected) and the data preparation methods that implement that technique. Both the technique and the method nodes have properties. The text displayed on the nodes is the name of the method/technique that each node represents. To show the other properties of a node, one has to click on the desired node. Clicking on the node "Imputation" gives the result visible in Figure 3.2: the node label, the node ID, the technique name, and its granularity of application are displayed.

Node properties

DATA_PREPARATION_TECHNIQUE

<id>	130
granularity_of_application	Both
name	Imputation

Figure 3.2: Imputation technique properties

As it is shown in the conceptual schema, a technique can affect a data quality dimension. This is illustrated in the example in Figure 3.3.

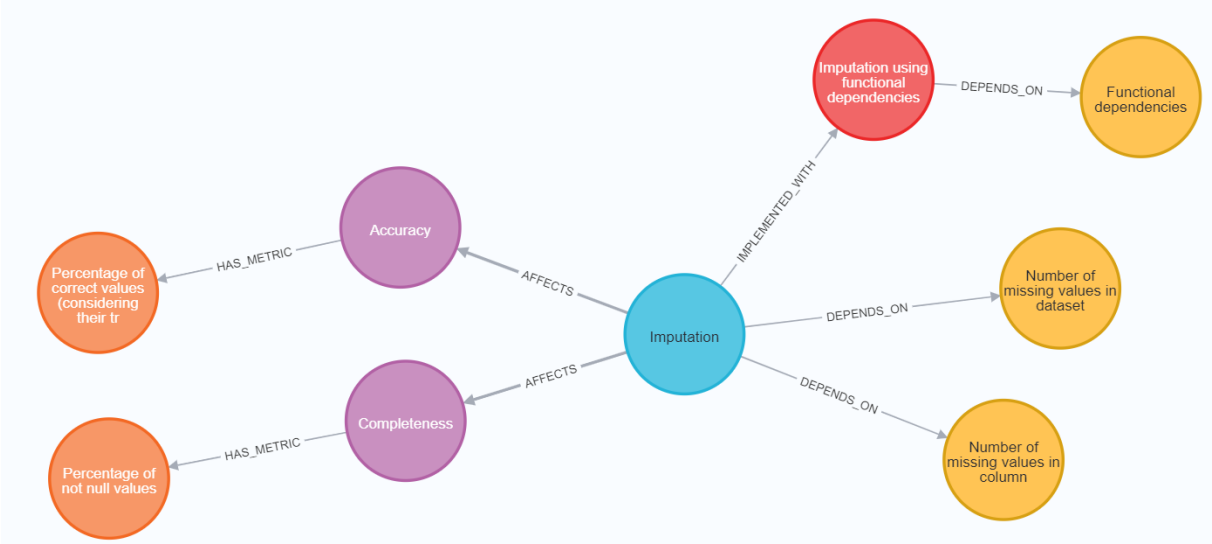

Figure 3.3: Imputation technique, dimensions and profile features

In this example, it is evident what dimensions (in purple) are affected by the Imputation technique. As previously mentioned, a technique can affect a dimension in various ways: it can bring an improvement or a worsening to the dimension. This information is stored in the "influence type" property of the relationship "AFFECTS". As with nodes, to show the properties of a relationship, it is necessary to click on that relationship. For instance, selecting the relationship between Imputation and Completeness gives the result of Figure 3.4.

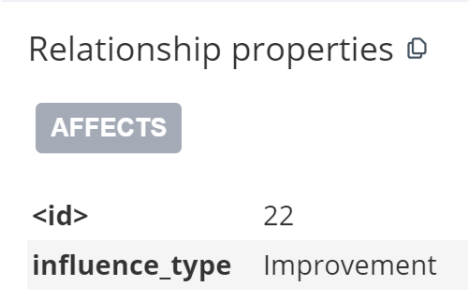

Figure 3.4: AFFECTS relationship properties

So, this is how in the KB it is stored that the technique Imputation improves the dimension Completeness.

In Figure 3.3 also other important concepts of the KB are displayed. It is possible to see how a dimension is related to (at least) one metric (in orange). Furthermore, as explained during the conceptual description of the KB, both techniques and methods can depend on some data profile features (in yellow in the picture). It is shown how "Imputation" depends, obviously, on the number of missing values of the data object considered, while one of the imputation methods, "Imputation using functional dependencies", depends on whether or not functional dependencies are present. The description of these dependencies is stored in the property "description" of the "DEPENDS ON" relationship. For instance, selecting the relationship between the method "Imputation using functional dependencies" and feature "Functional dependencies" gives the result in Figure 3.5.

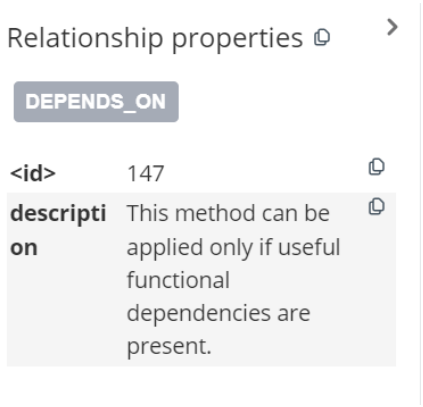

Figure 3.5: DEPENDS ON relationship properties

The example in Figure 3.6 focuses on another data preparation technique: Normalization.

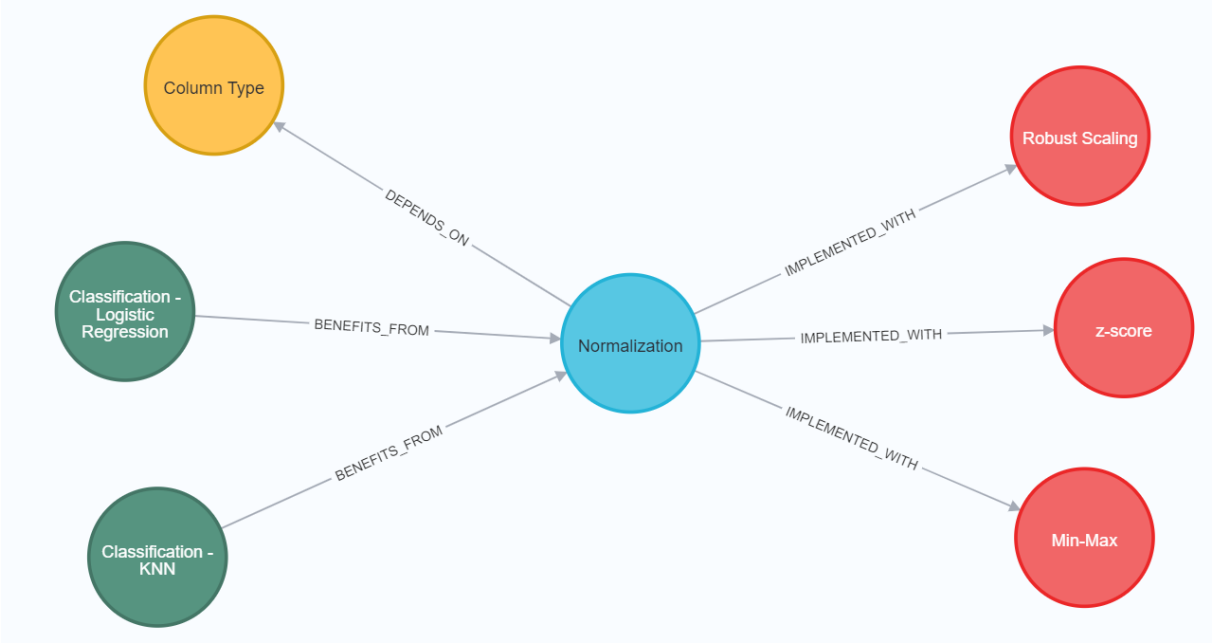

Figure 3.6: Normalization technique and ML applications

Here, it is possible to see how a technique can benefit one or more machine learning applications. An application benefits from a data preparation technique if the execution of that technique can improve the overall performance of the application. A ML application (in green) is characterized by a ML category and a method, both displayed on top of the nodes, separated by a dash. This example shows how the applications Logistic Regression and KNN benefit from the Normalization technique. Moreover, it shows some possible methods that implement this technique, as well as a dependency with the feature "Column Type". This dependency is present because the normalization technique is only applicable to numerical columns (this is specified in the "description" property of the relationship, as in the previous examples).

The following example, in Figure 3.7, moves the attention to the concept of Data Object and the relationships it has with other parts of the KB. Note that in the implemented database there are no nodes with the label "Data Object", because the data objects are already divided with labels "Dataset" and "Dataset Column" (respectively in brown and in dark pink).

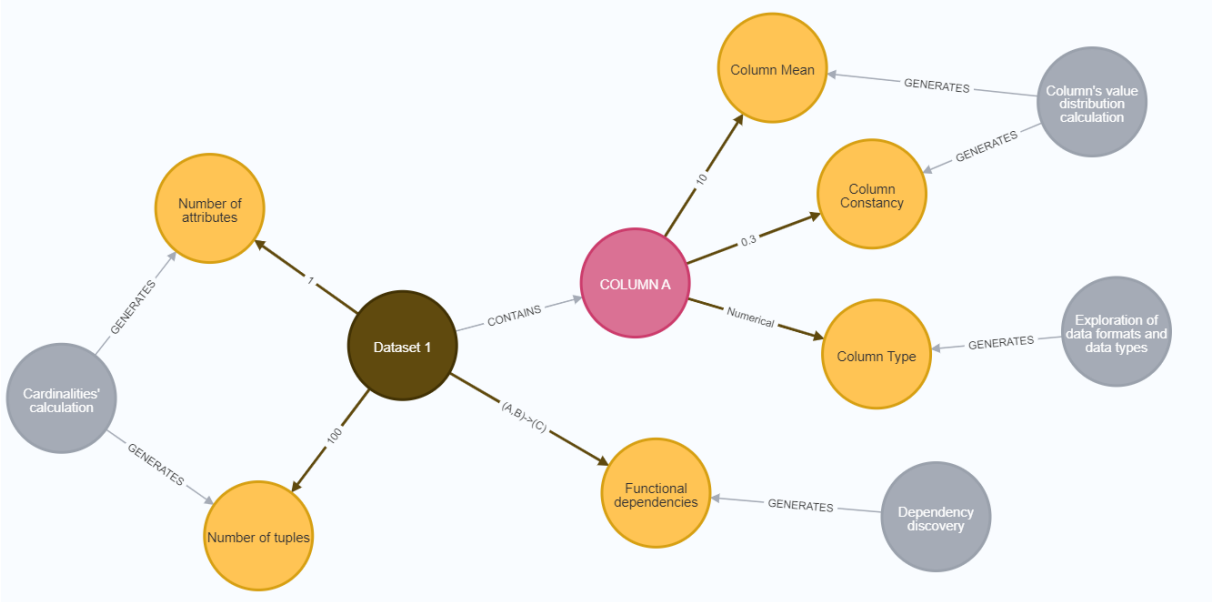

Figure 3.7: Data objects and profile features - Example 1

In Figure 3.7 there is a toy dataset, called "Dataset 1", related to a column with name "COLUMN A" through the relationship CONTAINS. Both the dataset and the column are characterized by some profile features. The type of the relationship between the data objects and the features is "IS CHARACTERIZED BY", like in the conceptual schema, but for visualization purposes on the edge representing that relationship is written the "Value" property: this indicates the value taken by that feature in the case of that data object. For instance, the selected dataset is characterized by a certain number of tuples, which is 100. Furthermore, in this example, it is shown how each data profile feature is generated by a data profiling technique (in grey), while a profiling technique can also generate more than one feature (see, for instance "Cardinalities' calculation").

