FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Visualization of Data Flags in Table Lens Views to Improve the Readability of Metadata and the Tracking of Data Cleaning

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software and Information Engineering

eingereicht von

Muhammad Mujahed Hainoun
Matrikelnummer 01325827

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao. Univ. Prof. Univ.-Doz. Dipl.-Ing. Dr. Techn. Eduard Gröller
Mitwirkung: Dipl.-Ing. Dr. Techn. Harald Piringer
Dipl.-Ing. Clemens Arbesser

Wien, 15. August 2019

Muhammad Mujahed Hainoun	Eduard Gröller

FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Visualization of Data Flags in Table Lens Views to Improve the Readability of Metadata and the Tracking of Data Cleaning

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software and Information Engineering

by

Muhammad Mujahed Hainoun
Registration Number 01325827

to the Faculty of Informatics

at the TU Wien

Advisor: Ao. Univ. Prof. Univ.-Doz. Dipl.-Ing. Dr. Techn. Eduard Gröller
Assistance: Dipl.-Ing. Dr. Techn. Harald Piringer
Dipl.-Ing. Clemens Arbesser

Vienna, 15th August, 2019

Muhammad Mujahed Hainoun   Eduard Gröller

Erklärung zur Verfassung der Arbeit

Muhammad Mujahed Hainoun
Karmarschgasse 18a/2/13, 1100 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 15. August 2019

Muhammad Mujahed Hainoun

Abstract

Recent evaluation indicates that wrong decisions resulting from systems operating based on bad data costed worldwide about $30 billion in the year 2006. This work addresses the importance of *Data Quality* (DQ) as a critical requirement in any information system. In this regard, DQ criteria and problems such as missing entries, duplicates, and faulty values are identified. Different approaches and techniques used for data cleaning to fix DQ issues are reviewed. In this work a new technique is integrated into `VISPLORE`, a framework for data analysis and visualization, that allows the framework to visualize multiple types of per-value meta-information. We will show how our work enhances the readability of the *table lens* view, one of the many viewing modes provided in `VISPLORE`, and helps the user understand the status of data entries to decide on what entries need to be cleaned and how. This work also expands on the interactive data cleaning tools provided by `VISPLORE`, by allowing the user to manually delete implausible values or replace them with more plausible ones, while keeping track of this cleaning process. With the integrated new features to the table lens view, `VISPLORE` is now able to present more detailed data with enhanced visualization features and interactive data cleaning.

Contents

CHAPTER 1

Introduction

The advancement of digital technologies, mainly driven by smart devices and solutions, internet and smartphones, and information & communication technologies (ICTs), is penetrating all fields of our daily life. Digitization is simply defined as the process of converting information into a digital format with the purpose of enabling the automation of different tasks for making them easy to control and less time consuming ([Satyendra, 2016]). Digitization has been extremely expanded over the last years and is currently on the way to conquer all areas of modern society encompassing information, trades, banking and insurance, online services for hotel, travel and shopping, health, education, management of energy demand, and the emergence of smart grids and smart cities. This continuously evolving trend will shape our future and change our familiar life environment. It is expected to improve the quality of life and making it more resource efficient and sustainable.

One important and vital field of life that has been experiencing remarkable digitization is the energy system spanning from the energy production to the transmission and distribution up to the delivery of modern energy services covering heating, cooling, lighting, cooking, processing, and mobility. Digital technologies will help us make our energy systems more connected, intelligent, efficient, and sustainable ([IEA, 2017]). This trend is reflected in the ongoing transition towards clean energy systems characterized by the increased contribution of intermittent renewable energy generations (e.g., photovoltaics and wind) and significant improvement of energy efficiency in all consumption sectors like building, industry, and transportation. The increased contribution of renewable energies with their weather dependent intermittent behavior, requires increased management effort to ensure the balance between energy demand and supply to achieve the required supply security. On the supply side we need to employ various flexibility options like grid flexibility and storage facilities for power and heat. On the demand side we need to employ several measures supported by socio-economic incentives of consumers like demand side management, load shifting, and load management. The successful implementation of those measures relies upon digital applications such as smart appliances and devices, smart electric grid, and shared mobility (e.g., car sharing). Operating, controlling, and monitoring of these applications requires the employment of adequate ICT technologies to manage the whole process that results in generating a big amount of data in short periods of time. Big data requires an integrated process of data management to deal with the measurement, acquisition, collection, and transfer of data and the subsequent data processing to evaluate and visualize the data.

One important purpose of such an undertaking is to generate useful information and extract key indicators to monitor the performance of the whole energy system and support the decision-making process in ensuring affordable, reliable, and sustainable energy services for the end consumers ([IEA, 2017]). An Essential part of this process is the visualization of the collected data and the subsequent extraction of related performance indicators. As example for processing of selected energy consumption data, one might consider the hourly electricity and gas consumption in different districts of the city Vienna. Visualization of the consumption delivers valuable information about the time and magnitude of the peak demand (e.g., daily, weekly, and annually) of both energy forms. Based on that the load factors can be extracted as key indicators to optimize the electricity and gas supply of the considered districts. One can also bring the consumption in relation with the external temperature and thus understand the impact of weather conditions on the consumption of both energy forms and make use of that for future demand projection. Processing data to extract information however is beyond the scope of this work, and instead data visualization is the focus here.

It is the visualization of big data that makes them meaningful and reveals their scientific value in order to conduct further consultations and come to tangible conclusions. Visualization helps data analysts to locate where data are faulty or missing, beside identifying anomaly and inconsistency of the measured data for the purpose of detecting devices malfunction and identify responsible entities to fix bugs and intervene to rectify the malfunction in due time ([Kern and Zinck, 2016]). Finding solutions to deal with the problem of the observed anomaly, errors, and missing of data is becoming increasingly important in view of the emergence of big data management, like the above-mentioned field of energy data that are characterized by periodic patterns.

With the variety of sources and types of data issues, it is advised to handle the whole process of providing reliable, robust, and accurate data within a comprehensive procedure of ensuring Data Quality (DQ) as a basis for enabling a data-based decision-making process. Ensuring DQ is becoming a new vital requirement for the design of ICT systems ([Ganapathi and Chen, 2016, Chen and Jiang, 2014]). DQ assurance is being assessed by multiple dimensions called *DQ metrics*, many of them are based on the application domain and the user's needs. The most commonly DQ metrics refer to the following aspects:

- Completeness.
- Validity.

- Accuracy.
- Non-duplication.
- Consistency.

In all information systems, dirty data presents one of the biggest challenges that impact the DQ metrics. Errors associated with dirty data originate from various sources of data entry, measurement, distillation, and integration ([Hellerstein, 2008]). Dirty data can be exemplarily classified by the following groups ([Swapna et al., 2016, Dasu and Johnson, 2003]):

- Syntax errors such as typos and wrong formats.
- Semantical errors such as data layout and types (string vs. integral), format and scale ($ vs. €), gaps in time series.
- Missing entries (i.e., nulls).
- Dummy values.
- Inconsistent values that contradict with other values.
- Cryptic or hidden data.
- Data integration issues.
- Data violating the business rules.

Systems of effective data analysis require high data quality that reflects the real world correctly and are free of the above-mentioned *glitches*. Such quality can be achieved by preprocessing the raw data before analysis, i.e., while collecting and/or after writing it into the system's dataset ([Swapna et al., 2016]). It is about detecting errors in data and eliminating them in the upmost early stage. This preprocessing stage is referred to as *Data Cleaning* and is an important step in any data analysis system. *Data cleaning* is about defining strict rules of what clean data is, and then applying automated data-cleaning tools to ensure that the data is under constant validation and auditing to maintain a high level of data integrity and correctness. This becomes a challenging process if dealing with big data, where the scale and complexity of the data makes the cleaning process more difficult. In many decision systems, data cleaning consumes about 80% of the time required during the analysis process. Still, the benefits of data cleaning are huge if the correctness of the analysis depends on the quality of the data. In one example related to a system of data classification, the result's accuracy is increased by 100% by cleaning the data first ([Ganapathi and Chen, 2016]). Moreover, a study found that the cost of bad data reached in the year 2006 about $30 billion. These include the cost of lost opportunities and wrong decisions based on bad data ([Lee, 2007]).

Data cleaning makes use of different technique like data parsing (e.g., syntax errors), type mapping (e.g., casting), removal of duplication, and statistical methods for data analyses (e.g., mean value, standard deviation, clustering algorithms, imputation, etc.). Data cleaning focuses mainly on detecting data anomalies and missing entries, and then replacing them with plausible values. Among many techniques and methods used for cleaning, imputation is one of the frequently used ones. Imputation works by correcting anomalies and missing entries through replacing these data errors with plausible ones. One method of data imputation is by manually replacing the missing data via expert users, another is regression imputation where a regression model is used to predict the value of missing data entries based on the available data. Beside achieving data quality, data cleaning is also used to identify and attach meta-information to the data, which can be beneficial for the data analysis later ([Swapna et al., 2016]). The different techniques and approaches used for data cleaning are applied to many types of data sources such as text files, spreadsheets, databases, semi-structured data files (e.g., `XML`), log files, etc. Moreover, data cleaning is used in different types of applications like big-data systems, data mining, data warehousing, legacy systems integration, and machine learning applications as well.

In this work we are trying to enhance the framework VISPLORE to aid the user in the process of data cleaning through a user-friendly UI by allowing users to manually delete or replace faulty data entries. The framework will also properly visualize and represent certain DQ metrics that are relevant to the user. These DQ metrics, also referred to as meta-information, will allow the user to form a comprehensive understanding of the data, which can affect the decisions made for data analysis and data cleaning. All the developed features for data cleaning and meta-information visualization are showcased in the Table Lens View, which is one of the many data visualization viewing modes provided by `VISPLORE`. Our focus will be on structured datasets (e.g., relational tables), which are one type of data representation. Other types of data, which are beyond the scope of this work, include semi-structured (e.g., `XML`) and unstructured data (e.g., data expressed in natural languages) ([Batini and Scannapieco, 2006]).

This work consists of six chapters. After this introduction, the second chapter will provide a detailed overview on the state of the art of the topics related to DQ and data cleaning being applied in this work. The third chapter gives a detailed description of `VISPLORE`, its main features and the different modes it provides for data visualization. The Forth chapter describes the new functionalities added to `VISPLORE` to *(i)* support the visualization of meta-data inside the table lens view, *(ii)* keep track of cleansed data records inside the view by visualizing them differently, *(iii)* how these features will function inside dashboards with multiple views, and *(iv)* how these features will enhance the visualization and readability of different data attributes. In chapter five, we will evaluate the added features and discuss the challenges we faced during this work. Finally, the last chapter summarizes the key outcomes of the achieved work and gives an outlook to future activities.

CHAPTER 2

Data Quality and Data Cleaning

The processing of big data for the purpose of generating key performance indicators and extracting system development trends for data-based decision support requires the assurance of high quality of data that implies among others cleaning and insuring consistency of the processed data.