A more realistic example, with a real dataset and more profile features, can be found in Figure 3.8:

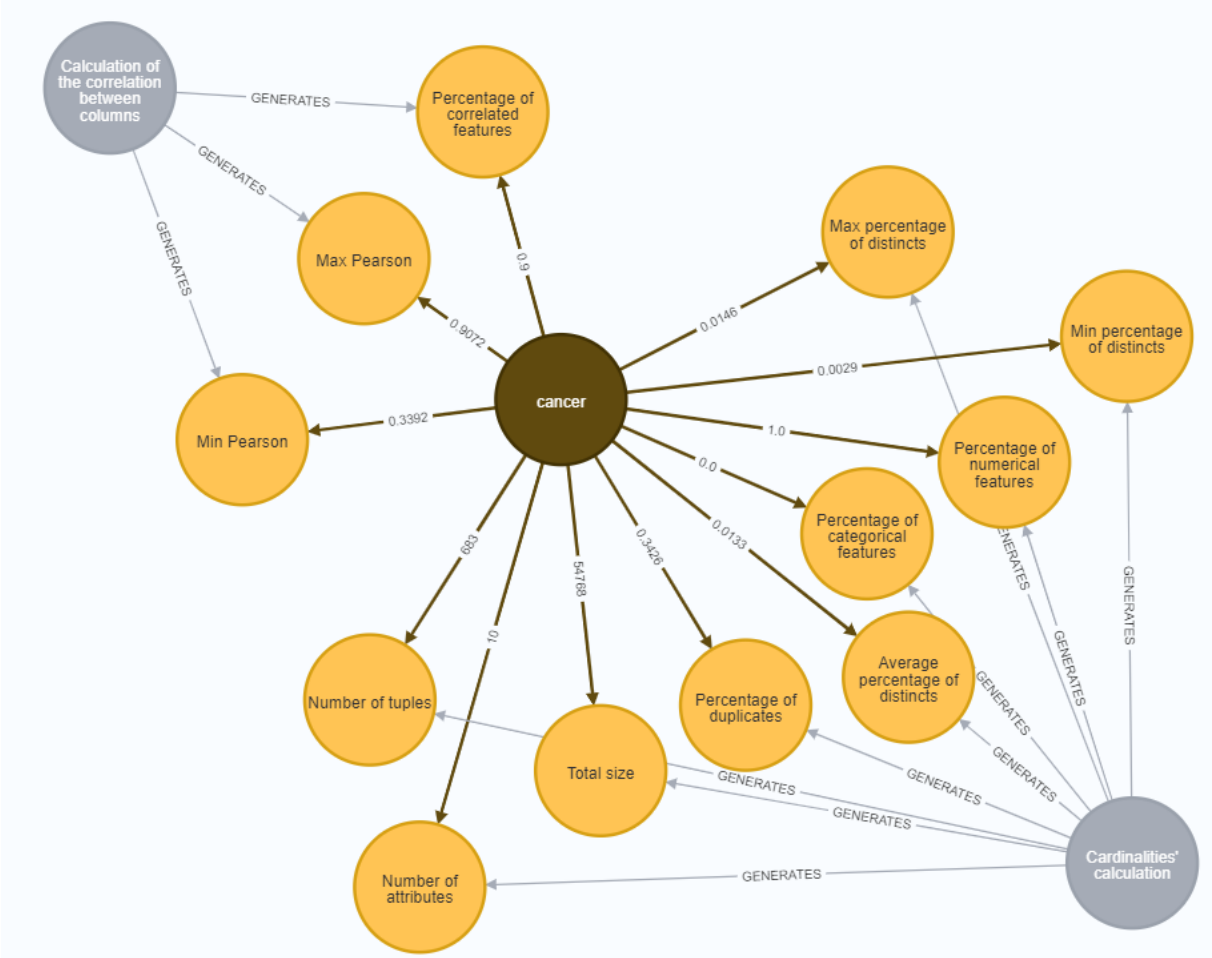

Figure 3.8: Data objects and profile features - Example 2

The KB part that stores the impact of data quality dimensions in different contexts is represented in the conceptual schema with a ternary relationship between dimensions, machine learning applications and data objects. However, in Neo4j ternary relationships are not natively supported. Therefore, intermediate nodes were used to implement this KB section. An example of this implementation is shown in Figure 3.9.

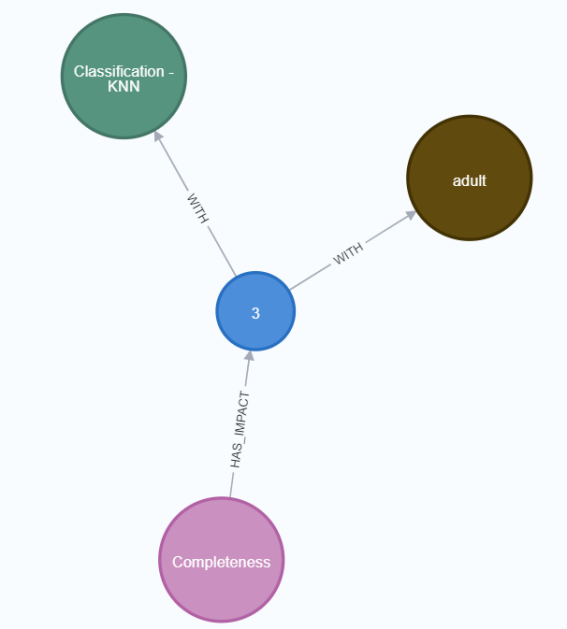

Figure 3.9: Data quality dimension's impact - Example 1

In this example, both a quality dimension and a context (ML Application + Data Object) have been selected and it is possible to see the impact that dimension has on that context. There is an intermediate node (in blue), with label "IMPACT" and property "relevance" displayed on top. An intermediate node is connected with a dimension through a relationship of type "HAS IMPACT" and with a data object and an application through relationships of type "WITH". So, in this example, it is evident that dimension "Completeness" has an impact of relevance 3 with context formed by dataset "adult" and ML application "Classification - KNN". The relevance property is an indicator of how strong the impact of the dimension is: the lower the value, the stronger is the impact. For instance, if within a certain context Completeness has an impact of relevance 1 and Accuracy has an impact of relevance 2, it implies that the Completeness dimension is more relevant and impactful in that particular context.

Writing different queries makes it possible to interrogate the database in various ways. For instance, it is possible to retrieve what impact a certain dimension has with a specific dataset for all the possible ML applications. This is illustrated in Figure 3.10.

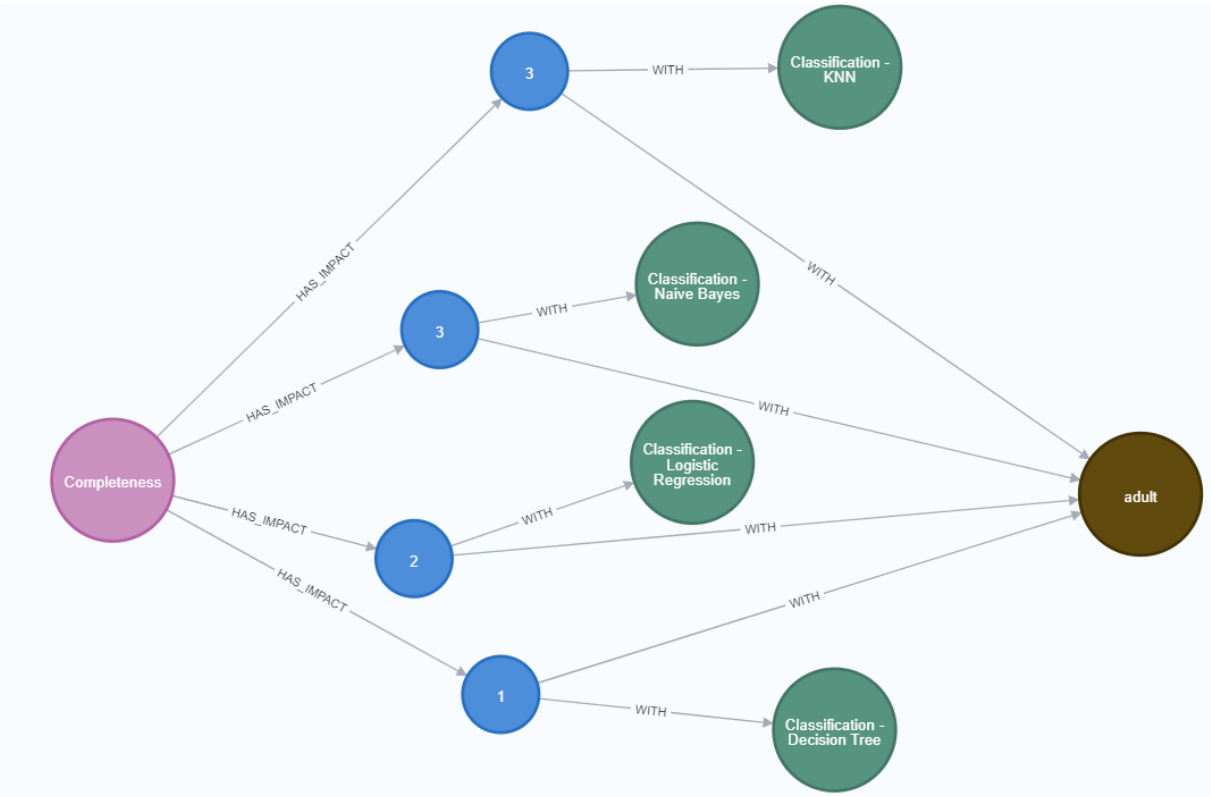

Figure 3.10: Data quality dimension's impact - Example 2

Note that here, it is clear that a dimension can have different impacts even with the same dataset, depending on the application selected.

A similar implementation has been used to store the knowledge about the best methods to use in different contexts. A first example can be found in Figure 3.11.

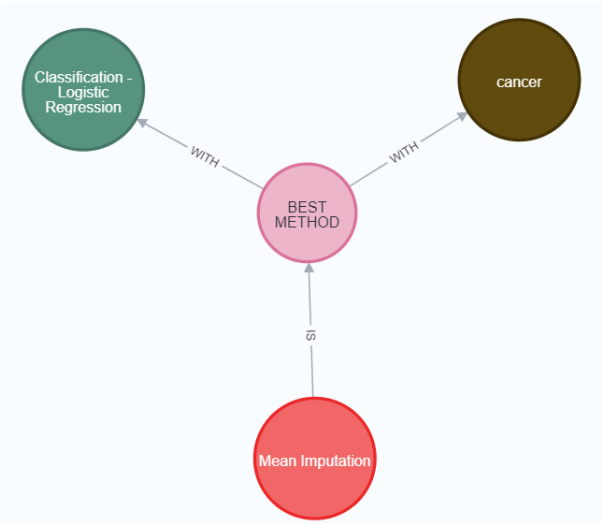

Figure 3.11: Best data preparation method - Example 1

Intermediate nodes were also used to implement this part of the KB. This kind of intermediate node (in pink) has label "BEST METHOD" and is connected to a data preparation method through the relationship "IS" and to a data object and a ML application through relationships "WITH". Recalling that each preparation method implements a single technique, this structure indicates that the connected method is the best way to implement the respective technique in the connected context (data object + ML application). Therefore, regarding the previous example, "Mean Imputation" is the best method to implement the technique "Imputation", in the context of dataset "cancer" and application "Classification - Logistic Regression".

In Figure 3.12, there is a similar example that highlights how one method may be the best not necessarily with respect to an entire dataset, but also just for a specific column. In fact, the KB contains information on both the best imputation methods to use for entire datasets as well as for individual columns. Furthermore, this example shows that even with the same data object, the best method may change depending on the ML application selected.

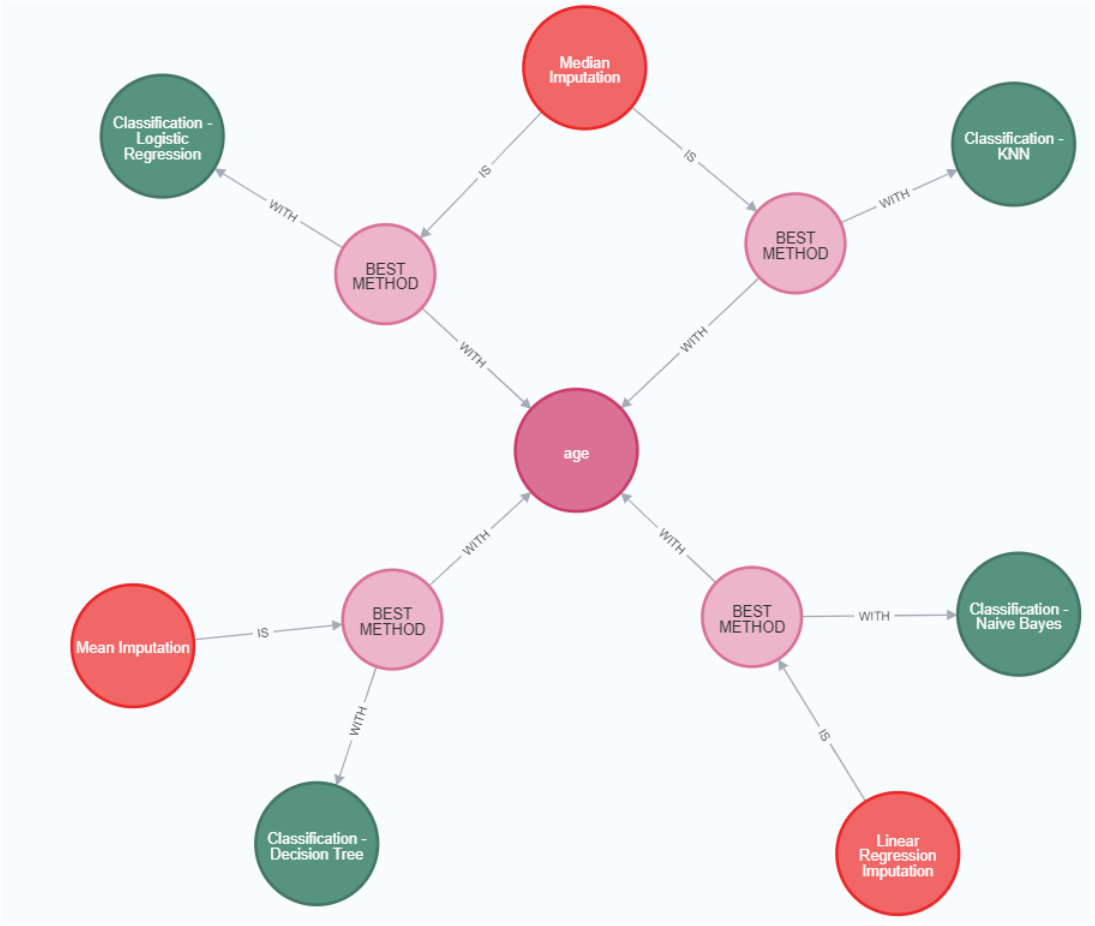

Figure 3.12: Best data preparation method - Example 2

In particular, Figure 3.12 shows the best methods for imputing missing values in the "age" column.

This knowledge regarding the best imputation method to use in different contexts was generated during this thesis, through a series of experiments. In these experiments, several ML applications and datasets with various characteristics were considered. The obtained knowledge was subsequently stored within the KB. The detailed process used to generate this knowledge is described in Section 4.1.

The previous examples are all focused on the Imputation technique, but, as mentioned in Section 2.2, in the KB there is also knowledge regarding the best Outlier Detection methods to use. An example of this knowledge is displayed in Figure 3.13.

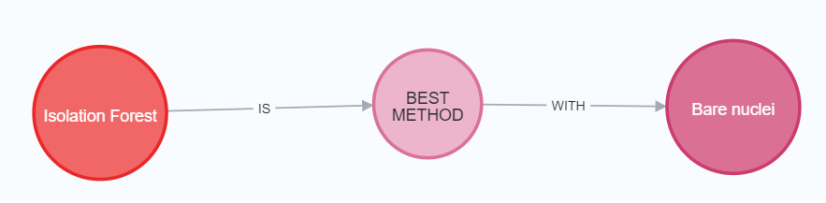

Figure 3.13: Best data preparation method - Outlier Detection

Note that, unlike the Imputation case, the best outlier detection method does not depend on the ML application selected but only on the data object.

The knowledge regarding the best outlier detection method to use with different data objects was generated in [10]. The process used for generating this knowledge can be summarized as follows: starting from clean data objects, artificial outliers with different levels of detection difficulty were injected into the data, obtaining "dirty" data objects containing anomalies; subsequently, several outlier detection methods were applied on these data object, recording how many of the injected outliers were detected by each method; based on the resulting performance, the best outlier detection method was determined for each data object.

Since the architecture has not been used by actual users, the history of users' choices is not yet present in the KB. When the architecture will be used, these choices will be stored in a manner similar to how best methods are stored.
An example of how users' choices will be stored can be found in Figure 3.14:

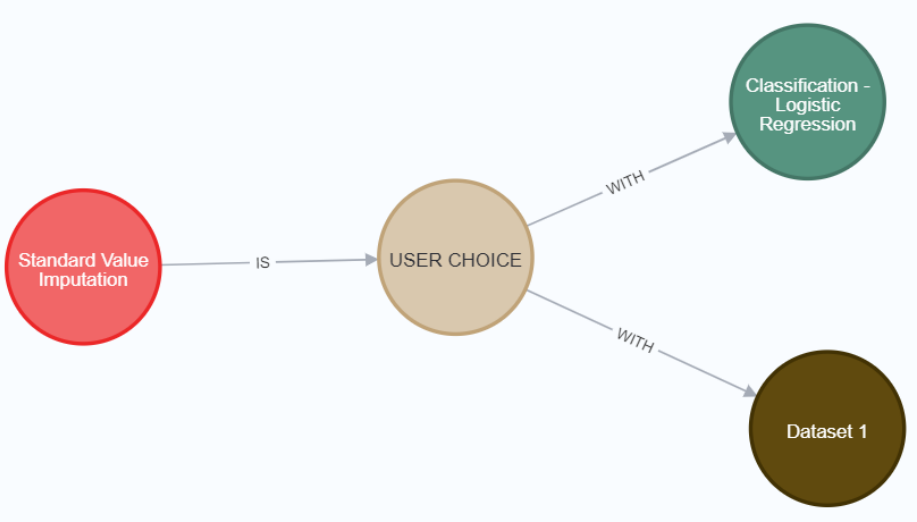

Figure 3.14: User choice

4 | Classifiers Implementation and Results

As described in Section 2.3, one of the goals of this thesis is to build a classifier capable of exploiting the knowledge present in the Knowledge Base to suggest the best data preparation method to use in the context of a certain data object and ML application. The classifiers implemented in this thesis are focused on the data quality dimension of Completeness, specifically on the missing values Imputation technique.

The classifiers receive as input the data profile features of the data object and the selected ML application, and return the best imputation method to fill in the missing values of that data object. It is a multiclass classification problem because the target variable to predict is clearly the best imputation method and several (more than two) possible methods exist.

Since a data object can either be an entire dataset or a single column, two separate classifiers have been developed in this thesis, one for each of the two cases. In the entire dataset case, the classifier predicts the best imputation method to impute all the missing values in the dataset, while in the single column case, it predicts the best method to impute the missing values of just that column specifically.