2.1 Data Quality

2.1.1 Data Quality Definition

Data quality (DQ), is a property that describes datasets. A dataset is of a high quality if *(i)* it fits its intended usage, *(ii)* it correctly represents real-world values, and *(iii)* its content is consistent by using the same format and structure for example. Another definition for DQ is: a characteristic that describes the reliability and usability of the data ([Dasu and Johnson, 2003]). In many modern ICT systems, DQ became a new functional requirement for the design of such systems, where these systems require sound and complete data to deliver reliable results. Lots of business these days relies on big data analysis for decision making, and any decision can be heavily affected by the quality of the analyzed data. Data-driven decision systems operating on incorrect data will deliver wrong results, which will cause wrong decisions being made and thus economic losses ([Ganapathi and Chen, 2016]). A study found that the cost of bad data in the year 2006 was nearly $30 billion. These costs include the cost of lost opportunities and wrong decisions based on bad data ([Lee, 2007]).

Errors that affect the data quality exist in the form of *data noises* or *data glitches*, the former are random and accidental errors introduced to the data due to human mistakes or failing measurement instruments, whereas the latter are errors introduced systematically to the data such as unreported changes in the layout or format of the data that causes parts of the system to operate on wrong data. Both data glitches and data noises are also referred to as *dirty data* ([Swapna et al., 2016], [Hellerstein, 2008]). Examples for *dirty data* include:

- Errors in the data such as typos.
- Missing entries.
- Dummy values such as dummy email addresses used for online account registration.
- Inconsistent values that are contradicting with other values.
- Faulty entries, where the representation of an entity x does not match its actual value in the real-world.
- Cryptic or hidden data.
- Data violating the business rules.
- Non-unique IDs.

The above-mentioned data errors can slip into operational data in many ways, either during the data collection stage in the form of data entry errors and measurement errors, during the data preprocessing and preparation stage in the form of distillation errors (i.e. errors that occur when raw data are summarized before entered into the database, to reduce the size or to perform some statistical analysis), or during the integration of multiple data sources from different systems in the form of data integration errors ([Hellerstein, 2008]).

Any system is prone to such types of dirty data, making DQ an important aspect for consideration in many data processing systems. When working with big data, Data quality issues presents a challenge toward making big data operational in businesses, one of those challenges is how to perform data cleaning on such big data. In data warehouse systems the impact of dirty data is huge because new data are constantly loaded into the system to keep the data fresh, and thus the quality of the data needs to be insured before loaded into the warehouse to achieve reliable results ([Rahm and Do, 2000]). DQ is also important in data mining systems, machine learning systems as well as traditional database application systems ([Swapna et al., 2016]).

While all the previously mentioned dirty data effect DQ, missing data will be a major focus in this work because of the bias that could be introduced to the analysis by missing data, especially when missing entries are replaced with wrong default values. A frequently occurring instance of missing data many analyses must deal with, is the presence of a huge number of missing records in a time-series dataset. In Section 4.2 we will examine some of the techniques used to handle missing entries in datasets.

2.1.2 Types of Data

To form a comprehensive understanding of DQ, the issues related to it, and how to fix them, we first need to understand the different types of datasets. Depending on the type of data in question, different tools and techniques can be applied to fix DQ issues, as well as different metrics for measuring DQ. In the following we will review the most common types of datasets found in many analysis systems, and the DQ challenges associated with them ([Dasu and Johnson, 2003]):

- Federated data: A dataset that results from joining (merging) data from multiple sources together, a practice mainly done in enterprises. The type of problems that can arise in this type of data are improper joins and conflicts.
- High dimensional data: Massive and big data, which raises the challenge of scalability and performance.
- Descriptive data: A dataset that consists of multiple tables with relationships between them. The challenge with this type of data is to keep it consistent and up-to-date to reflect the actual state of the system, e.g., resources' data tables must reflect the actual state of the inventory in a company.
- Time-series data: Also referred to as longitudinal data, a series of data records ordered by time such as the gas consumption for a city over the course of a year, such data could be then used for forecasting future behaviors. Data errors that could arise in time series data is related to synchronization issues, i.e., improper correlation of time series data. DQ issues related to time-series data have a high priority when it comes to data cleaning because data corruptions that occur over time can propagate to other data sources. Time-series data were used very often in this work, as most of the provided data samples contained time-series data (e.g., gas consumption, temperature readings, solar radiation, etc.).
- Streaming data: A sequence of data that get emitted at a high rate from a data source and are then accumulated. Temperature readings measured at hourly intervals and accumulated over the course of a year is an example for streaming data.
- Web data: Also referred to as scraped data, are data collected from various sources on the internet such as usage information extracted from web server logs. The internet is a major source of data, where lot of reports, statistics and other types of information get posted for easy access. The data found there however is typically messy since it does not have a uniform structure and is hard to integrate with other systems. Web scraping tools are mainly used to extract data from the web, which then needs to be cleaned because of low DQ. Data records like 2i2-555-i2i2 instead of 212-555-1212 or "my last name at domain dot com" instead of name@domain.com are typical examples for DQ issues found in web data.

Another classification of data is based on the type of data attribute ([Hellerstein, 2008, Dasu and Johnson, 2003]):

- Numeric attributes, also referred to as quantitative attributes, are numerical values (integers and floats) that measure the quantity of an attribute such as the number of employees, or the temperature. Statistical methods can be used to control the quality of this type of attributes.
- Text and categorical attributes: data attributes that are stored in the form of names or codes. Categorical attributes have no ordering between the values, and each dataset can use a different namespace for its categorical values (e.g., a "vehicle" in one dataset is called a "car" in another one). This prevents statistical methods from being applied to categorical attributes, and instead most of the approaches used to maintain the quality of this kind of data are context dependent and require a lot of scripting. Another example for DQ issues related to text attributes is when '0's are replaced with 'o's in address attributes or phone numbers.
- Descriptive attributes: patterns and statistical models that can be used with numeric and text attributes are not applicable here most of the time and require instead sampling techniques and human validation (e.g., *"Is equipment A really located at the geographical address mentioned in the database?"*).

2.1.3 Data Quality Metrics

While the defining criteria for high data quality in a given system are context dependent and subject to the application domain and the user's needs, certain data quality constrains are common and relevant to any datasets:

- Completeness: is a characteristic that describes the operational dataset containing no missing records, for example, having some customers' birthdate missing in the dataset. Another definition of completeness is *"the extent to which data are of sufficient breadth, depth, and scope for the task at hand"* ([Wang and Strong., 1996]). The opposite to complete data is data with missing entries, although sometimes missing might indicate that the actual value of an entry is irrelevant, and therefor omitted.
- Validity: a data value is valid if it complies with the domain and business rules, for example, the uniqueness of a data field across the dataset.
- Accuracy: describes how accurate the stored data is compared to its original source. For example, how accurate are a sensor's temperature readings. The opposite to accurate data is faulty.
- Non-duplication: the data has a one-to-one relation between the real-world objects and the stored records. This metric is sometimes also referred to as *uniqueness.*

- Consistency: similar data follow the same rules across the system, so that the recorded data does not cause any conflicts with other data or with business rules. For example, date records in the dataset should be using the same date format.

Other DQ constraints that are more specific to certain analyses and applications are:

- Derivation integrity: the correctness of data generated from combining multiple data pieces.
- Definition conformance: any data object should have complete details of the real-world object.
- Accessibility: data is always accessible on demand.
- Timeliness: data is kept up-to-date and stored with correct time stamps.
- Interpretability: the presence of metadata alongside the data, so that the user can understand and interpret the meaning of the data correctly.
- Suitability: the data being used is suitable and relevant for the analysis (e.g., not using outdated data).
- Extent of automation: requiring the least amount of manual intervention to complete a process.
- Successful completion of end-to-end process: does the data being used in the process deliver the correct outcome.

All the above mentioned DQ constraints are usually referred to as *DQ dimensions* or *DQ metrics* ([Dasu and Johnson, 2003]). In this work, we will deal very often with the DQ metrics *valid*, *missing*, and *faulty*, as the sample data often suffer from DQ issues related to these metrics.

As mentioned in the previous section, each type of data brings its own set of problems and challenges when it comes to maintaining a high level of DQ. For example, descriptive data require DQ metrics to be continuously validated, which can be an expensive process if dealing with big- and fast-growing data. Streaming data on the other hand require DQ metrics to be checked in real-time, which can become more challenging if we do not have access to the data stream all at once. Scraped web data, usually stored as text data (e.g., `CSV`, `CLF`), require information retrieval methods and natural language models to check the DQ metrics, making the process highly domain specific and challenging.

2.1.4 Measuring Data Quality

After establishing the definition for DQ and specifying the set of metrics that make data of high quality, we can measure the DQ in each dataset. While the correctness of an analysis can be measured by the number of faulty outcomes, DQ is measured by how much of the data comply with the static constraints, i.e., schema and key constraints in a DB, and the dynamic constraints enforced on the data, i.e., business rules. This varies depending on the DQ metric that is being measured. Below is how the most common metrics are measured:

- Accuracy: generally hard to measure especially if dealing with huge data. Sampling can be used, which can however introduce bias in the measurement.
- Uniqueness: can be measured by applying duplication removal techniques, and then counting the number of removed records.
- Completeness can be roughly estimated by taking a sample of the data and measuring the number of nulls and missing entries in it.
- Extent of automation can be measured by the number of clicks provided by the user in the GUI.
- Successful completion of end-to-end process can be measured by sampling of some process instances and observing the outcome.
- Accessibility: measured by the time between issuing a request to access the data and being able to view the data.

2.1.5 Causes for DQ Issues and How to Prevent Them

DQ issues can arise for different reasons and during different workflow stages of all information systems, and by avoiding some of these practices or altering them many DQ issues can be prevented ([Ganapathi and Chen, 2016]). Examples for practices that could result in DQ issues:

- Ad-hoc instrumentation: collecting data indirectly through a proxy, like measuring users' satisfaction with a software product based on upgrade adoption rates. Outliers in the collected data could result from misinterpretation of the data collected indirectly, e.g., from a survey. To overcome such an issue, automated data collection tools must be integrated into data processing systems like sensors in cyber physical systems (CPSs).
- Inconsistent data: occurs when humans are involved in the creation of the data. The correctness of the input data can be affected by the user interface perception, system's defaults, or having too many options for the user to choose from. Having too many optional fields for the user to fill in could result in a lot of null values,

which affects the quality of the data. The solution to this type of issue is to use a combination of human and computational data entry, using drop-down lists with predefined options in case of manual data creation and carefully choosing default values.

- Unclear ownership of data: as datasets and analysis reports get forwarded to, used and altered by multiple parties, it gets hard to keep track of the original owner and creator of the data as well as the most up-to-date version of the data. If the data contained unclear or ambiguous portions, it would be hard later to explain the meaning of those records if information on the original creator of the data gets lost over time, e.g., former employee. This type of issue can be resolved technically by introducing a versioning and auditing system to track changes and ownerships over time.

- Disconnection between data engineers and data consumers: in many data systems, the person who consumes the reports and analysis is different from the one who creates them. Miscommunication between the two could result in data quality and performance issues. This can be fixed by making the data platform easier to use by the average user and proper communication between the data engineers and stakeholders.

On the other hand, some practices have been proven to be successful when it comes to avoiding DQ issues in businesses that rely on big data to operate:

- Elevate data as an organizational-wide asset: data quality issues increase as data get exposed to new systems and users beyond the original intended use. When an organization connects multiple data sources of different departments together, data quality issues can be mitigated as analysis results can be validated against multiple data sources.

- Address business process issues that affect data quality: such as miscommunication between data consumer and data engineers, inconsistent data collection or neglecting certain measurements while collecting data.

- Being data-informed without overly relying on data: the captured data sometimes does not tell the entire story, which is way it is important to relay on human expertise.

- A role for centralized data teams: migrating the data sources and data processing tools of different departments to a centralized platform can address data quality issues easier.

In the next section we will explore different data cleaning techniques used for handling DQ issues.

2.2 Data Cleaning

2.2.1 Data Cleaning Definition

Data cleaning, also called *data cleansing*, *data scrubbing* and sometimes referred to as *data preprocessing*, is any processing done to raw data before analysis (not in real-time), whether while collecting the data or after writing it into the system's dataset. The purpose is to remove dirty data entries from the dataset to achieve a high data quality ([Swapna et al., 2016, Hamad and Jihad, 2011]), which in return benefits the analysis. In one instance in a system for classifying data, the result's accuracy is increased 100% by cleaning the data first ([Ganapathi and Chen, 2016]). In a fraud prediction system, the detection rate has improved from 62% to 91% by cleaning the data first before applying the machine learning model ([Chu et al., 2016]). Data cleaning is an expensive process that consumes between 30% to 80% of the analysis time ([Ganapathi and Chen, 2016, Dasu and Johnson, 2003]). The process of data cleaning becomes especially challenging if dealing with dig data where the data are constantly evolving, and their scale and complexity makes the cleaning process more difficult ([Chen and Jiang, 2014]).

Data cleaning is all about defining strict rules for what clean data and what faulty data are (*the definition of data error, error detection*), and then applying automated data-cleaning-tools to correct and clean the data (sometimes cleaning can be performed manually by humans). Beside achieving data quality, data cleaning is used for analyzing the data to identify and attach *meta-information* to the data. The existence of meta-information, or*metadata*, is important for the data cleaning process because metadata can be used to write data cleaning rules and constraints to enforce DQ metrics. Metadata are extracted by analyzing the data using data profiling and data mining, another way to obtain metadata is manually by domain experts. Data profiling works by analyzing individual data attributes, and can deliver meta-information such as data type, value range, frequency, uniqueness, occurrence of null, etc. Data mining on the other hand analyzes large datasets to discover meta-information across multiple attributes, which can be used to generate business rules to fill up missing values ([Rahm and Do, 2000]).

Data cleaning techniques are used in many information systems such as data mining, machine learning, and data warehouse systems ([Swapna et al., 2016]). Depending on the type and size of the data, different cleaning approaches and techniques are required to achieve data quality. For example, if we are working with linear aggregations such as the average, data cleaning can be ignored, as the results can be correctly estimated from small, clean samples of the overall data. This however fails when working with complex systems such as machine learning and statistical model-building systems, as results obtained from small samples of the data might contradict with results from the aggregate (Sampson paradox).

2.2.2 Data Cleaning Types

As mentioned earlier, based on the size, the type of data, the type of errors and the DQ metrics that are targeted, different cleaning methods are applied. One of the most applied methods is to distinguish between qualitative and quantitative data cleaning ([Hellerstein, 2008]).

- Qualitative data cleaning uses constraints, rules, or patterns to detect errors in the data. For example, qualitative data cleaning can detect a variation in the salaries of two employees with the same employment level.
- Quantitative data cleaning uses statistical methods to detect outliers in the data. For example, using standard deviation and the mean value, quantitative data cleaning can detect salary anomalies.

2.2.2.1 Qualitative Data Cleaning

Qualitative data cleaning is sometimes also referred to as rule-based data cleaning or constraints-based data cleaning ([Chu et al., 2016]). In this cleaning approach for detecting errors in the dataset we specify: *(i)* the type of errors we want to capture, *(ii)* which techniques will be used for that, and *(iii)* at what stage in the business intelligence (BI) stack will those techniques be applied (Figure 2.1). Normally integrity constraints (ICs) are mainly used to improve the quality of the database schema and not the data itself. By using integrity constraints ICs to define data quality rules, we can detect a variety of data quality issues such as missing values, integrity and inconsistency in the data. One of the most common integrity constraints used are functional dependencies (FDs) in relational databases. Beside functional dependencies, integrity constraints include other types of constraints:

- An extension to FDs called conditional functional dependencies (CFDs) is proposed, where the dependencies satisfy a pattern and apply conditionally to a subset of the relation and not to the entire relation as opposed to FD ([Bohannon et al., 2007]). CFDs are useful if integrating datasets from multiple resources, where dependencies from a subset would only hold conditionally in the integrated dataset.
- Denial constraints (DCs) use first order logic to express constraints that contain order predicates (e.g., *"If two persons live in the same state, the one earning a lower salary has a lower tax rate"*), or to express constraints that compare different attributes in the same predicate (e.g., *"It is not possible to have a single tax exemption higher than the salary"*). Both these constraints cannot be modeled as FD/CFD and that is where DC come into use ([Chu et al., 2013]).
- Key constraints as another example of ICs, are constraints applied to data attributes to prevent null or duplicate values from being entered. Key constraints can be used to detect duplications or missing values in a dataset.

These dependencies and constraints can be deployed to validate the source data against quality violations, or during the data processing stage to discover anomalies in the target data once business logic rules become available, for example, constraints on total budget can only be enforced after aggregating cost and expenses.

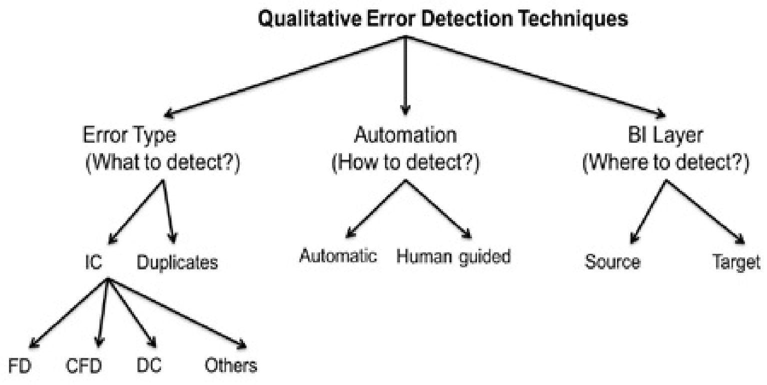

Qualitative Error Detection Techniques
Error Type (What to detect?)
Automation (How to detect?)
BI Layer (Where to detect?)
IC
Duplicates
Automatic
Human guided
Source
Target
FD
CFD
DC
Others

Figure 2.1: Classification of qualitative error detection techniques and where to apply them ([Chu et al., 2016]).

Integrity constrains require domain experts to formulate the constrains, mostly in logical form, which is hard to be done by the average user. One proposed approach for an interactive and user-friendly data cleaning system, FALCON, uses SQL update queries (SQLU) to generate the cleaning rules and perform the data cleaning ([He et al., 2016]). While Integrity constrains rules define data errors as violations and detect them but without providing any fixes, SQLU goes a step further beyond the error detection by providing deterministic fixes to the data. FALCON can be used by non-expert users and contains no predefined data cleaning rules. With FALCON, the user can detect some data quality issues (e.g., by browsing the data) and then provides a fix for the issues, which gets translated to an SQL update statement. The system can then suggest a list of additional fixes (SQL update queries, some of which are valid and others invalid), and once the user verifies one of them, that fix will be applied to the rest of the data and the suggestion list will be refined.

Once the error detection stage is completed, it is time to repair (clean) the data. Like in error detection, three questions need to be answered before we can determine our repairing strategy:

1. *What to repair*: do the actual data need to be repaired, the integrity constraints such as obsolete business rules, or both the data and the integrity constrains? And if the actual data need to be repaired, will we handle one error type at a time or multiple types?

2. *How to correct*: is the correction process fully automated or is a human involvement required? And if a human is involved, is it to verify the repairs or to train a machine learning model that can make correction decisions?

3. *Where to repair*: either the operational database is modified and repaired, or a model that describes the possible repairs to the databased is build. The first approach results in destructing the original database, whereas in the second approach queries are answered against the repairing models and the data are left unmodified.

While most of the cleaning techniques are fully automated and do not require a human intervention like detecting FD violations ([Chu et al., 2016]), other techniques may require a human operator at some stages in the cleaning process. Other systems for detecting and cleaning data duplicates are proposed that use crowdsourcing platforms to assign batches of data to human operators, to detect duplicate entries in the data ([Gokhale et al., 2014, Wang et al., 2012]).

2.2.2.2 Quantitative Data Cleaning

Quantitative data cleaning is also known as statistical-based data cleaning. In systems with large data, statistical techniques can be applied to the data to build a model (code) that can discover patterns in the data to solve problems and make predictions. Systems that use this process are called machine learning (ML) systems. Large datasets can boost the training of ML models, which results in more accurate predictions, but with large data the ML model will be exposed to more dirty data, which can affect the reliability of the ML model. Cleaning the training data first can increase the ML model's reliability noticeably. In this perspective for data cleaning, the focus is on repairing duplicated, missing, and erroneous values by applying cleaning techniques and algorithms based on statistical analysis methods:

- Active learning in crowdsourcing: while crowdsourcing is actively being used in data cleaning, i.e., where human participants get assigned tasks for cleaning the data such as validating businesses phone numbers and verifying new information and facts, large data and crowds are hard to scale in such systems. That is where active learning as an algorithm gets applied to select a sample of records to be cleaned by the crowd, and then use those cleaned records as a training set to train a ML model that cleans the rest of the data ([Gokhale et al., 2014, Wang et al., 2012]).

- Aggregate queries: in large datasets, processing aggregate queries such as sum, count, and avg can take a lot of time. Sampling-based approximate query processing (SAQP) techniques are used to get fast results by applying aggregate functions to a sample of the data. The effectiveness of this technique can can be lower if the sample contains dirty data. Since cleaning the entire dataset is not a feasible solution, the best approach is to clean the sample data, as in the *SampleClean* framework ([Wang et al., 2014]).

- Machine learning: as described earlier, the presence of dirty data in the training dataset for a ML model will affect the reliability of the model. *ActiveClean* selects the most valuable and clean data to update the ML model gradually instead of retraining it ([Krishnan et al., 2016]).

2.2.3 Data Cleaning Concept

One approach for data cleaning identifies the different stages of data usage, since each stage can have a different effect on the DQ, and then multiple techniques for data cleaning are implemented at each stage. *Data usage* can be divided into six stages as follows ([Dasu and Johnson, 2003]):

1. Data Collection: problems at this stage can result from manual data entry, no uniform standards for data format, measurement errors, and replacing measured/recorded data with approximations. The best approach to handle DQ issues at this stage is preemptively, preventing DQ issues from entering the data in the first place. For example, the use of integrity constraints, defaults, drop-down lists, etc. The other approach is retrospective, cleaning dirty data. For example, using data cleaning tools for duplicates removal, normalization, etc.

2. Data Delivery: DQ issues at this stage can result from inappropriate data preprocessing (converting nulls to defaults), loss or alteration of data during transmission. The solutions to DQ issues at this stage: building reliable transmission protocols, verification (e.g., checksums), and verifying the data flow into the system to detect errors and then tagging them.

3. Data storage: the main causes of DQ issues here are missing metadata and inappropriate data models (i.e., missing timestamps, incorrect normalization, etc.). Some of solutions to handle these issues are by incorporating metadata and analyzing the data to detect issues.

4. Data integration: is the stage at which data from different sources, mainly with different structures, are combined. Problems at the data integration stage can occur when integrating heterogenous data (e.g., different formats, no common key), legacy data (e.g., spreadsheets vs. SQL tables), or if dealing with time synchronization (e.g., Does the data refer to the same time window?). The solutions to integration issues would be in the use of integration tools and approximate matching.