The following sections describe in detail how these classifiers were implemented and the results obtained.

4.1. Knowledge generation

The first important part of the work focused on the generation of the knowledge needed for the training and testing of the classifiers. Since the needed knowledge differs for the two classifiers, the two knowledge generation processes are similar but have several relevant differences.

Note that the knowledge generated by these processes is first stored in the Knowledge

Base, and then exploited through the classifiers. In fact, the idea is to populate the KB, and then the classifiers are fed and trained using the knowledge present in it. In particular, this knowledge is stored in the "Best Method" part of the KB.

4.1.1. Knowledge generation: entire dataset case

The knowledge to be obtained in the entire dataset case is of this form:
dataset profile, ML application → best imputation method to use for that dataset.

The overall process to obtain this knowledge is described in pseudo-code in Algorithm 4.1. A more detailed description of the process follows.

Algorithm 4.1 Knowledge Generation Process: Entire dataset case

```
for each clean_dataset in clean_datasets do
    data_profile_features ← Extract data profile features of clean_dataset
    for each missing_values_percentage in [50%, 40%, 30%, 20%, 10%] do
        incomplete_dataset ← Missing values injection into clean_dataset with missing_values_percentage
        imputed_datasets ← []
        for each imputation_method in imputation_methods do
            imputed_dataset ← Impute missing values in incomplete_dataset using imputation_method
            imputed_datasets.append(imputed_dataset)
        end for
        for each ml_algorithm in ml_algorithms do
            ml_performances ← []
            for each imputed_dataset in imputed_datasets do
                ml_performance ← Apply cross-validation on imputed_dataset with ml_algorithm
                ml_performances.append(ml_performance)
            end for
            best_imputation_method ← Determine the best imputation method based on ml_performances
            Store knowledge unit:
            (data_profile_features,
            missing_values_percentage,
            ml_algorithm,
            best_imputation_method)
        end for
    end for
end for
```

First of all, a clean, complete dataset is considered, and its data profile features are extracted and saved.

Then, there is an error injection phase, during which the dataset is injected with missing values in random positions. This injection is repeated many times, each time with a different percentage of missing values injected. In this way, several incomplete datasets are produced from a single clean dataset. Specifically, five datasets are produced, with 50, 40, 30, 20 and 10 percent of missing values.

At this point, the missing values of these incomplete datasets are imputed several times, each time with a different imputation method.

After applying an imputation method, the imputed dataset obtained is given in input to an ML algorithm, which is trained and tested (using cross-validation) on that dataset. The performance of the ML algorithm is used as an indicator of the effectiveness of the imputation method used to impute that dataset. A better performance of the ML algorithm indicates a better imputation method. In this way, it is possible to establish what is the best imputation method to use with a certain dataset and ML algorithm.

This last phase is repeated several times with a different ML algorithm to determine which is the best imputation method with that dataset and each of those ML algorithms.

The original dataset profile features stored at the beginning, the percentage of missing values of the incomplete dataset considered, the ML algorithm considered and the best method in that context are stored as a unit of knowledge. Therefore, from a clean dataset $5 * \#ML\ algorithms$ units of knowledge are obtained.

This whole procedure is repeated for each clean dataset to be considered.

This was a high-level description of the process by which the knowledge is generated. More precise and technical details can be found in the following.

The data profile features extracted at the beginning from the clean dataset summarize the characteristics of that dataset and are listed below:

- Number of tuples.
- Number of attributes.
- Percentage of numerical variables.
- Percentage of categorical variables.
- Percentage of duplicate tuples.
- Total size occupied in memory.
- Average percentage of distinct values in columns.

- Maximum percentage of distinct values in a column.
- Minimum percentage of distinct values in a column.
- Average density of values in columns.
- Maximum density of values in a column.
- Minimum density of values in a column.
- Average entropy of the columns.
- Maximum entropy of a column.
- Minimum entropy of a column.
- Percentage of columns considered to be correlated.
- Maximum Pearson correlation coefficient between two columns.
- Minimum Pearson correlation coefficient between two columns.

The imputation methods used to impute the missing values of the incomplete datasets are of various types; some are more "traditional" while others use machine learning techniques. The list of the employed methods is presented below (see Section 1.5 for the detailed descriptions of all the methods):

- Standard Value Imputation.
- Mean Imputation.
- Median Imputation.
- Mode Imputation.
- No Imputation
- KNN Imputation.
- Mice Imputation.
- softImpute Imputation.
- Random Imputation.
- Linear and Logistic Regression Imputation.

Regarding the ML algorithms considered, this thesis focuses only on classification; in particular, the algorithms used are Decision Tree, Logistic Regression, K-Nearest Neighbors, and Naive Bayes.

To check the performance of an ML algorithm on an imputed dataset, k-fold cross-validation is used, with a weighted f1 score as a performance indicator.

The clean datasets used for this knowledge generation process were selected so that they would have various characteristics: some of these datasets have only numerical attributes, some have only categorical attributes, and others are mixed with numerical and categorical attributes. In this way, the knowledge generated is not focused only on a specific kind of dataset. Furthermore, since as mentioned the ML algorithms considered focus on classification, all the datasets have a discrete class attribute to predict.

To make this process more robust, and thus the obtained results more reliable, parallelization was used: each experiment was repeated in parallel eight times, each time with the missing values injected in different positions of the clean dataset. Then, the scores obtained from each repetition were averaged, and the imputation method with the higher average score was selected as the best method.

An example of a unit of knowledge generated can be found in Figure 4.1 (for visualization purposes, the row is broken into several parts):

name	n_tuples	n_attributes	p_num_var
iris	150	5	1.0

p_cat_var	p_duplicates	total_size	p_avg_distinct
0.0	0.02	6128	0.168

p_max_distinct	p_min_distinct	avg_density	max_density
0.2867	0.02	0.0351	0.0644

min_density	avg_entropy	max_entropy	min_entropy
0.0	2.7058	3.4892	1.0986

p_correlated_features	max_pearson	min_pearson	%missing
0.8	0.9628	-0.4205	0.4002

ML_ALGORITHM	BEST_METHOD
dt	impute_standard

Figure 4.1: Example of the generated knowledge - Entire dataset case

This example shows the clean dataset considered (iris) with its data profile features, the percentage of missing values injected (displayed in the field "%missing"), the ML algorithm considered (decision tree) and the best imputation method found (standard

value imputation).

4.1.2. Knowledge generation: single column case

In the single column case, the knowledge to be generated concerns the best imputation method to apply to impute the missing values of a specific column of a dataset. In particular, the knowledge to obtain is of form:
dataset profile, column profile, ML application → best imputation method to use for that column.
Note that not only the column profile is considered, but also the profile of the entire dataset from which the column is from.

This knowledge generation process is similar to the one regarding the entire dataset case described in Section 4.1.1, but with some differences. The process is described in pseudo-code in Algorithm 4.2. A detailed description of the process follows.

As in the entire dataset case, a clean complete dataset is considered, but in this case, a feature selection technique is applied to it. This technique (more details below) selects the four most relevant features of the dataset and deletes the others (keeping obviously the class variable). The data profile features of this new dataset (composed of the four selected columns and the class) are extracted and saved.

At this point, one at a time, each of the four columns is considered independently: the data profile features of the column considered in that iteration are extracted, and then there is an error injection phase during which missing values are injected in random positions in that column. At the end of this phase, the dataset obtained is complete except for the missing values injected in the considered column. Note that, as in the entire dataset case, the error injection is repeated several (five) times with different percentages of missing values, thus from one complete dataset, several incomplete datasets are created. But differently from before, the missing values are not injected into the whole dataset but only in the considered column.

From this point on, the process continues in a manner similar to the entire dataset case: the missing values of the incomplete datasets (this time concentrated in only one column) are imputed with several imputation methods, and the best imputation method is determined by the performance of an ML algorithm executed on the imputed datasets. This last phase is repeated with several ML algorithms, as explained before.

This procedure is executed for each of the four columns of the dataset for each of the clean datasets to be considered. In this way, at the end of the process, the best methods for

Algorithm 4.2 Knowledge Generation Process: Single column case

```
for each clean_dataset in clean_datasets do
    reduced_dataset ← Apply feature selection on clean_dataset and keep the four most relevant columns and the class
    dataset_profile_features ← Extract dataset profile features of reduced_dataset
    dataset_columns ← List of columns in reduced_dataset (excluding class variable)
    for each column in dataset_columns do
        column_profile_features ← Extract column profile features of column
        for each missing_values_percentage in [50%, 40%, 30%, 20%, 10%] do
            incomplete_dataset ← Missing values injection into column with missing_values_percentage
            imputed_datasets ← []
            for each imputation_method in imputation_methods do
                imputed_dataset ← Impute missing values in incomplete_dataset using imputation_method
                imputed_datasets.append(imputed_dataset)
            end for
            for each ml_algorithm in ml_algorithms do
                ml_performances ← []
                for each imputed_dataset in imputed_datasets do
                    ml_performance ← Apply cross-validation on imputed_dataset with ml_algorithm
                    ml_performances.append(ml_performance)
                end for
                best_imputation_method ← Determine the best imputation method based on ml_performances
                Store knowledge unit:
                (dataset_profile_features,
                column_profile_features,
                missing_values_percentage,
                ml_algorithm,
                best_imputation_method)
            end for
        end for
    end for
end for
```

imputing missing values of particular columns drawn from different datasets are obtained.

The entire dataset profile features, the column profile features, the percentage of missing values injected in the column, the ML algorithm considered, and the best method in that context are stored as a unit of knowledge. Therefore, from a clean dataset $5 * 4 * \#ML\ algorithms$ units of knowledge are obtained.

Regarding the technical details, a random forest approach was used to implement the feature selection phase: a random forest model was trained on the considered dataset and, according to the importance scores returned by the model, only the four most important columns of the dataset were selected.

The data profile features regarding the whole dataset to which the considered column belongs are the following:

- Number of tuples.
- Number of attributes.
- Percentage of numerical variables.
- Percentage of categorical variables.
- Percentage of duplicate tuples.
- Total size occupied in memory.
- Percentage of columns considered to be correlated.
- Maximum Pearson correlation coefficient between two columns.
- Minimum Pearson correlation coefficient between two columns.

The data profile features regarding the considered single column are the following:

- Uniqueness of the column.
- Density of the column.
- Entropy of the column.
- Column type (categorical or numerical).
- Column mean.
- Column standard deviation.

The imputation methods used to impute the missing values of the incomplete column are the following (see Section 1.5 for the detailed descriptions of the methods):

- Standard Value Imputation.
- Mean Imputation.
- Median Imputation.
- Mode Imputation.
- No Imputation.
- KNN Imputation.
- Random Imputation.
- Linear Regression Imputation.
- Logistic Regression Imputation.

The considerations about the ML algorithms used, the parallelization and the selection of the clean datasets are the same as those described in Section 4.1.1 for the entire dataset case.

An example of a unit of knowledge generated by this process can be found in Figure 4.2 (for visualization purposes the row is broken in several parts):

name	column_name	n_tuples	n_attributes
iris	sepal_length	150	5

p_num_var	p_cat_var	p_duplicates	total_size
1.0	0.0	0.02	6128

column_uniqueness	column_density	column_entropy	p_correlated_features
0.2333	0.0644	3.3424	0.8

max_pearson	min_pearson	column_type	mean
0.9628	-0.4205	numerical	5.8433

std	%missing	ML_ALGORITHM	BEST_METHOD
0.8281	0.4942	dt	impute_knn

Figure 4.2: Example of the generated knowledge - Single column case

This example shows the clean dataset considered (iris), the dataset's single column considered (sepal_length), the data profile features of the whole dataset and of the specific

column, the percentage of missing values injected, the ML algorithm considered (decision tree) and the best imputation method found (KNN imputation).

4.1.3. Data leakage considerations in imputation

As described before, many imputation methods, such as Linear or Logistic Regression Imputation or KNN Imputation, use machine learning techniques to impute missing data. These methods aim to fill in the missing values of a column using models that leverage information contained in other columns of the dataset.

However, during the implementation of the knowledge generation process, a strong suspicion emerged: using also the class of a dataset to impute the missing values of its other columns might lead to a data leakage problem. This suspicion arose because exploiting the information contained in the class variable to fill in the missing values of the features potentially introduce data leakage. Then, during the evaluation phase of the imputation method, a ML algorithm is trained and tested on the imputed dataset, trying to predict the class based on the information present in the features. Therefore, there is a circular relationship where information from the class variable leaks into the features, and then this leaked information is exploited to predict the class. This problem raised concerns about a potential distortion of the performance evaluation of the imputation methods.

To confirm the suspicion, a series of experiments were conducted. First of all, a clean dataset was considered and a ML algorithm was trained and tested on it, keeping track of the performance obtained by the algorithm. Then, missing values were injected into the dataset, and imputed with various imputation methods (as described in Section 4.1.1 and 4.1.2). However, the imputation phase (i.e., the application of every imputation method) was repeated twice, one time also considering the class variable of the dataset as a feature to impute the missing values, and one time not considering it. The ML algorithm was then trained and tested on the resulting imputed datasets, recording the performance obtained.

The results obtained not considering the class variable to impute the missing values are illustrated in Figure 4.3:

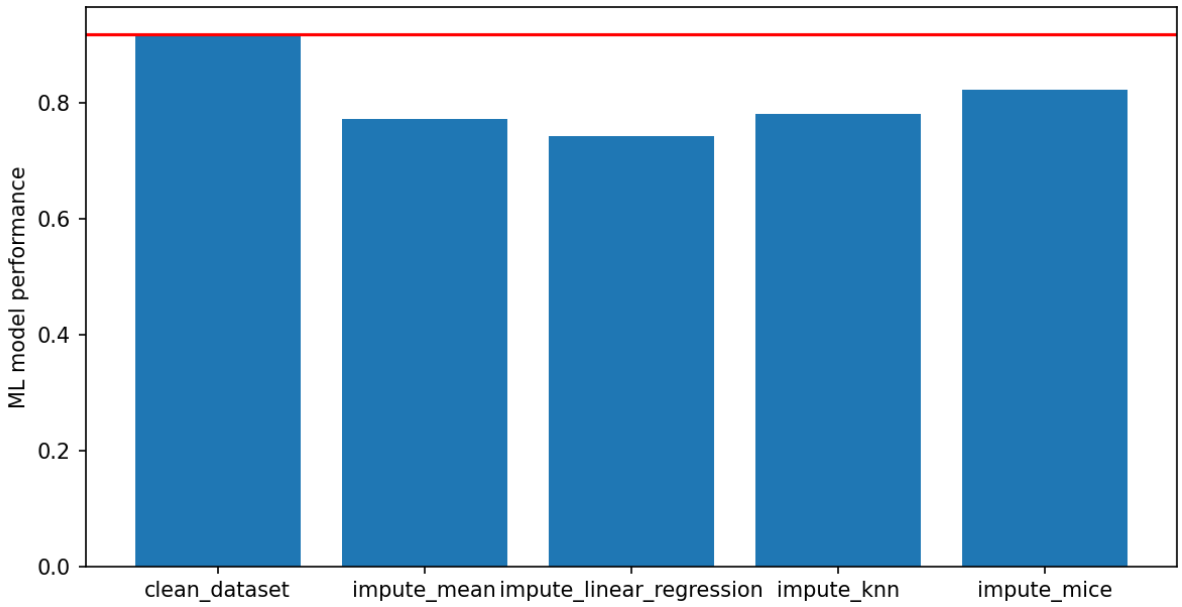

Figure 4.3: Results obtained not considering the class

The x-axis indicates the imputation method used to impute the missing values of the dataset, while the y-axis indicates the performance obtained on that dataset by the ML algorithm. As expected, the performance obtained on the clean dataset is much better than the others.

Instead, the results obtained considering also the class variable to impute the missing values are illustrated in Figure 4.4:

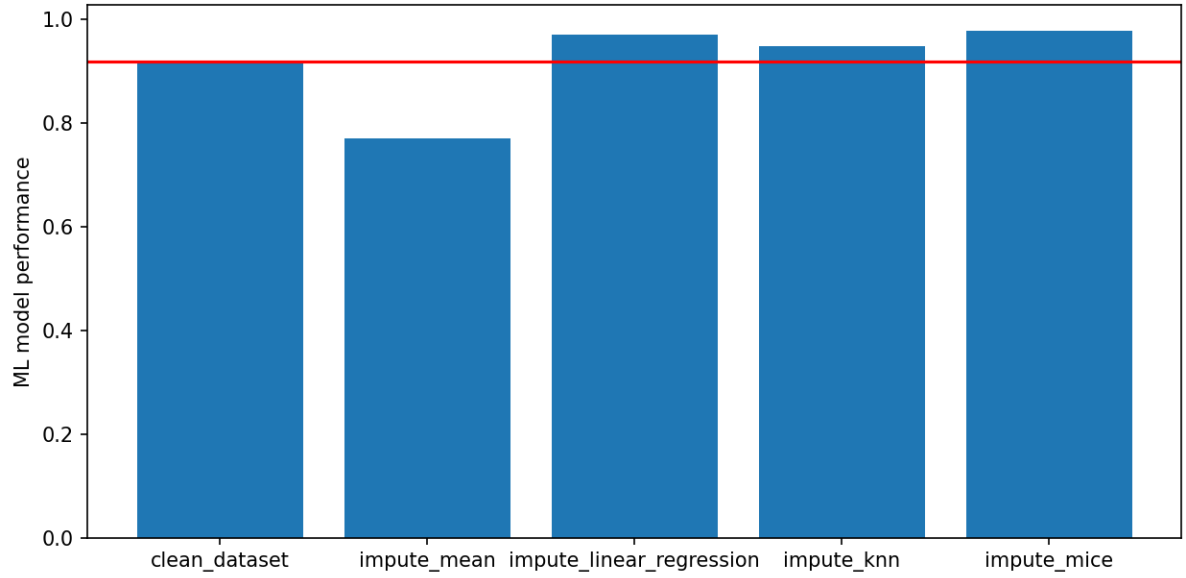

Figure 4.4: Results obtained considering the class

This results show that, in this case, the performances obtained by the ML algorithm on the dataset imputed using machine learning techniques are better than the performance recorded on the clean original dataset. Therefore, it is clear that using also the class variable to impute the missing values of a dataset would introduce data leakage, biasing the performance evaluation of the imputation methods. In particular, the imputation methods that utilize machine learning techniques would have an unfair advantage. Note that the performances obtained on the dataset imputed using Mean Imputation are the same in both cases: this happens because Mean Imputation imputes the missing values of each column separately, using the mean of the column.

These experiments were conducted with various datasets and ML algorithms with similar results. The results shown were obtained with the iris dataset and logistic regression as ML algorithm.

4.2. Building of the classifiers and results

Once the knowledge regarding the best imputation methods was generated, it was stored in the Knowledge Base and at this point was ready to be utilized for the building of the classifiers. The generated knowledge can be gathered in two datasets: one focused on the best methods to impute missing values of entire datasets and one focused on the best methods for single columns. These two datasets can be used for the training and testing of the two classifiers to be built. Recall that the goal is to build two classifiers that predict the best imputation method to use for a given dataset or column and a selected ML algorithm.

At this point, both for the entire dataset case and the single column case, many kinds of classification algorithms have been trained and tested, to try to achieve the most accurate predictions possible.
A procedure similar to k-fold cross-validation has been applied to check the performance of these classifiers. In the two datasets containing the knowledge, there are more rows regarding the same data object because, as described in Sections 4.1.1 and 4.1.2, from a clean dataset more units of knowledge are generated, with a different percentage of missing values or a different ML algorithm. For this reason, the traditional k-fold cross-validation is not a good choice here: if the k folds are picked randomly from the data, more samples generated from the same data object could end up in different folds, so both in training and testing the classifier would see samples derived from that same data object, and this does not reflect the real use case of the classifier, where it must predict the best imputation method for a data object never seen before. Therefore, to check the

performance of the classifiers, it was used a k-fold cross-validation procedure, but in which each fold contains only the samples related to one data object.

For both for the entire dataset case and the single column case, the results obtained were not satisfactory, the accuracy of the predictions was quite low with any kind of classifier tried, even after hyperparameter tuning.
After analyzing the situation, it was observed that in the vast majority of cases the scores of the three best imputation methods to apply in a certain context (data object + ML algorithm) are very close to each other. In other words, the difference in performance among the top three best imputation methods is often minimal. Furthermore, many times the trained classifiers even if they do not predict exactly the best method, predict one of the three top performing methods, thus their prediction is still a very good choice for imputation.