5. Data retrieval: here, problems will result from not understanding the data and thus retrieving the wrong data, for example, join mistakes in an SQL select statement. Understanding the structure of the data and the relationship between the tables will prevent retrieval problems.

6. Data analysis and processing: insufficient domain expertise, the scale of the data, and performance issues, all can cause of problems during the data analysis stage. The solution to problems here is mostly context dependent.

Another more general approach for data cleaning applies the DQ metrics and rules to the data after they are aggregated, and before analyzed and processed by the system, as follows: The first step in the process is to gather the data from all the different sources such as data collected from sensing devices, files and the web. Then the data gets aggregated and loaded into the analysis system. Finally, the data is validated against schema specifications (static quality constraints) such as key uniqueness and functional dependencies, followed by the validation of the business rules and other DQ metrics (dynamic constraints). The last step is the most important step and is deeply context dependent and requires close interaction from domain experts. Data faults and glitches that get detected in the process are corrected if possible, and the clean data satisfying the quality metrics will be used to quantify and measure the DQ to generate statistics about the data (e.g., the percentage of missing data) and to identify the sources of dirty data (e.g., certain data files or sensors). Moreover, any lessons and knowledge acquired from the previous steps can be fed back into the system to improve the DQ constraints and rules (Figure 2.2). Regardless of the approaches used for data cleaning (i.e., when and where to perform cleaning), the actual cleaning techniques and methods used (i.e., how to perform the cleaning) are determined by many factors comprising: *(i)* the type of data source such as text files, spreadsheets, or SQL databases, *(ii)* the type of data entries to be cleaned such as numerical, strings, or other types of date values, and *(iii)* the type of data issues such as missing, duplicated, erroneous, or dummy data) ([Swapna et al., 2016]).

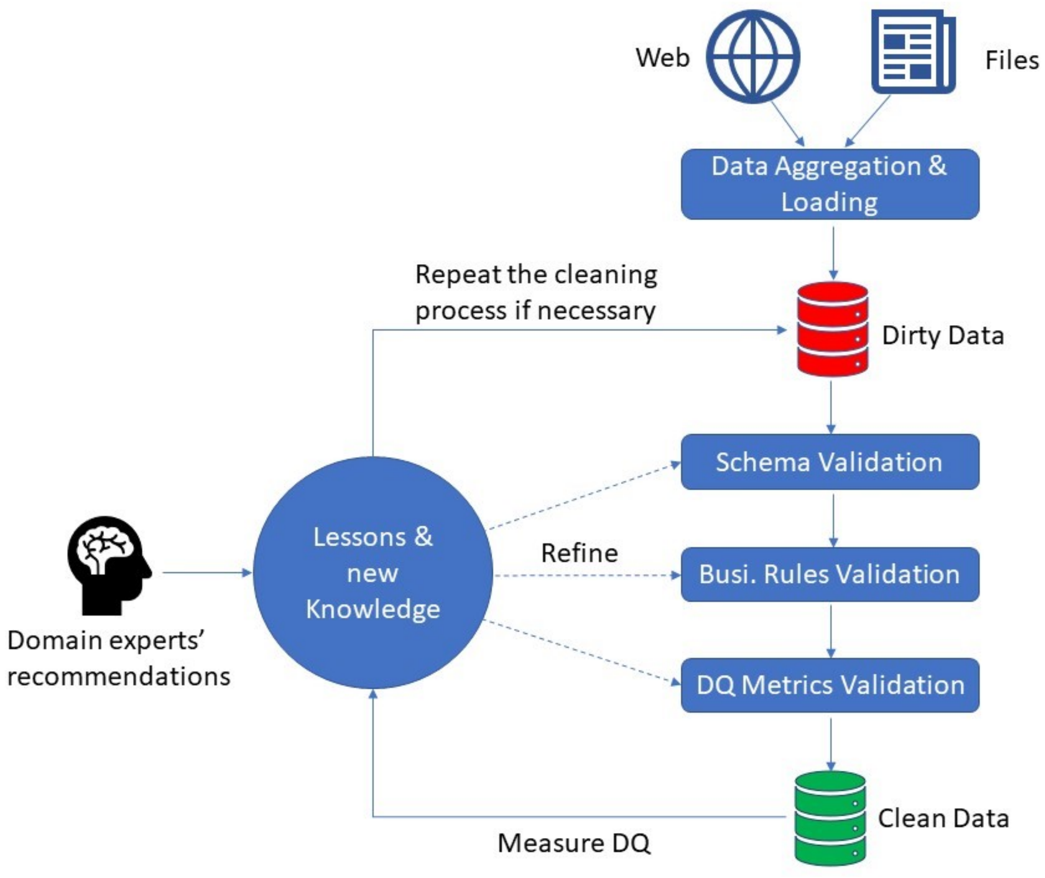


Figure 2.2: Systematic concept for the process of data cleaning.

In Figure 2.2, after data are aggregated in a single data source, different DQ constraints (integrity constraints, business rules, duplicates removal, etc.) are applied to obtain a cleaner version of the data. The quality of the cleaned data is then evaluated to generate statistics and new knowledge about the cleaning process, which can be used to modify the DQ constraints with the help from domain experts. The cleaning process can be repeated until the required DQ metrics are met.

2.2.4 Data Cleaning Techniques

Some basic data cleaning techniques involve using dictionaries to fix misspellings and typos in data entries, address books to correct wrong addresses, and zip codes and attribute dependencies (e.g., birthdate/age) to fix wrong values or fill in missing ones. Standardization of values such as date/time format, upper/lower case for text fields, etc. can avoid and fix many data quality issues ([Rahm and Do, 2000]). In this section we will outline the various techniques and mathematical methods used for cleaning different types of data issues ([Dasu and Johnson, 2003]).

2.2.4.1 Missing Data

Missing data is one of the most common data issues in many datasets and handling missing data is a major task before performing data analysis. Missing values and entries in data can originate from multiple sources such as data segments getting lost during transmission, omitting some data entries by mistake if manually entering data, faulty sensors or measuring instruments that fail to capture some data. Another source for DQ issues arises when using the same value to represent defaults and missing values. Not always does a missing entry in a dataset mean that a data attribute is missing due to some errors in measurement or storage, rather than it might not be relevant. For example, a customer not subscribed to a service, omitting such entry will not affect the *completeness* matric of the data. In all cases, it is important to examine whether missing data may lead to bias in the analysis because increasing amount of missing data will cause more threat to the study. The most practical approach to mitigate the effect of missing data to an analysis is to increase the sample size to compensate the losses. Other approaches toward missing data are to delete data records with missing values or estimating the missing values. In retrospective studies for example, like using data from patient medical history, missing data here are very expected. Of course, deleting a large amount of missing data may lead to bias because we are excluding cases that are associated with those missing entries which could affects the study. In federate datasets for example, anywhere between 30% to 70% of the data would be lost if we deleted all data records with a missing field. Therefore, researchers must be careful when they delete data or replace them with predicted values. Also, since missing data does not occur randomly and are mostly a sign of a problem in the system, analyzing the missing data could help us identify the cause of the issue such as discovering a faulty device that is not transmitting/collecting readings. There are multiple techniques and methods in use to detect missing data such as:

- Performing simple checks of the data during the transmission and storage to identify any missing values, and then referring to the source data for correction. For example, number of files and their sizes, number of records and attributes, etc.

- Using historical information to validate the correctness of data, e.g., a sensor device known to deliver n readings in one hour, but is now sending less readings.

- Validating the data against accurate estimates such as total counts, averages, medians, and other values.

- In time series data, when using the same value (0 for example) to represent both defaults and missing values, the flip-flop pattern can be used to identify weather a value is missing or set to the default value.

Beside ignoring or removing data records with missing fields, *data imputation* can be used to replace missing values. Data imputation is the process of guessing the value of a missing entry and replacing it with the guessed value. One of the simplest imputation techniques is *point estimate* that estimates the value of a missing attribute while ignoring any relation it has to other attributes, such as using the mean or median value to replace missing entries. Another more complex imputation technique uses *regression* methods to impute multiple values and accounts for the relationships between attributes. Regression, also known as statistical imputation, is applied to predict missing values by considering the variable with missing data as the dependent variable, and all cases with no missing data are used to generate the regression equation. The resulted equation then can be used to estimate the missing values, as in the following example:

Weight	Age	Height	Health index
20	2	10	5
15	5	9	3
25	5	10	MISSING
20	4	MISSING	MISSING
10	1	MISSING	MISSING

$Height = \alpha + \beta_1 Age + \beta_2 Weight$
$Height = \alpha + \beta_1 Age + \beta_2 Weight + \beta_3 Height$

Where α is the error term and β_i are the regression coefficients.

Different error terms α can produce multiple regressed values for the imputation, which are then analyzed and combined to produce a single reliable value set. Other imputation techniques include cold-deck imputation, hard-deck imputation, probability distribution-based imputation, replacing with unknown, constant value ([Musil et al., 2002, Huisman, 2000]).

Another technique fills up missing entries using an algorithm called *decision tree induction* ([Dara and Satyanarayana, 2015]). For each attribute (column) in the dataset a decision tree will be constructed. The root of the tree is the data attribute in question and the nodes represent tests (questions) for the other data attributes. Each node has two branches: yes or no. The missing entries will be filled using the leaf nodes (Figure 2.3).

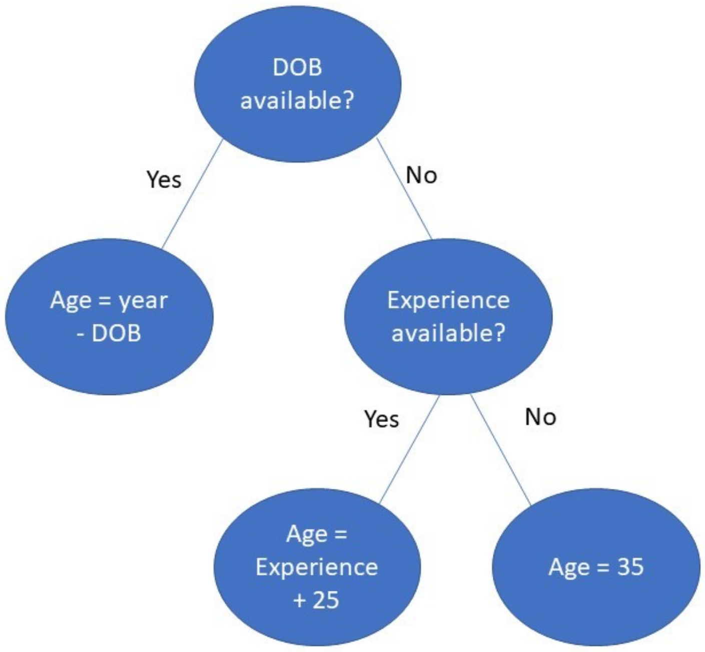


Figure 2.3: Employee age decision tree.