For these reasons, it was decided to extend the generated knowledge in order for it to contain all three best imputation methods. The structure of the knowledge is identical to the one shown in Sections 4.1.1 and 4.1.2, but instead of indicating only the best method, it indicates the three best methods. This operation was performed for both the entire dataset case and for the single column case. An example of the entire dataset case can be found in Figure 4.5 (for visualization purposes the row is broken in several parts):

name	n_tuples	n_attributes	p_num_var
iris	150	5	1.0

p_cat_var	p_duplicates	total_size	p_avg_distinct
0.0	0.02	6128	0.168

p_max_distinct	p_min_distinct	avg_density	max_density
0.2867	0.02	0.0351	0.0644

min_density	avg_entropy	max_entropy	min_entropy
0.0	2.7058	3.4892	1.0986

p_correlated_features	max_pearson	min_pearson	%missing
0.8	0.9628	-0.4205	0.4002

ML_ALGORITHM	BEST_METHOD1	BEST_METHOD2	BEST_METHOD3
dt	impute_standard	impute_median	impute_std

Figure 4.5: Example of the extended knowledge

Further experiments with the classifiers were conducted after enriching the knowledge in this manner. In particular, the classifiers were trained using the same previously described procedure, but to assess the classifiers performance a new accuracy metric was used: this metric considers a prediction correct if the imputation method predicted belongs to the top three methods.

With these new experiments, as expected, there was a relevant boost in performance, for both the entire dataset case and the single column case. Specifically, for the entire dataset case an accuracy of 0.5944 was reached, while for the single column case it reached 0.5484.

At this point, a different experimental approach was tried: the goal was to aggregate the knowledge by merging the rows referred to the same data object and ML algorithm but with different percentage of missing values. This approach was taken due to concerns that presenting nearly identical rows to the classifier might confuse it. Therefore, each group of rows that were identical (of course not considering the best method field) but had a different percentage of missing values was merged in a single row. As a consequence, the "%missing" field was removed from the knowledge.

To perform this aggregation, in the case of the original knowledge (containing only the best method, not the top three), mode was utilized. The rows referred to the same data object and ML algorithm were grouped together and aggregated into a single row having as best imputation method the mode of the best methods of the initial rows.

On the other hand, for the knowledge containing the top three methods the aggregation process was more complex. Given a group of rows referred to the same data object and ML algorithm, each row with its ranking of the top three methods, the objective was to aggregate those rows in a single row with a unique ranking. Therefore, it was necessary to solve a rank aggregation problem.

The approach used in this thesis to solve this rank aggregation problem is loosely inspired to Borda's method. For every candidate imputation method, in each of the rankings to aggregate, the candidate method earns points based on its ranking position. Specifically, the method accumulates 3 points for the first position, 2 points for the second position and 1 point for the third. The points obtained in each ranking are then summed. This process is repeated for each candidate method. At the end a total unique ranking is generated, ordering the methods according to the total number of points they have acquired (more points indicates a better method).

The aggregation process was applied to both the knowledge regarding the entire dataset case and the one regarding the single column case.

The aggregation led to an improvement of the classifiers performance. After the testing of several different classifiers, coupled with the corresponding hyperparameter tuning, the highest performance achieved were an accuracy of 77.78 for the entire dataset case, and 67.96 for the single column case. These performance were obtained, in both cases, using the knowledge containing the top three methods and in which rows referring to the same data object and ML algorithm were aggregated.

Regarding the technical details, for the entire dataset case the classifier reporting the highest performance is a k-Nearest Neighbors model, with a number of neighbors k = 4. While for the single column case the best classifier is a Logistic Regression model, with Lasso regularization with hyperparameter C = 0.01.
The datasets containing the knowledge on which the classifiers were trained and tested were preprocessed in this manner: the categorical variables, i.e., ML algorithm and column type, were encoded using one-hot-encoding, while the numerical variables were normalized using MinMaxScaler, which scaled the data in range (0,1). Also, the variables containing dataset and column names were obviously eliminated before the training.

5 | Tool Implementation

To demonstrate a practical application of the concepts discussed in the previous chapters, an already existing tool was extended and enriched with new functionalities.

This chapter provides a brief general description of the tool and a more in-depth exploration of the new functionalities developed during the course of this work.

The tool is implemented with Flask, a framework for web application development utilizing Python programming language.

The objective of the tool is to guide users through the data preparation process. Firstly, the users can load a dataset on which they want to perform data preparation, as illustrated in Figure 5.1.

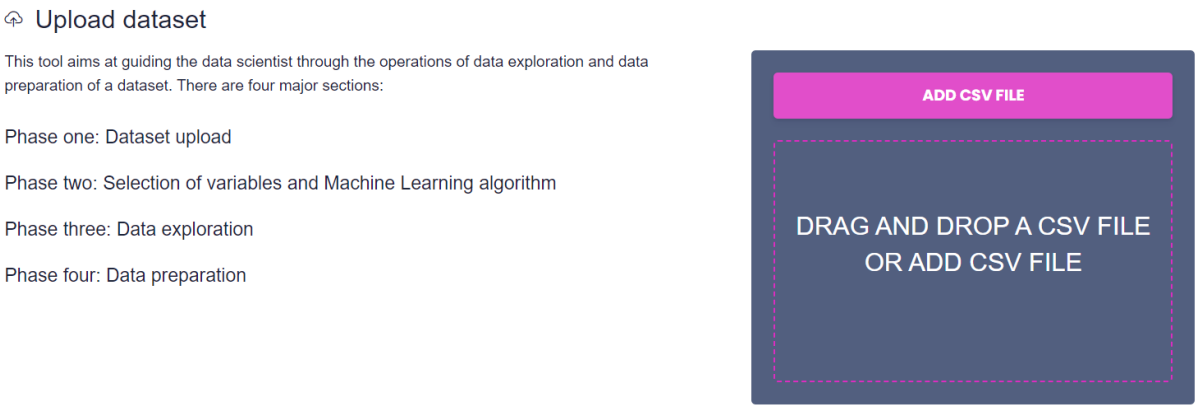

Figure 5.1: Dataset Uploading Phase

Then, it is asked to the users which machine learning application is their objective analysis for the loaded dataset. At the moment, the tool is focused only on classification, and the possible methods that can be chosen are Decision Tree, K-Nearest Neighbors, and Naive Bayes. This phase is shown in Figure 5.2.

Figure 5.2: Machine Learning Algorithm Selection

At this point, the loaded dataset undergoes a data profiling phase, during which it is explored and its data profile is extracted. Through a data profiling panel, the tool shows the users the characteristics of the dataset, to make them more aware of their data and the data quality problems it may have.

After the data profiling panel comes the actual data preparation phase. The users can build their data preparation pipeline by choosing the actions to perform from a list of possible data preparation techniques and methods. The users can then confirm the selected pipeline and the chosen actions will be actually performed on the dataset.

Figure 5.3 shows the entire data preparation panel. In the upper left part of the panel, it is possible to see the data preparation pipeline built by the users. In the lower left part is the list of all available data preparation actions that users can select. The right part, instead, displays the loaded dataset, so that users can always monitor it while building the data preparation pipeline. The specific elements of this figure will be described in detail in the following paragraphs.

Figure 5.3: Data Preparation Panel

The contributions to this tool added during this thesis are focused on the data preparation phase just described. Specifically, the objective of this work was to integrate into the tool the knowledge base (described in Chapter 3) and the classifier (described in Chapter 4). This integration aims to improve the data preparation pipeline process and provide the users with recommendations for the best data preparation methods to apply based on their specific dataset and ML application selected.

To integrate the Knowledge Base into the tool, the first step was to connect the tool's Flask environment with the Neo4j database in which the KB is stored. This was possible through the use of the Neo4j Python driver, the library provided by Neo4j to interact with a database instance through a Python application. After establishing a connection with the database, it became possible to query the database directly by writing queries within the Python code.

The list of available data preparation actions is divided into several segments: each segment is dedicated to the improvement of a particular data quality dimension, and then there is a final segment containing actions explicitly aimed to improve the execution of the selected ML application. Currently, the data quality dimensions considered in the tool are Completeness, Accuracy and Uniqueness. The list of the four segments currently present in the tool (1. Completeness, 2. Accuracy, 3. Uniqueness, 4. ML Oriented Actions) is visible in Figure 5.3.

Within each segment, several possible data preparation actions are available for users to select. Each action is characterized by a data preparation technique and by a data preparation method that implements that technique. The actions are presented to the users within draggable elements, for instance, as shown in Figure 5.4.

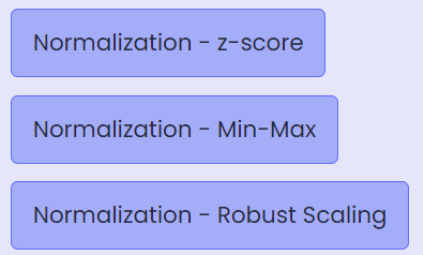

Figure 5.4: Data Preparation Actions Example

The values preceding the dash indicate what data preparation technique will be applied, while those following the dash specify what particular method will be used to execute the technique.

To select a certain action, users can drag and drop it from the list of all the available actions into the container representing the data preparation pipeline to be executed. The order of the actions within the pipeline is important: their execution will happen following the precise sequence in which they are arranged.

All the data preparation actions available in the various segments are loaded from the knowledge base. Regarding the segments dedicated to improving a data quality dimension, the actions are obtained through the following querying process: for the considered dimension, all the techniques that affect that dimension positively (i.e., the influence type is "Improvement") are extracted from the knowledge base, then all the methods implementing each technique are also retrieved.

The code of the queries executed to interrogate the knowledge base can be found in Appendix A.

Therefore, each segment dedicated to a data quality dimension will contain all the actions retrieved from the knowledge base, that improve that dimension. The users can then choose what actions to select and include in their pipeline.

Regarding instead the segment dedicated at improving the execution of the ML application, the procedure is slightly different. As for the previous segments, all the actions available are queried from the knowledge base. However, the actions within this segment are customized for the specific ML application selected by the users; in this way, the

recommendations are personalized for their particular scenario. The querying process is the following: for the selected ML application, all the techniques that benefit that specific application are retrieved from the knowledge base, along with the methods implementing the techniques. Note that users may have different proposed actions in this segment if they select different ML applications.

Retrieving the data preparation actions from the knowledge base provides several advantages.

Firstly, it offers flexibility and scalability: the knowledge base operates as a dynamic repository of actions, in which it is easily possible to add new data preparation techniques or methods, or to update their relationships with data quality dimensions and ML applications. Therefore, there is not a fixed set of actions hard-coded into the system, but the techniques and methods proposed to users are drawn from a repository that can be constantly expanded and updated.

Secondly, taking actions from the knowledge base allows the provision of personalized recommendations for specific scenarios: for instance, as described before, in the segment dedicated to improving the execution of the ML application, actions are proposed based on the user's chosen application.

Another contribution made in this thesis, again aimed to exploit the potential of the knowledge base, is a functionality that retrieves and presents to the user information about selected data preparation actions. This functionality empowers the users to choose a certain data preparation action (composed as previously mentioned by a technique and a method) and to request detailed information about that action. When a user's request is received, the tool executes a series of queries to the knowledge base, retrieving all the information about the chosen data preparation technique and method. The results of these queries are then assembled together and presented in a user-friendly textual response.

More precisely, the information extracted from the knowledge base includes: which quality dimensions are affected by the chosen technique, specifying the type of its influence; which ML applications (if any) benefit from the technique; the data profile features on which the technique and method depend on, along with the corresponding description of the dependencies.

The primary objective of this functionality is to make users more aware of the data preparation actions they can choose and to clarify when and why it is appropriate to use them.

To request information about a data preparation action, users are required to drag and

drop the desired action into the pipeline container, then they can use the "Get information" button (shown in Figure 5.5) to send their request.

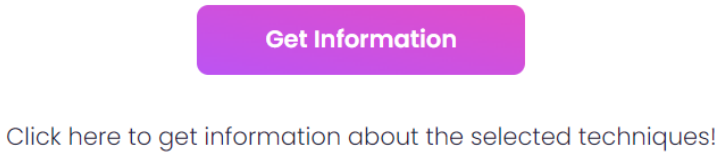

Figure 5.5: Get Information Button

Following the request, a textual response is presented to the user in a paragraph below the button. For instance, in the case of requesting information about the action "Imputation - Imputation using functional dependencies", the obtained response is displayed in Figure 5.6:

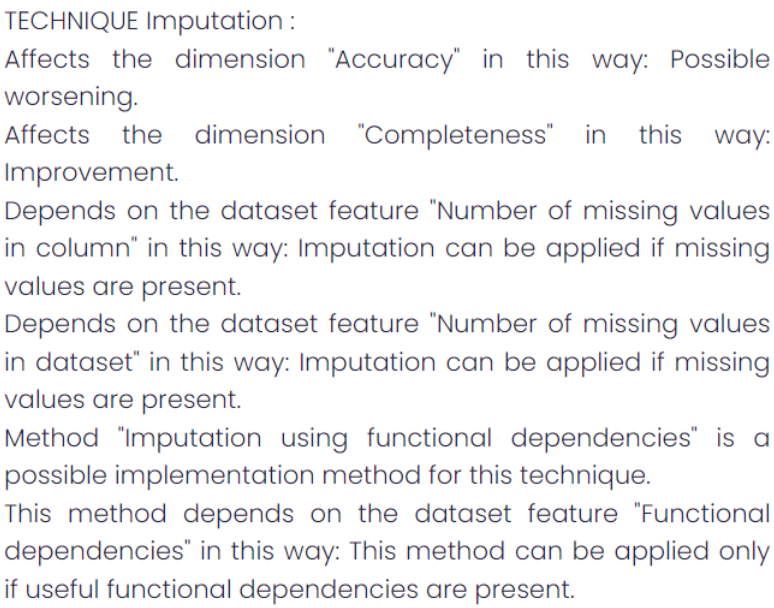

Figure 5.6: Get Information Response

The users can exploit the received information to make more appropriate and aware choices about which actions to insert in their pipeline.
After reading the information, users can request details about other actions if they wish.

Besides the knowledge base, the classifier described in Chapter 4 was also integrated into the tool.
In particular, of the two classifiers implemented, the one integrated into the tool was the

classifier focusing on the best imputation methods for imputing missing values of entire datasets.

To realize this integration, the classifier was trained on all the available knowledge, saved (already trained) as a file, and then imported into the tool. In the following paragraphs, there is a detailed description of how the classifier is applied inside the tool.

After a new dataset is loaded by the users, during the data profiling phase the data profile features of the dataset are extracted and saved. Specifically, the data profile features extracted are those on which the classifier was trained, listed in Section 4.1.1. At this point, the ML Application chosen by the users is appended to the extracted features, because it was another attribute taken in input by the classifier to make predictions. The obtained data, consisting in the data profile features of the dataset + the ML Application chosen, is stored temporarily into a file.

The obtained data is subjected to the same preprocessing steps applied on the dataset used to train the classifier. Following the preprocessing, the data is then provided as input to the trained classifier. The classifier utilizes this information to predict the best imputation method to be applied to the dataset, considering both the dataset's characteristics and the chosen ML application.

Once the best imputation method is predicted, it is displayed to the users during the data preparation phase. In particular, it is communicated textually using the sentence shown in Figure 5.7a, and the data preparation action corresponding to that imputation method is highlighted in a different color, as illustrated in Figure 5.7b.

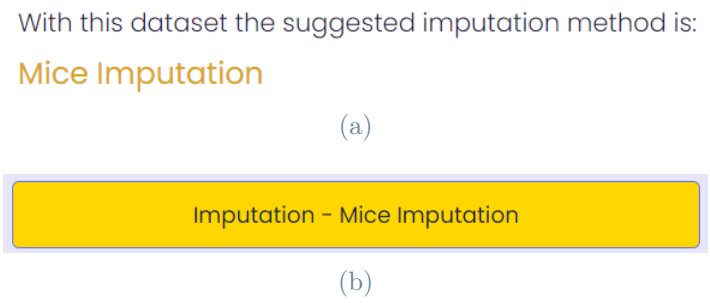

Figure 5.7: Best Imputation Method

Furthermore, the tool automatically includes the data preparation action corresponding to the predicted imputation method in the pipeline container for the users' convenience. However, users have the freedom of removing or changing that action from the pipeline, if they believe another imputation method is more suited for their needs.

Therefore, the integration of the classifier provides users with a recommendation tailored for their specific data and chosen application while preserving their complete freedom in selecting the actions they prefer.

6 | Conclusions

This thesis presented a methodology aimed to guide users through the data preparation process, suggesting actions appropriate for their specific data and analysis objectives. The foundation of this methodology is a Knowledge Base, in which all the concepts required for supporting data preparation are stored and retrieved when needed. During the thesis, the Knowledge Base was conceptually designed, implemented using a graph database, and populated.

Through a series of experiments, valuable knowledge regarding the best imputation methods to use in various contexts was generated and stored in the Knowledge Base. These experiments explored the performance of several imputation methods in combination with different datasets and Machine Learning applications. From the experiments, it emerged that both the dataset characteristics and the employed Machine Learning application significantly impact which imputation method is more appropriate to use.

The presented methodology makes use of classifiers to provide tailored recommendations for users' specific needs. For a certain data preparation technique, a classifier is given as input the data profile of a data object and a selected Machine Learning application, and predicts the best method to implement the technique with that specific context.

In particular, the generated knowledge was leveraged for implementing classifiers predicting the best imputation methods to fill in missing values of a data object. Two separate classifiers were developed, one for entire datasets, predicting the best method to impute all the missing values in a dataset, and one for single columns, predicting the best method to impute the missing values of a specific column.

Embracing a Human-In-The-Loop approach, in this methodology, users are actively involved during the entire course of data preparation and can freely accept or reject any suggestion they receive. Users' choices are then stored in the Knowledge Base and are utilized as valuable feedback to adjust and refine future suggested actions.

In conclusion, the developed Knowledge Base and classifiers are useful resources within the proposed methodology. They support users through the data preparation process,

pointing them toward optimal choices and providing tailored suggestions, but always ensuring complete freedom.

6.1. Future Work

One of the possible improvement that can be introduced in the methodology in future work is the refinement of the learning mechanism that leverages past users' choices to adjust and improve the suggestions. For instance, it would be interesting to suggest to a user some actions that other users have in the past chosen in similar contexts.

Furthermore, while at the moment the only developed classifiers are focused on the imputation technique, the conceptual framework is extensible. Future work may focus on the creation of additional classifiers, predicting optimal methods for implementing other data preparation techniques. This expansion would enlarge the scope of the methodology, offering a more comprehensive and precise set of recommendations regarding all the preparation techniques.

Bibliography

[1] Z. Abedjan, L. Golab, F. Naumann, and T. Papenbrock. *Data Profiling*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2018. ISBN 978-3-031-00737-8. doi: 10.2200/S00878ED1V01Y201810DTM052. URL https://doi.org/10.2200/S00878ED1V01Y201810DTM052.

[2] E. Acuna and C. Rodriguez. The treatment of missing values and its effect on classifier accuracy. In *Classification, Clustering, and Data Mining Applications: Proceedings of the Meeting of the International Federation of Classification Societies (IFCS), Illinois Institute of Technology, Chicago, 15–18 July 2004*, pages 639–647. Springer, 2004.

[3] J. Barateiro and H. Galhardas. A survey of data quality tools. *Datenbank-Spektrum*, 14:15–21, 01 2005.

[4] C. Batini and M. Scannapieco. *Data and Information Quality - Dimensions, Principles and Techniques*. Data-Centric Systems and Applications. Springer, 2016. ISBN 978-3-319-24104-3. doi: 10.1007/978-3-319-24106-7. URL https://doi.org/10.1007/978-3-319-24106-7.

[5] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino. Methodologies for data quality assessment and improvement. *ACM Comput. Surv.*, 41(3):16:1–16:52, 2009. doi: 10.1145/1541880.1541883. URL https://doi.org/10.1145/1541880.1541883.

[6] L. Berti-Équille. Learn2clean: Optimizing the sequence of tasks for web data preparation. In L. Liu, R. W. White, A. Mantrach, F. Silvestri, J. J. McAuley, R. Baeza-Yates, and L. Zia, editors, *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 2580–2586. ACM, 2019. doi: 10.1145/3308558.3313602. URL https://doi.org/10.1145/3308558.3313602.

[7] M. Bovee, R. P. Srivastava, and B. Mak. A conceptual framework and belief function approach to assessing overall information quality. In E. M. Pierce and R. Katz-Haas, editors, *Sixth Conference on Information Quality (IQ 2001)*, pages 311–328. MIT, 2001.