2.2.4.2 Incomplete Data

Usually data is labeled as incomplete if the whole data originally exists but part of it is systematically omitted upon storage or delivery. Incomplete data is somehow like missing data when it comes to the affect it has on the analysis by causing bias and uncertainty. Two types of incomplete data are to be distinguished: *Censored* and *truncated data*. "*Censored data* are data and measurements that come with uncertainty, because of some missing data or the lack of measuring tools." ([Dasu and Johnson, 2003]). Data are censored if the values of some data entries are replaced with other estimates, either because the actual values are irrelevant, e.g., *all annual incomes above 100K will be treated the same and reported as* $\geq$ *100K*, or hard to capture, e.g., *a scale that cannot measure above 100 kg will give the same estimate readings for all weights above that boundary.* If we were replacing censored data with defaults, for example, the analysis could give inaccurate results. Using histograms such anomalies can be identified in the form of spikes in the histogram as seen in Figure 2.4. On the other hand, data are truncated if portions of the data that lay beyond a boundary value are neglected and removed from the dataset altogether, e.g., *customers who spend less than a certain amount a year get dropped from the database.* Unlike censored data, in truncated data it is not just the variable of interest that we do not have full data on, all the data from that case are dropped from the dataset.

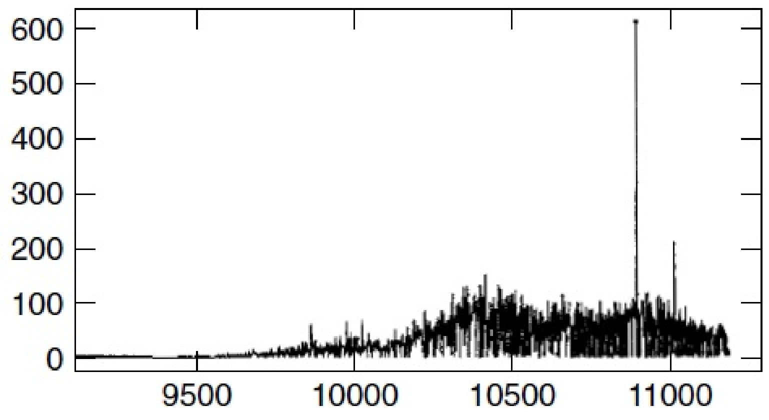


Figure 2.4: Data anomalies such as censored data can appear in a histogram as spikes ([Dasu and Johnson, 2003]).

2.2.4.3 Duplicates

When merging and integrating data from multiple sources the biggest problem is duplicated data, i.e., overlapping data, which needs to be solved by data cleaning. Duplicates elimination is usually performed on multiple data sources, cleaned in advance, as follows: similar records that refer to the same real-world object are identified first, and then merged into one record that contains all the relevant attributes without redundancy. The challenge with duplicate elimination is how to match similar records, referred to as the *instance matching* problem ([Rahm and Do, 2000]). The simplest case is if each data record has a unique identifying attribute such as a primary key: in the case of a single data source, data records are sorted based on the identifying attribute first, next neighboring records in the sorted data are checked for match. For multiple data sources, data are joined first before sorted and then matched. In the absence of identifying attributes, fuzzy matching is used to identify similar records based on some defined matching rules (often measured by a numerical value between zero and one). For example, one matching rule could determine that two person-records are the same if portions of the name and address match. Wildcards, character frequency, edit distance, and other techniques are useful for fuzzy matching string attributes.

2.2.4.4 Dummy Data

Dummy data are data that contains no useful information and only act as placeholder, for example, dummy email addresses used for online accounts registration. There are no specific techniques for detecting and fixing dummy data. Every system can define some patterns and categories specific to the business domain and then deal with the dummy values accordingly, either by replacing them with useful values or by ignoring the dummy values in the analysis process altogether ([Swapna et al., 2016]).

2.2.4.5 Outliers

An outlier is an observation that lies an abnormal distance from other values in a random sample from a population (Figure 2.5). There are many reasons for outliers like Some entries in the sample are extreme or data were scaled incorrectly, also an error could be occurred during data entry. Sometimes outlier could be a true of a rare occurrence or seems to noticeable affect the results, in this case the results of analysis should be reported with and without outliers. Otherwise, it is better to remove outliers if they do not affect the data or cause bias. If the outliers occurred due to an entry mistake, assigning them to new values like mean could be an appropriate approach.

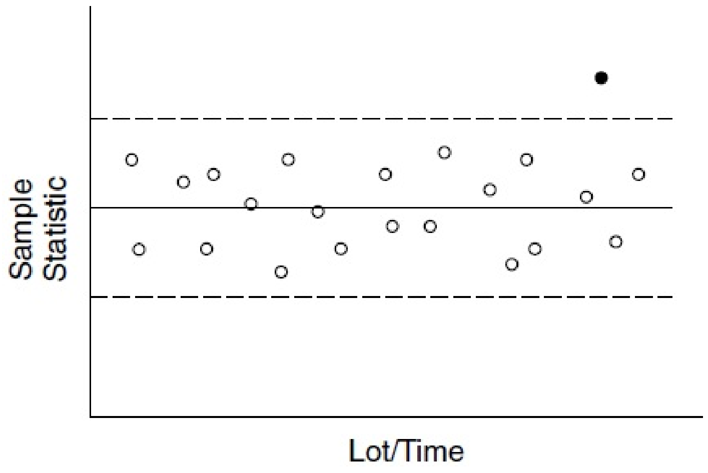


Figure 2.5: Example for an outlier in a sample set ([Dasu and Johnson, 2003]).

2.2.5 Examples for Data Cleaning Systems

- One advanced data cleaning system uses a parallel algorithm developed in MapReduce ([Chen and Jiang, 2014]) that runs on a distributed system, where the massive data is distributed among many clusters and analyzed in parallel to find missing or inconstant data and clean it.
- In data warehouse systems, data cleaning is an important task to perform before loading the data into the data warehouse to insure correct and reliable reporting and analysis results. Usually the process of data cleaning in data warehouse systems involves the following steps:
  - Combining data from multiple sources into one homogeneous data structure.
  - Removing redundant values.
  - Resolving inconsistencies in data due to heterogenous data sources (e.g., unifying measurement units, data format, etc.)

Many techniques to ensure DQ in data warehouse systems are applied in practice ([Hamad and Jihad, 2011]). First a set of rules for each data attribute in a data source will be defined, these rules will be used later for the purpose of data cleaning. Beside the set of rules, a list of all the data attributes related to a data attribute is also defined. All the data attributes involved in the cleaning process here are quantitative attributes. Once the data is loaded into the warehouse, the user will select what data attributes to load, and the algorithm will use the rules related to the selected attributes to check the data against many types of errors such as formatting errors, lexical errors, constraints violation, missing entries and duplication, and then correct them. The list of related attributes mentioned earlier is used in conjunction with the rules to fill in missing entries and correct faulty values. In an employee database, the *salary* attribute would have the following attributes related to it: {Base salary, Position fees, Childs Number, Service years, Degree}, and the following set of rules:

– Salary must be numerical
– Not negative
– Not missing
– $Salary = f(Basesalary, Positionfees, Childs, Serviceyears, Degree)$

When loading the data into the warehouse, the above rules and attributes will be used by the system to detect dirty data and perform data cleaning on the *salary* attribute. As we can see, the related attributes are used here by the rules to fill in missing *salary* entries.

- The decision tree induction algorithm described earlier in 2.2.4.1.
- FALCON, an interactive and user-friendly data cleaning system, mentioned earlier in 2.2.2.1, which uses SQL update statements (SQLU) to generate cleaning rules and perform cleaning on a database([He et al., 2016]). For any given update query:

`UPDATE` *Table_T* `SET` *Attribute_A = correct_value* `WHERE` *X = t[X]*

Where X is an arbitrary subset of attributes of relation R, which can range from the empty set $\emptyset$ to all attributes in R. The number of possible repair rules that can be derived from such a query is $2^{|R|}$: where $|R|$ is the arity of the relation R (the number of attributes). If the user detects a data quality issue (e.g., by browsing the data), she can provide a manual fix for that issue, which gets translated to a `SQL` update statement. All the possible repair rules, i.e., update queries, that could be derived from the user's fix will be organized in a lattice graph (Figure 2.6) that shows the containment relation between different repair rules. For any two repair rules Q_1 and Q_2, the most specific rule (query) is a rule contained by both Q_1 and Q_2, in other words, a rule that covers all the common records updated by Q_1 and Q_2. The most general rule is a rule that contains both Q_1 and Q_2, in other words, a rule that covers all the records updated by both Q_1 and Q_2. The edge from node Q to $\acute{Q}$ (each corresponds to an update query) indicates that Q is contained in $\acute{Q}$ (denoted by $Q \leq \acute{Q}$). Consequently, if a rule is marked as valid by the user, then all rules contained in it are also valid, and if a rule is marked as invalid than any rule containing it will be invalid, i.e., if one query is valid, then any query that is more specific is also valid; conversely, if it is invalid, then any query that is more general is also invalid. As the user makes changes to the data, i.e., updates the data by validating/invalidating suggested SQLU queries, and provides new configurations for the system, i.e., new *where* condition in the SQLU queries), the search space will become dynamic.

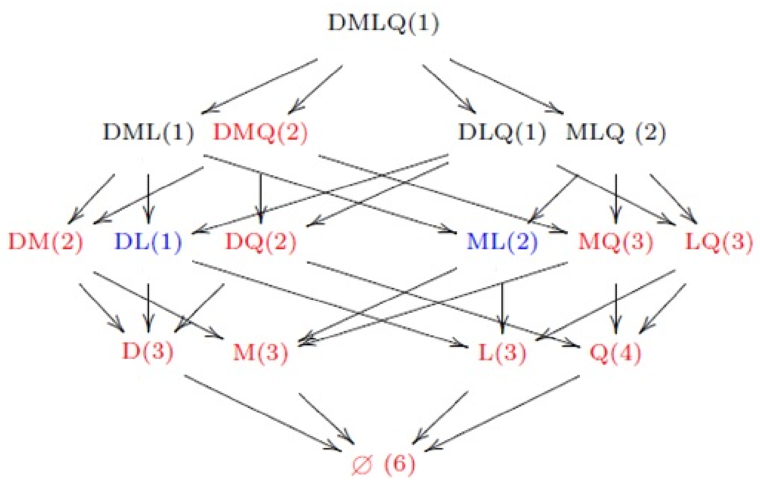


Figure 2.6: Lattice graph showing SQLU queries, and the containment relations between them ([He et al., 2016]). D, M, L and Q are abbreviations for the attributes (Date, Molecule, Labor, Quantity) of a data table. Each node represents an update query with D, M, L, Q indicating the attributes that appear in the WHERE clause, and the number of records affected by the update query. Red nodes represent invalid queries invalidate by the user, blue nodes are maximal (most general) valid queries, and the rest are valid queries.

2.2.6 Data Cleaning Challenges and Future Works

From the previous systems and approaches used to clean data and maintain the DQ, we can summarize some of the challenges in the data cleaning process ([Chu et al., 2016, He et al., 2016, Chu and Ilyas, 2016]):

- Scalability of data cleaning techniques to big data and rapidly growing data, while maintaining accuracy and performance.

- User engagement in the data cleaning process, such as collecting the user feedback to identify new data quality rules.
- Semi-structured data sources like `XML`, `CSV` files and unstructured data sources such as text files present a big challenge when cleaning data because their lack of schema. Schema-based data sources such as databases can use integrity constraints, triggers, and other schema related constraints to enforce data quality metrics. Such constraints are missing in data sources without schema ([Rahm and Do, 2000]).
- In the era of *CPSs*, the increasing number of sensors poses a challenge for data quality and data cleaning.
- If personal data are involved in the cleaning process, privacy becomes a concern.
- Create fast cleaning systems with continuous user interaction, especially if dealing with large-scale data with many attributes.