[8] J. Brownlee. *Data preparation for machine learning: data cleaning, feature selection, and data transforms in Python*. Machine Learning Mastery, 2020.

[9] J. Brownlee. Why one-hot encode data in machine learning?, 2020. URL `https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/`.

[10] M. Caffagnini. Sliding autonomy in data preparation: Balancing human expertise and automated processes. Master Thesis, 2023.

[11] B. Chaudhari, A. Agarwal, and T. Bhowmik. Simultaneous improvement of ML model fairness and performance by identifying bias in data. *CoRR*, abs/2210.13182, 2022. doi: 10.48550/arXiv.2210.13182. URL `https://doi.org/10.48550/arXiv.2210.13182`.

[12] M. R. Chmielewski and J. W. Grzymala-Busse. Global discretization of continuous attributes as preprocessing for machine learning. *International Journal of Approximate Reasoning*, 15(4):319–331, 1996. ISSN 0888-613X. doi: https://doi.org/10.1016/S0888-613X(96)00074-6. URL `https://www.sciencedirect.com/science/article/pii/S0888613X96000746`. Rough Sets.

[13] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. KATARA: A data cleaning system powered by knowledge bases and crowdsourcing. In T. K. Sellis, S. B. Davidson, and Z. G. Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1247–1261. ACM, 2015. doi: 10.1145/2723372.2749431. URL `https://doi.org/10.1145/2723372.2749431`.

[14] A. C. Cohen. *Truncated and censored samples: theory and applications*. CRC press, 1991.

[15] I. F. Ilyas and X. Chu. *Data Cleaning*. Association for Computing Machinery, New York, NY, USA, 2019. ISBN 9781450371520.

[16] J.-M. Jo. Effectiveness of normalization pre-processing of big data to the machine learning performance. *The Journal of the Korea institute of electronic communication sciences*, 14(3):547–552, 2019.

[17] J. Kaiser. Dealing with missing values in data. *Journal of Systems Integration*, 5: 42–51, 01 2014. doi: 10.20470/jsi.v5i1.178.

[18] S. Krishnan and E. Wu. Alphaclean: Automatic generation of data cleaning pipelines. *CoRR*, abs/1904.11827, 2019. URL `http://arxiv.org/abs/1904.11827`.

[19] S. Krishnan, M. J. Franklin, K. Goldberg, and E. Wu. Boostclean: Automated error detection and repair for machine learning. *CoRR*, abs/1711.01299, 2017. URL http://arxiv.org/abs/1711.01299.

[20] G. Lahera. Unbalanced datasets and what to do about them, 2019. URL https://medium.com/strands-tech-corner/unbalanced-datasets-what-to-do-144e0552d9cd.

[21] M. Mahdavi, F. Neutatz, L. Visengeriyeva, and Z. Abedjan. Towards automated data cleaning workflows. In R. Jäschke and M. Weidlich, editors, *Proceedings of the Conference on "Lernen, Wissen, Daten, Analysen", Berlin, Germany, September 30 - October 2, 2019*, volume 2454 of *CEUR Workshop Proceedings*, pages 10–19. CEUR-WS.org, 2019. URL https://ceur-ws.org/Vol-2454/paper_8.pdf.

[22] F. Naumann and M. Herschel. *An Introduction to Duplicate Detection*. Morgan and Claypool Publishers, 2010. ISBN 1608452204.

[23] Neo4j. Neo4j official documentation, 2023. URL https://neo4j.com/.

[24] J. Pokorný. Graph databases: Their power and limitations. In K. Saeed and W. Homenda, editors, *Computer Information Systems and Industrial Management*, pages 58–69, Cham, 2015. Springer International Publishing.

[25] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000. URL http://sites.computer.org/debull/A00DEC-CD.pdf.

[26] T. C. Redman. *Data quality for the information age*. Artech House, 1996. ISBN 978-0-89006-883-0.

[27] T. C. Redman and A. B. Godfrey. *Data Quality for the Information Age*. Artech House, Inc., USA, 1st edition, 1997. ISBN 0890068836.

[28] N. N. K. S. Pandas — bfill and ffill, 2020. URL https://navinniish001.medium.com/pandas-bfill-and-ffill-b79e46ab87ae.

[29] M. Saar-Tsechansky and F. Provost. Handling missing values when applying classification models. *Journal of Machine Learning Research*, 8(57):1623–1657, 2007. URL http://jmlr.org/papers/v8/saar-tsechansky07a.html.

[30] C. Sancricca, C. Cappiello, et al. Supporting the design of data preparation pipelines. In *CEUR WORKSHOP PROCEEDINGS*, volume 3194, pages 149–158. CEUR-WS, 2022.

[31] S. Shrivastava, D. Patel, A. Bhamidipaty, W. M. Gifford, S. A. Siegel, V. S. Ganapavarapu, and J. R. Kalagnanam. DQA: scalable, automated and interactive data quality advisor. In C. K. Baru, J. Huan, L. Khan, X. Hu, R. Ak, Y. Tian, R. S. Barga, C. Zaniolo, K. Lee, and Y. F. Ye, editors, *2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9-12, 2019*, pages 2913–2922. IEEE, 2019. doi: 10.1109/BigData47090.2019.9006187. URL https://doi.org/10.1109/BigData47090.2019.9006187.

[32] R. Somasundaram and R. Nedunchezhian. Evaluation of three simple imputation methods for enhancing preprocessing of data with missing values. *International Journal of Computer Applications*, 21(10):14–19, 2011.

[33] N. Tang, J. Fan, F. Li, J. Tu, X. Du, G. Li, S. Madden, and M. Ouzzani. Relational pretrained transformers towards democratizing data preparation [vision]. *CoRR*, abs/2012.02469, 2020. URL https://arxiv.org/abs/2012.02469.

[34] R. M. Trevor Hastie. softimpute: Matrix completion via iterative soft-thresholded svd, 2021. URL https://cran.r-project.org/web/packages/softImpute/index.html.

[35] A. Vukotic, N. Watt, T. Abedrabbo, D. Fox, and J. Partner. *Neo4j in action*, volume 22. Manning Shelter Island, 2015.

[36] R. Y. Wang. A product perspective on total data quality management. *Commun. ACM*, 41(2):58–65, feb 1998. ISSN 0001-0782. doi: 10.1145/269012.269022. URL https://doi.org/10.1145/269012.269022.

[37] R. Y. Wang and D. M. Strong. Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems*, 12(4):5–33, 1996. doi: 10.1080/07421222.1996.11518099. URL https://doi.org/10.1080/07421222.1996.11518099.

[38] J. Webber. A programmatic introduction to neo4j. In *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*, SPLASH '12, page 217–218, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450315630. doi: 10.1145/2384716.2384777. URL https://doi.org/10.1145/2384716.2384777.

[39] C. Yan and Y. He. Auto-suggest: Learning-to-recommend data preparation steps using data science notebooks. In D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA],*

June 14-19, 2020, pages 1539–1554. ACM, 2020. doi: 10.1145/3318464.3389738. URL https://doi.org/10.1145/3318464.3389738.

[40] Z. Zhang. Multiple imputation with multivariate imputation by chained equation (mice) package. *Annals of translational medicine*, 4:30, 02 2016. doi: 10.3978/j.issn.2305-5839.2015.12.63.

A | Appendix A

```
# retrieving techniques improving a certain dimension

"MATCH (n:DATA_PREPARATION_TECHNIQUE)-[a:AFFECTS]->(d:DQ_DIMENSION)
WHERE a.influence_type = $influence_type and d.name = $dimension_name
RETURN n.name AS name",
influence_type="Improvement",
dimension_name=dimension,
database_="neo4j"
```

Listing A.1: Extraction of data preparation actions improving a dimension - Part 1

```
# retrieving methods

"MATCH (n:DATA_PREPARATION_TECHNIQUE)-[:IMPLEMENTED_WITH]->
(m:DATA_PREPARATION_METHOD)
WHERE n.name = $technique_name
RETURN m.name AS name",
technique_name=tech["name"],
database_="neo4j"
```

Listing A.2: Extraction of data preparation actions improving a dimension - Part 2

```
# retrieving techniques that benefit a certain ML application

"MATCH (n:DATA_PREPARATION_TECHNIQUE)-[:BENEFITS_FROM]
-(ml:ML_APPLICATION)
WHERE ml.application_method = $ml_algorithm
RETURN DISTINCT n.name AS name",
ml_algorithm=ml_algorithm,
database_="neo4j"
```

Listing A.3: Extraction of data preparation actions that benefit a ML application - Part 1

```
# retrieving methods

"MATCH (n:DATA_PREPARATION_TECHNIQUE)-[:IMPLEMENTED_WITH]->
(m:DATA_PREPARATION_METHOD)
WHERE n.name = $technique_name
RETURN m.name AS name",
technique_name=tech["name"],
database_="neo4j"
```

Listing A.4: Extraction of data preparation actions that benefit a ML application - Part 2

```
# dimensions affected by the requested technique

"MATCH (t:DATA_PREPARATION_TECHNIQUE{name: $technique})-[a:AFFECTS]->
(d:DQ_DIMENSION)
RETURN d.name AS dimension, a.influence_type AS influence_type",
technique=technique,
database_="neo4j"
```

Listing A.5: Executed queries for the "Get Information" functionality - Part 1

```
# ML applications benefiting from the technique

"MATCH (t:DATA_PREPARATION_TECHNIQUE{name: $technique})
<-[:BENEFITS_FROM]-(ml:ML_APPLICATION)
RETURN ml.application_method AS ml_algorithm",
technique=technique,
database_="neo4j"
```

Listing A.6: Executed queries for the "Get Information" functionality - Part 2

```
# dependencies between technique and features

"MATCH (t:DATA_PREPARATION_TECHNIQUE{name: $technique})
-[d:DEPENDS_ON]->(p:DATA_PROFILE_FEATURE)
RETURN p.name AS feature, d.description AS description",
technique=technique,
database_="neo4j"
```

Listing A.7: Executed queries for the "Get Information" functionality - Part 3

```
# dependencies between method and features

"MATCH (t:DATA_PREPARATION_METHOD{name: $method})-[d:DEPENDS_ON]
->(p:DATA_PROFILE_FEATURE)
RETURN p.name AS feature, d.description AS description",
method=method,
database_="neo4j"
```

Listing A.8: Executed queries for the "Get Information" functionality - Part 4

List of Figures

List of Tables