Some of the trends and future work in the data cleaning domain that are worth pointing out are:

- Introducing new integrity constraint languages that can help the user define stronger data quality rules to detect additional types of data errors.
- Cleansing and maintenance of master data such as knowledge bases.
- Involving humans in new data cleaning domains other than duplications cleaning, such as repairing integrity constraint violations.

CHAPTER 3

VISPLORE: A VISual exPLORation Framework for Data Analysis

VISPLORE is an application framework developed by the *Virtual Reality and Visualization Research Center GmbH (VRVis)* for visually supported knowledge discovery in large and high-dimensional datasets. VISPLORE allows the user to visually represent big and complex multivariate datasets using a variety of viewing modes such as table lens, histogram, scatter plot, and many other views ([Piringer et al., 2009a]). The wide range of functions and tools provided by VISPLORE allow expert users in many application fields, such as the energy sector, to solve a variety of tasks in analysis and statistical modelling. Beside knowledge representation, visual data analysis in VISPLORE can be used to detect anomalies in the data and to provide comprehensive overviews of the data structure ([Arbesser et al., 2017]). However, the broad range of functions VISPLORE offers can overwhelm inexperienced users, hence comes the need for dashboards that offer a selected set of functions and views, tailored for the needs of specific tasks and users.

3.1 Features

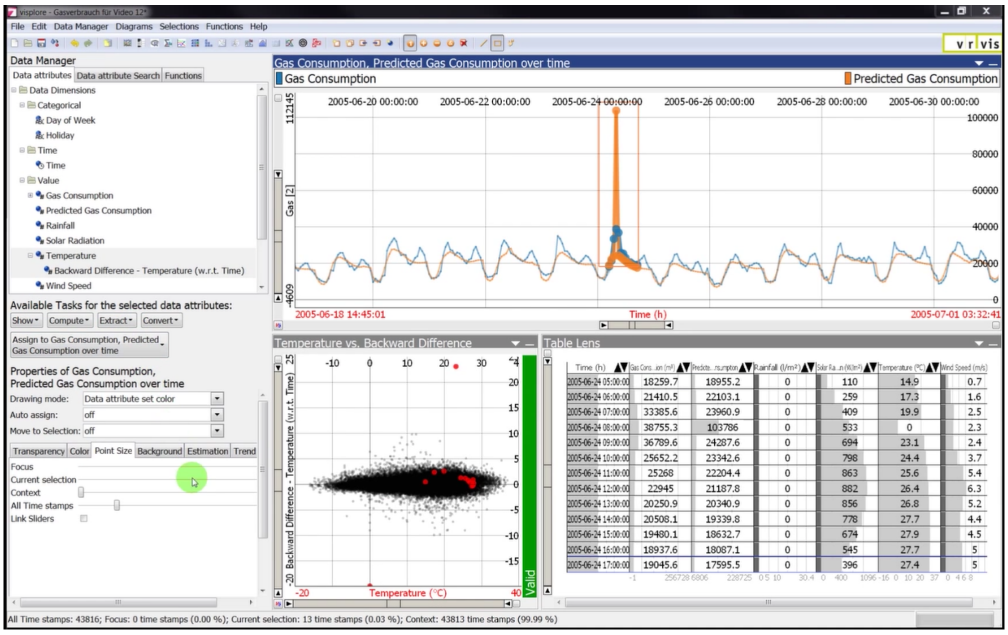
VISPLORE leverages multithreading techniques to support interactive exploration of huge multivariant data in real time with little response time, even when working with millions of entries and hundreds of dimensions. This continuous user interaction is supported through a multithreaded architecture that splits the execution across two layers of threads; the main application thread and the visualization threads. The application thread handles user requests like triggering updates and changing parameters. The visualization threads are responsible for handling expensive computations in the views, where a new thread is spawned for each view ([Piringer et al., 2009b]). VISPLORE provides more than ten visualization viewing modes that support many visualization techniques for multivariant data, from 2d and 3d scatter plots to histograms and other views. The user can assign data columns to views and then rearrange the columns within the view. Multiple views can be linked together in `VISPLORE` to create task-tailored dashboards. Datasets are imported into `VISPLORE` from multiple sources such as `SQL` databases, `Excel` spreadsheets and `CSV` files. Any modifications made to the data in `VISPLORE` can be exported to external data sources or copied to the clipboard.

3.2 Views

Multiple views are supported in `VISPLORE` for data visualization such as 2d scatter plots, histograms and table lenses.

3.2.1 2d Scatter Plot

In a 2d scatter plot, the values of two data variables are assigned to the horizontal axis (X) and the vertical axis (Y) of a diagram. The points of a diagram show the correlation between the two variables. By using colored points, a 2d scatter plot can show a third variable. A 3d scatter plot is an extension to the 2d scatter plot where variables and their relationships will be visualized in a 3-dimentional space. The bottom-left view in Figure 3.1 shows how a 2d scatter plot view is visualized in VISPLORE ([Pfahler, 2015]).



Drawing mode: Data attribute set color
Auto assign: off
Move to Selection: off

Time (h)	Gas Cons...ion (m³)	Predicte...nsumption	Rainfall (l/m²)	Solar Ra...n (W/m²)	Temperature (°C)	Wind Speed (m/s)
2005-06-24 05:00:00	18259.7	18955.2	0	110	14.9	0.7
2005-06-24 06:00:00	21410.5	22103.1	0	259	17.3	1.6
2005-06-24 07:00:00	33385.6	23960.9	0	409	19.9	2.5
2005-06-24 08:00:00	38755.3	103786	0	533	0	2.3
2005-06-24 09:00:00	36789.6	24287.6	0	694	23.1	2.4
2005-06-24 10:00:00	25652.2	23342.6	0	798	24.4	3.7
2005-06-24 11:00:00	25268	22204.4	0	863	25.6	5.4
2005-06-24 12:00:00	22945	21187.8	0	882	26.4	6.3
2005-06-24 13:00:00	20250.9	20340.9	0	856	26.8	5.2
2005-06-24 14:00:00	20508.1	19339.8	0	778	27.7	4.4
2005-06-24 15:00:00	19480.1	18632.7	0	674	27.9	4.5
2005-06-24 16:00:00	18937.6	18087.1	0	545	27.7	5
2005-06-24 17:00:00	19045.6	17595.5	0	396	27.4	5

All Time stamps: 43816; Focus: 0 time stamps (0.00 %); Current selection: 13 time stamps (0.03 %); Context: 43813 time stamps (99.99 %)

Figure 3.1: A screenshot of VISPLORE showing a session with different views. The bottom-left view is a 2d scatter plot, and the bottom-right view is a table lens view.

3.2.2 Table Lens

Figure 3.1 (the bottom-right view) shows the table lens view in action. A dataset is visualized as a table, with each data attribute being displayed as a column in that table. The table can be sorted in an ascending or descending order based on a column, and the columns can be rearranged as needed. Each cell in the table lens displays the data value as percentage bar relative to the maximum value in the current column, and on the far-left of a cell a small flag called the *per-value meta-information flag* (or the *validity flag*) is displayed to indicate the status of the data entry (i.e., valid or missing). Zooming out of the table lens view allows the user to view more data rows until each the data column in the view is displayed as a vertically-oriented histogram, as seen in Figure 3.2 (the bottom-right view).

The table lens view represents one of the basic modes for data visualization in `VISPLORE`. Our focus in this work will be on the table lens view, as all the new features are added to this viewing mode, and all the results from this work will be illustrated through table lens views.

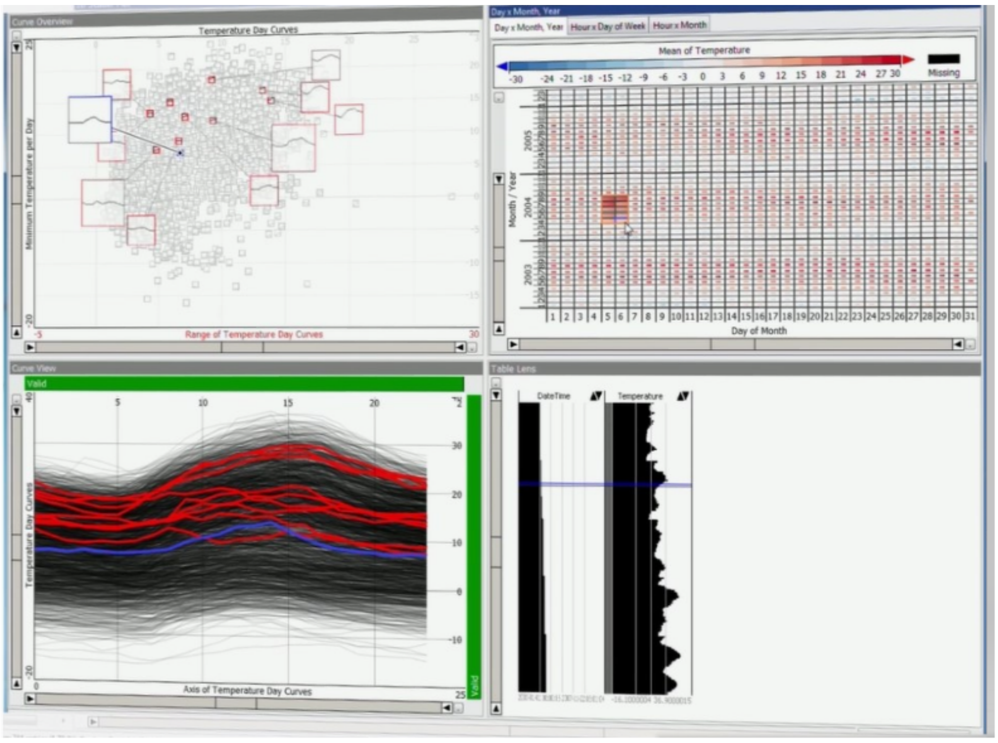


Figure 3.2: A screenshot of VISPLORE showing a session with different views. The bottom-right view shows a zoomed-out table lens view, where each data column is visualized as a histogram.

3.2.3 Histogram

Histogram is a plot that shows the frequency distribution of continues data such as numerical values. Histograms are used to relate and visualize multiple variables (data columns), making histograms suitable for data inspection to detect outliers and to visualize the distribution of data, both of which are easy to spot by the user. Spikes and V-shaped valleys in a histogram indicate problems in the data such as inexplicable preponderance of a given value or data being lost. Outliers in the data are those that appear in sparsely populated areas, not in the company of other data points. Figure 3.2 (the bottom-right view) shows the visualization of a histogram in `VISPLORE`.

3.3 Dashboards

Dashboards are user interfaces that resemble the functionality and purpose of a dashboard in a car, they combine information in a single board and present them in an organized, easy-to-read manner. Using a variety of data visualization elements (i.e., diagrams, charts, and coloring legends), dashboards can visualize key performance indicators (KPI) to help the users comprehend and monitor multiple information at a glance, which can support the user to accomplish large workflow tasks ([Elias and Bezerianos, 2011]). Dashboards are designed in collaboration with domain experts (e.g., data analysists in the energy sector) and tailored to fit the expert's needs. Dashboards can visualize multiple views in a single window, which provide a comprehensive overview of the data being used to perform a specific analysis. The design of the dashboard is subject to the data and the workflow. The number of views to be displayed can vary based on the use case. Too many views can make the dashboard cluttered and hard to comprehend, while the right number of views can increase human cognition and enhance information understanding ([Arbesser et al., 2017]). `VISPLORE` provides task-tailored dashboards (Figure 3.3) that contain a collection of multiple visualization views, such as table lenses and scatter plots, to help users solve a given task.

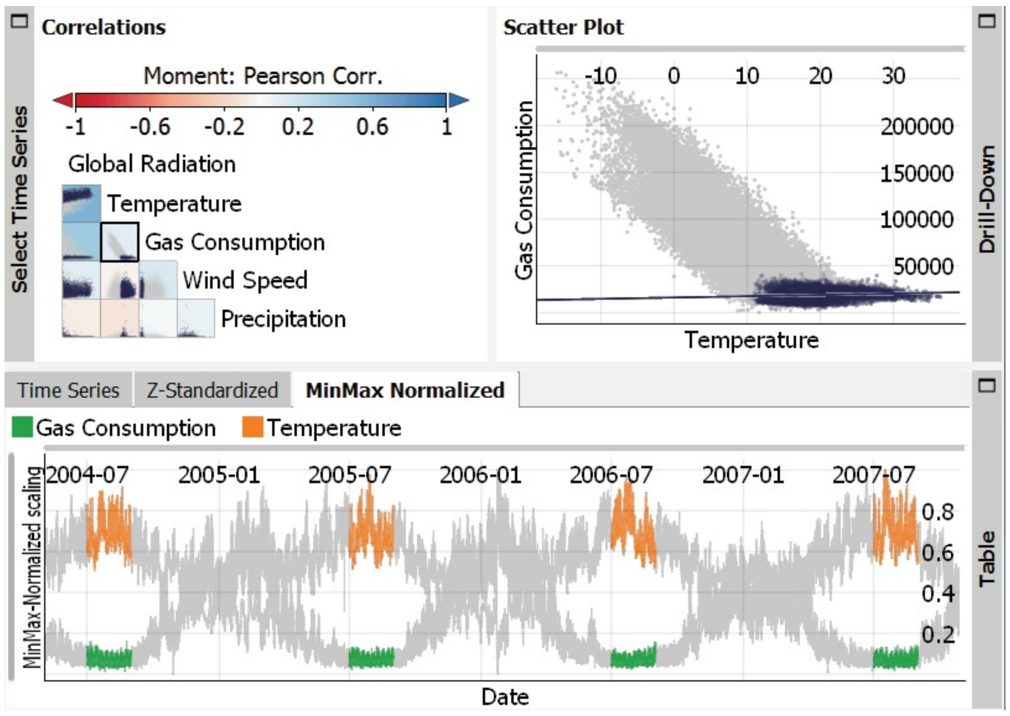


Figure 3.3: Dashboard in VISPLORE combining multiple views: correlation, scatter plot, and time series view.

CHAPTER 4

Work Tasks and Problem Description

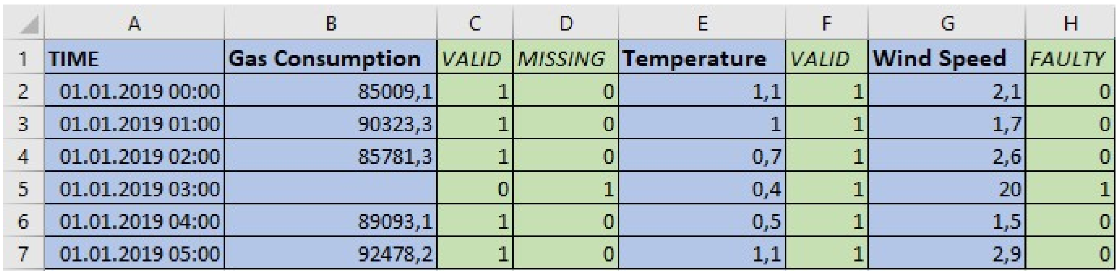
DQ metrics are represented as meta-information attached to the source data when loaded into `VISPLORE`, e.g., extra columns beside each data column when working with `Excel` spreadsheets as in Figure 4.110. Our focus will be on the following types of DQ metrics in `VISPLORE`:

- *Imputed*: indicates that an entry holds a value that is set by some means of imputation.
- *Valid*: indicates that an entry holds a valid value.
- *Missing*: indicates that an entry has no value.
- *Accounted*: indicates that an entry holds a value that was calculated by other values.
- *Faulty*: indicates that an entry has a wrong value.
- *Manually replaced*: indicates that an entry has a value that was set manually by a user.

This work addresses two research topics: *(i)* expanding the framework `VISPLORE` to support the visualization of above mentioned DQ metrics in the table lens view and *(ii)* integrating selected techniques in `VISPLORE` for data cleaning that enables the user to mark data entries as implausible and to correct data errors interactively, while keeping track of this cleaning process.

4.1 Data Quality Visualization

When importing a dataset into VISPLORE, all *meta-information* (e.g., *valid* or *missing*) attached to the imported dataset will be loaded into the system as well. Data columns (referred to as *Channels* in VISPLORE) can have multiple binary masks associated to them, where each mask represents a meta-information type. For instance, if a data channel has the binary mask *faulty* assigned to it, then each bit field in the mask will indicate whether its corresponding data channel entry is marked as faulty or not (a TRUE/1 value indicates the entry is marked as *faulty*, FALSE otherwise), as shown in Figure 4.1. Our focus will be on the following meta-information types: *imputed, valid, missing, accounted, faulty and manually replaced.*



	A	B	C	D	E	F	G	H
1	TIME	Gas Consumption	*VALID*	*MISSING*	Temperature	*VALID*	Wind Speed	*FAULTY*
2	01.01.2019 00:00	85009,1	1	0	1,1	1	2,1	0
3	01.01.2019 01:00	90323,3	1	0	1	1	1,7	0
4	01.01.2019 02:00	85781,3	1	0	0,7	1	2,6	0
5	01.01.2019 03:00		0	1	0,4	1	20	1
6	01.01.2019 04:00	89093,1	1	0	0,5	1	1,5	0
7	01.01.2019 05:00	92478,2	1	0	1,1	1	2,9	0

Figure 4.1: A snapshot from an Excel datasheet showing different data columns (highlighted in blue), and alongside each column the meta-information masks associated to it (in green).

In VISPLORE, *valid* and *missing* are regarded as special meta-information types. Whenever a new data channel is loaded into the system, a binary channel named *validity* is created and linked to the newly added data channel. The *validity channel* reflects the content of the data channel, whether a data entry has a value or is empty (null). Using the *validity channel*, the two meta-information types *valid* and *missing* can be easily deduced using the binary operation *bitwise NOT* , i.e., swapping 0s for 1s and vice versa. In other words, the name *valid* is used to describe the meta-information binary channel in the source data as in Figure 4.1, while the name *validity* refers to the binary channel created by VISPLORE when loading a data channel into the table lens.

Prior to our work, the *table lens* only supported the visualization of two types of *per-value* meta- information: *valid* and *missing*. Since those two meta-information types are mutually exclusive (i.e., a data entry can either exist or not), a single binary channel (the *validity channel*) was adequate to represent both valid and missing. Until now, the *validity flag* (mentioned in 3.2.2) was only used to display a single meta-information type, either valid or missing. The behavior of the validity flag was hard-coded into VISPLORE to be *dark gray* for valid data entries and *white* for missing ones.

We started by adding support for the visualization of a new type of meta-information: *imputed*. The *validity flag* is called from now on the *meta-information flag*, because it is now capable of displaying the *imputed* status, which is an additional meta-information type, beside valid and missing. *Imputed* is visualized by coloring the validity flag in *blue*. However, the meta-information flag is not capable yet of displaying multiple per-value meta-information per data cell. If the meta-information type imputed exists for a data cell, the flag will be displayed in blue only.

The next step in our work is to introduce a new approach in terms of per-value meta-information visualization, by dividing the quad reserved for the meta-information flag into multiple colored *trapezoids*, one per each meta-information type as shown in Figure 4.2. We choose to divide the flag into trapezoids and not rectangles because they have the advantage of better visibility in a flag with too many meta-information types, since all the lines dividing the flag will have a consistent *slope* regardless of number of meta-information per flag. This way, per data cell, the visualization of multiple per-value meta-information can be combined into a single flag.

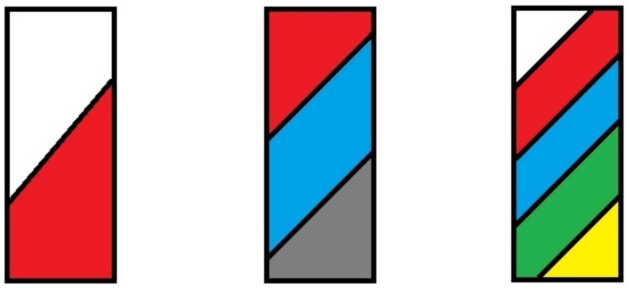


Figure 4.2: Different cases for meta-information flags divided into multiple trapezoids to allow the visualization of many per-value meta-information types, each type is allocated a unique color. In case of a single meta-information per cell, the flag is a single colored quad.

The *CategoricalColoring* class serves as a catalog providing unique coloring for each meta-information type when rendering the table lens. When the table lens is created, the *CategoricalColoring* class will be initialized with the two basic meta-information types *valid* and *missing*, and as new data channels are mapped (or unmapped) to the view, the *CategoricalColoring* class will be updated with all the *per-value meta-information* types contained in that channel. This will result in the *CategoricalColoring* class being up-to-date with all the meta-information types in the view. The colors are assigned randomly to each meta-information type, with *valid* and *missing* being hard-coded into `VISPLORE` to remain *dark gray* and *white* respectively as before. With multiple meta-information types being visualized now, a color legend is added at the top of the table lens to indicate the color for each meta-information type in the view. To keep the UI as simple as possible, the color legend will be hidden if only the meta-information *valid* and *missing* are present. The user also has the option to disable the visualization of per-value meta-information types altogether with a simple toggle, which will revert the visualization of the meta-information flag back to its initial state , i.e., white quads for missing values and dark gray quads for valid ones and ignoring any other type of meta-information.

4.2 Data Cleaning

In this work, all the data cleaning techniques added to `VISPLORE` allow the user to perform manual data cleaning, rather than automated and automatic cleaning. Any cleaning performed on the data will only modify the internal copy of the imported data (i.e., cached data) and will not be committed to the original data source (e.g., `Excel` file) unless the user chooses to export the data to an external source. The first method for data cleaning allows the user to select multiple data cells in the table lens and then perform the action *set as missing*. This will cause the values for the selected data entries in the view to be replaced by the value *missing* and the corresponding binary entries in the *validity channel* for each data entry will be set to *false*, i.e., the meta-information type *missing* is attached internally to those data entries and not to the original data source. To keep track of whether a data entry is originally missing when imported into `VISPLORE` or is marked as missing by a user later, we differentiate between two types of missing meta-information. *Explicit missing* denotes entries marked as missing by users through the *set as missing* action, while *implicit missing* is for those entries that are already missing when imported (nulls). The second method for data cleaning allows the user to *manually set* the value of a data cell in the table lens. A popup box will prompt the user for input to replace the value of the selected data cell(s). This cleaning method is an experimental feature and is partially supported, so the meta-information type *manually replaced* is not attached to the modified data entries. Only the editing of numerical and time entries is supported as string values (text and categorical entries) are translated to categorical types when imported into `VISPLORE`.

Replacing data values with nulls (or MISSINGs) can be considered a form of data cleaning. The basic idea is to detect anomalies and outliers in the data and then to replace them with empty values. In a later stage, suitable techniques for filling up missing entries will be applied for cleaning such as replacing empty values with defaults or using regression analysis to estimate the missing values. How to clean and replace missing values however is beyond the scope of this work. The data processing and visualization mentioned above are limited to table lens, with the main emphasis on time-series data and numerical values. This type of cleaning falls well within the category of quantitative cleaning mentioned earlier. Visualizing all the per-value meta-information can enhance the readability of the table lens, which allows the user to get a comprehensive understanding of the data resulting in better decisions regarding data analysis and cleaning.

CHAPTER 5

Discussion

In this section we will *(i)* discuss the implementation details, *(ii)* evaluate the work that was done, and *(iii)* discuss the challenges during the implementation of the added features.

5.1 Implementation

Most of the implementation work is done on the backend side, so not a lot of changes are visible to the user. A large part of the work is to create an efficient cache, which lives in the table lens's backend, to be used by the frontend when rendering the view. The cache stores the meta-information for all the data entries in the current view, i.e., all the data cells in the table lens view. In many cases we are dealing with large datasets (e.g., 100k records in some instances), where each data entry can have up to six types of meta-information attached to it. This requires a dynamic data structure to store all this information, while keeping the footprint of the cache minimal.

For each data channel loaded into the view, we compute and cache the number of meta-information types that exist in a data channel (i.e., the uniquely identified types). By caching this number, we can skip the rendering of the trapezoids (described in 4.1) if the number of meta-information types in a channel is zero. We also store the occurrences of all meta-information types in a data channel as a list of vectors, one for each meta-information type in the channel, similar to the boolean vector for validity in section 4.1. For each data channel, a vector caches the number of meta-information types that exist per data entry in a channel. As the cache needs to remain up-to-date, any changes made to the table lens view or other views will invalidate the cache, which will notify the backend to rebuild the cache again.

5.2 Evaluation

With the above design of the table lens view's cache, all the data required to visualize the meta-information are cached in the backend of the view. Using the cache, many calculations required for rendering the view can be skipped, which allows the frontend to render faster. Since all the caching is done in the backend of a view, which runs in a separate thread, the overhead of maintaining the cache is reduce and the rendering will not be affected every time the cache is invalidated. Since we are mostly storing binary data in the cache, memory usage is minimal even if we are working with large datasets.

When testing the performance and memory consumption of the table lens view with large datasets, while both enabling and disabling the visualization of the extra meta-information types in the flags, the responsiveness of the view (and `VISPLORE` as well) remains similar to the state of the application before implementing the new features. The animation while scrolling through the data and zooming in and out of the view remains smooth, and the response time remains well beyond the 100 milliseconds threshold. Although the response time did increase a few milliseconds due to the additional information inside the flags that need to be rendered, the overall performance and responsiveness of the table lens view remains fast and well perceived by the user.

5.3 Challenges

The challenge in this work was to create an efficient cache, which stores all the meta-information for all the loaded data, to speed up the rendering of the view, while keeping the cache up-to-date whenever a change is made to the data or the view. Any modification to the table lens view such as setting an entry to missing, adding a new data channel to the view, or removing a channel will invalidate the cache and trigger an event, which will cause the cache to be updated again.

Another challenge we had to deal with was the distinction between entries that are originally missing when importing the data, and entries that are set as missing by the user during the analysis of the data. While using *explicit missing* to refer to the former case and *implicit missing* for the latter solves the issue, it causes some inconveniences since many features related to missing data are hard-coded into `VISPLORE`.

Choosing the proper layout to divide the new flag was also challenging, as it is important to maintain a high readability for the flags in the table lens view, especially in the case of too many data entries and multiple meta-information types per entry (i.e., three and above). We choose to divide the flags into trapezoids as they provide consistency across flags with multiple meta-information.

The last challenge was that the visibility of the table lens view was affected by the visualization of the new meta-information types, as too much information were being visualized to the user, especially within dashboards. Hence we choose to hide the visualization of the meta-information by default. If the user needs the additional information, he/she can choose to toggle the visualization of the extra information on.

CHAPTER 6

Conclusion and Outlook

In this work we reviewed the criteria for clean data and how clean data is a critical requirement in any information system to obtain reliable analysis results. For example, in a system for data classification, the classification's accuracy is increased by 100% via cleaning the data first. We also addressed the most common data problems that affect the quality of data and how to detect these problems, such as missing data entries, duplicates and other data issues. We outlined the types of data cleaning: qualitative and quantitative cleaning, and the techniques used to fix data problems such as regression analysis for filling up missing entries.

The practical part of this work, realized in `VISPLORE`, focused on integrating selected techniques into the framework to enable the users to correct certain data errors interactively in the table lens, while keeping track of the cleaning process. Besides, the work allowed the visualization of per-value meta-information types to present more useful information in the table lens. This feature helps the user to understand the status of data entries and thus to decide what entries need to be cleaned and how. Further features are considered as part of future work on `VISPLORE`. The first is to allow the user to perform *in-place editing* of data values in the table lens. Another one would allow the user to enable/disable the visualization of certain meta-information types individually and to manually pick the coloring for each meta-information type. An additional feature is to partly support the automatization of tagging implausible values with meta-information without the user's intervention, and to suggest plausible values for manual cleaning when automatic cleaning fails.

List of Figures

Bibliography

[Arbesser et al., 2017] Arbesser, C., Mühlbacher, T., Komornyik, S., and Piringer, H. (2017). Visual analytics for domain experts: Challenges and lessons learned. Proceedings of the second international symposium on Virtual Reality & Visual Computing.

[Batini and Scannapieco, 2006] Batini, C. and Scannapieco, M. (2006). *Data Quality: Concepts, Methodologies and Techniques.* Springer Berlin Heidelberg New York, ISBN-13 978-3-540-33172-8.

[Bohannon et al., 2007] Bohannon, P., Fan, W., Geerts, F., Jia, X., and Kementsietsidis, A. (2007). Conditional functional dependencies for data cleaning. IEEE 23rd International Conference on Data Engineering.

[Chen and Jiang, 2014] Chen, F. and Jiang, L. (2014). A parallel algorithm for data cleansing in incomplete information systems using mapreduce. 10th International Conference on Computational Intelligence and Security.

[Chu and Ilyas, 2016] Chu, X. and Ilyas, I. (2016). Qualitative data cleaning. Proceedings of the VLDB Endowment, Vol. 9, No. 13.

[Chu et al., 2016] Chu, X., Ilyas, I., Krishnan, S., and Wang, J. (2016). Data cleaning: Overview and emerging challenges. pages 2201–2206. SIGMOD '16 Proceedings of the 2016 International Conference on Management of Data.

[Chu et al., 2013] Chu, X., Ilyas, I., and Papotti, P. (2013). Discovering denial constraints. Proceedings of the VLDB Endowment, Vol. 6, No. 13.

[Dara and Satyanarayana, 2015] Dara, R. and Satyanarayana, C. (2015). A robust approach for data cleaning used by decision tree induction method in the enterprise data warehouse. *International Journal on Computational Science & Applications (IJCSA) Vol.5, No.4.*

[Dasu and Johnson, 2003] Dasu, T. and Johnson, T. (2003). *Exploratory Data Mining and Data Cleaning.* John Wiley & Sons, Inc. New York, ISBN:0471268518.

[Elias and Bezerianos, 2011] Elias, M. and Bezerianos, A. (2011). *Exploration Views: Understanding Dashboard Creation and Customization for Visualization Novices.* Springer Berlin Heidelberg.

[Ganapathi and Chen, 2016] Ganapathi, A. and Chen, Y. (2016). Data quality: Experiences and lessons from operationalizing big data. IEEE International Conference on Big Data (Big Data).

[Gokhale et al., 2014] Gokhale, C., Das, S., Doan, A., Naughton, J. F., Rampalli, N., Shavlik, J., and Zhu, X. (2014). Corleone: Hands-off crowdsourcing for entity matching. Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data.

[Hamad and Jihad, 2011] Hamad, M. and Jihad, A. (2011). An enhanced technique to clean data in the data warehouse. Developments in E-systems Engineering.

[He et al., 2016] He, J., Veltri, E., Santoro, D., Li, G., Mecca, G., Papotti, P., and Tang, N. (2016). Interactive and deterministic data cleaning. pages 893–907. SIGMOD '16.

[Hellerstein, 2008] Hellerstein, J. M. (2008). Quantitative data cleaning for large databases.

[Huisman, 2000] Huisman, M. (2000). *Imputation of Missing Item Responses: Some Simple Techniques.* Springer Verlag, Volume 34, Issue 4.

[IEA, 2017] IEA (2017). Digitization and energy. paris: International energy agency, oecd. Last accessed: 09/08/2019.

[Kern and Zinck, 2016] Kern, S. and Zinck, R. (2016). A simple approach to anomaly detection in periodic big data streams. munich: Bmw car it gmbh. Last accessed: 04/07/2019.

[Krishnan et al., 2016] Krishnan, S., Wang, J., Wu, E., Franklin, M. J., and Goldberg, K. (2016). Activeclean: Interactive data cleaning while learning convex loss models. pages 948–959. Proceedings of the VLDB Endowment Volume 9 Issue 12.

[Lee, 2007] Lee, S. (2007). Challenges and opportunities in information quality. The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007).

[Musil et al., 2002] Musil, C. M., Warner, C. B., Yobas, P. K., and Jones, S. L. (2002). A comparison of imputation techniques for handling missing data. *Western Journal of Nursing Research, vol. 24, Issue 7*, pages 815–829.

[Pfahler, 2015] Pfahler, D. (2015). In-place interaction in dashboards. Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Media Informatics and Visual Computing. Vienna University of Technology.

[Piringer et al., 2009a] Piringer, H., Buchetics, M., Hauser, H., and Gröller, E. (2009a). Hierarchical difference scatterplots: interactive visual analysis of data cubes. *SIGKDD Explorations*, pages 49–58.

[Piringer et al., 2009b] Piringer, H., Tominski, C., Muigg, P., and Berger, W. (2009b). A multi-threading architecture to support interactive visual exploration. IEEE Transactions on Visualization and Computer Graphics, VOL. 15, NO. 6,.

[Rahm and Do, 2000] Rahm, E. and Do, H. H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin.*

[Satyendra, 2016] Satyendra (2016). Dgitization process. Last accessed: 04/07/2019.

[Swapna et al., 2016] Swapna, S., Niranjan, P., Srinivas, B., and Swapna, R. (2016). Data cleaning for data quality. pages 344–348. 3rd International Conference on Computing for Sustainable Global Development (INDIACom).

[Wang et al., 2012] Wang, J., Kraska, T., Franklin, M. J., and Feng, J. (2012). Crowder: Crowdsourcing entity resolution. pages 1483–1494. Proceedings of the VLDB Endowment, Volume 5 Issue 11.

[Wang et al., 2014] Wang, J., Krishnan, S., Franklin, M. J., Goldberg, K., Milo, T., and Kraskay, T. (2014). A sample-and-clean framework for fast and accurate query processing on dirty data. pages 469–480. Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data.

[Wang and Strong., 1996] Wang, R. and Strong., D. (1996). Beyond accuracy: what data quality means to data consumers. *Journal of Management Information Systems, vol. 12, Issue 4*, pages 5–33.