FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Grooviler: Ein Visual Analytics Ansatz zur Bereinigung von Qualitätsproblemen von Zeit-Orientierten Daten

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medieninformatik

eingereicht von

Oliver Erhart, BSc
Matrikelnummer 0825648

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Univ.Prof.^{in} Mag.^{a} Dr.^{in} Silvia Miksch
Mitwirkung: Univ.Ass.^{in} Mag.^{a} Dipl.Ing.^{in} Dr.^{in} Theresia Gschwandtner

Wien, 24. August 2017

Oliver Erhart	Silvia Miksch

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Grooviler: A Visual Analytics Approach to Communicate and Identify Time-Oriented Data Quality Problems

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Media Informatics

by

Oliver Erhart, BSc
Registration Number 0825648

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof.^{in} Mag.^{a} Dr.^{in} Silvia Miksch
Assistance: Univ.Ass.^{in} Mag.^{a} Dipl.Ing.^{in} Dr.^{in} Theresia Gschwandtner

Vienna, 24th August, 2017

Oliver Erhart	Silvia Miksch

Erklärung zur Verfassung der Arbeit

Oliver Erhart, BSc
Trillergasse 2b/104, 1210 Vienna

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, 24. August 2017

Oliver Erhart

Danksagung

Ich bedanke mich bei meiner Lebensgefährtin, die ich während dieser Arbeit geheiratet habe. Sie hat mich stetig motiviert diese Arbeit fertig zu stellen, und half mir, mich aus verstrickten Überlegungen zu befreien. Ein besonderer Dank gilt meinem Sohn, der im Zeitraum dieser Arbeit auf die Welt gekommen ist. Er hat mir während dem Schreiben dieser Arbeit mit seinem Lachen viel Freude bereitet, und dadurch indirekt darauf gedrängt mein Studium abzuschließen.

Theresia Gschwandtner möchte ich ebenso danken, da sie mich mit konstruktiver Kritik bei dieser Arbeit begleitet hat. Letzendlich bedanke ich mich bei meinen Eltern, die mich immer motiviert haben und mir gezeigt haben dass man alles schaffen kann wenn man hart genug dafür arbeitet.

Acknowledgements

I want to say thanks to my partner I married during this work. She constantly motivated me to finish this paper and helped me finding out of consideration impasses. A special ‘thank you’ to my son who was born during this work. While writing on this thesis, his laughing pushed me to finally finish my studies.

I also wish to thank Theresia Gschwandtner who accompanied me with a lot of constructive criticism. Finally i want to thank my parents for always motivating me and showing me that everything is possible if you work hard enough.

Kurzfassung

In wissenschaftlichen Disziplinen, wie Klimaforschung oder Elementarteilchenphysik enstehen oft große Datenmengen von Zeitreihen. Um diese Daten für explorative Analyse oder Data Mining verwenden zu können, muss die Datenqualität und -integrität gegeben sein. Mithilfe von Data Cleansing können unerwünschte und irreführende Datenanomalien bereinigt werden. Data Profiling hilft dabei Datenqualitätsprobleme zu entdecken und darzustellen.

Bestehende Data Profiling Tools bieten Mechanismen zur Identifizierung von Datenqualitätsproblemen an, unterstützen jedoch keine zeit-orientierten Datenqualitätsprobleme. Meist liefern diese Werkzeuge nur textuelle Statistiken über das Datenset und bieten keine Interaktionsmöglichkeiten an.

Auf der technischen Universität Wien wird bereits ein Prototyp entwickelt der fehlerhafte Datensätze erkennt, teilweise bereinigt oder kennzeichnet. Diese Arbeit handelt vom Design, Implementierung und Evaluierung eines prototypischen Moduls, bereits festgestellte Fehler darzustellen. Zusätzlich sollen mithilfe von interaktiven Visualisierungen weitere Datenqualitätsprobleme entdeckt werden können.

Abstract

Scientific disciplines like climate research or high-energy physics build up large repositories of time series data. In order to process big data sets for data mining or explorative analysis, data integrity and quality has to be guaranteed. To increase data quality, Data Cleansing is performed to remove unwanted and misleading data anomalies before executing the analytical process. Data Profiling also helps to identify and communicate data quality problems, which can then be cleansed.

Existing Data Profiling tools provide mechanisms to identify invalid data but without prior focus on time-oriented data. Often those tools are limited providing simple statistics about a given data set and do not offer interactive operations.

There is already a prototype being developed at the Technical University of Vienna which detects, partially cleanses, and annotates erroneous data entries. This thesis is about designing, implementing and evaluating a prototypical module of this prototype, visualizing detected errors and providing visualizations to find further data problems with a focus on time-oriented quality checks.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 General Introduction

Scientific disciplines like climate research or high-energy physics build up large repositories of time series data [3]. In order to process big data sets for Data Mining or explorative analysis, data integrity and quality has to be guaranteed. Data Mining is defined as "the application of specific algorithms for extracting patterns from data." [4, p.39] Inaccurate data increases with complexity of systems and it requires aggressive actions to correct problems which lead to data quality problems [5]. Therefore analysts need to identify data quality problems in their data sets before they can be used for data analysis.

Data sets often contain so called dirty data which can be for instance invalid, erroneous, missing values, outliers, or duplicate records [6]. To increase data quality, Data Cleansing is performed to remove unwanted and misleading data anomalies before executing the analytical process. Data Cleansing is a crucial task for data analysis which can take about 80% of a project's effort [7]. Therefore interactive Data Cleansing tools have the potential to reduce a project's cost and time.

Data Wrangling is defined as "a process of iterative data exploration and transformation that enables analysis" [8, p.272] to make data useful. It consists of (1) Data Profiling, which is getting an overview about the current state of data and containing errors [9], (2) Data Cleansing, which is the semi-automatic correction of data problems, and (3) Data Transformation, to get data into desired state. Data Wrangling tools can also be called Interactive Data Transformation tools (IDTs) [9]. The example of a Data Wrangling process is shown in Figure 1.1.

Data Profiling tools help to get an understanding of the content, structure and quality of data. With statistical methods and value distribution visualizations, this tools can help to identify data quality problems [5]. Data Profiling can be done with statistics or automated checks (e.g., giving an overview about missing/invalid data), but also with

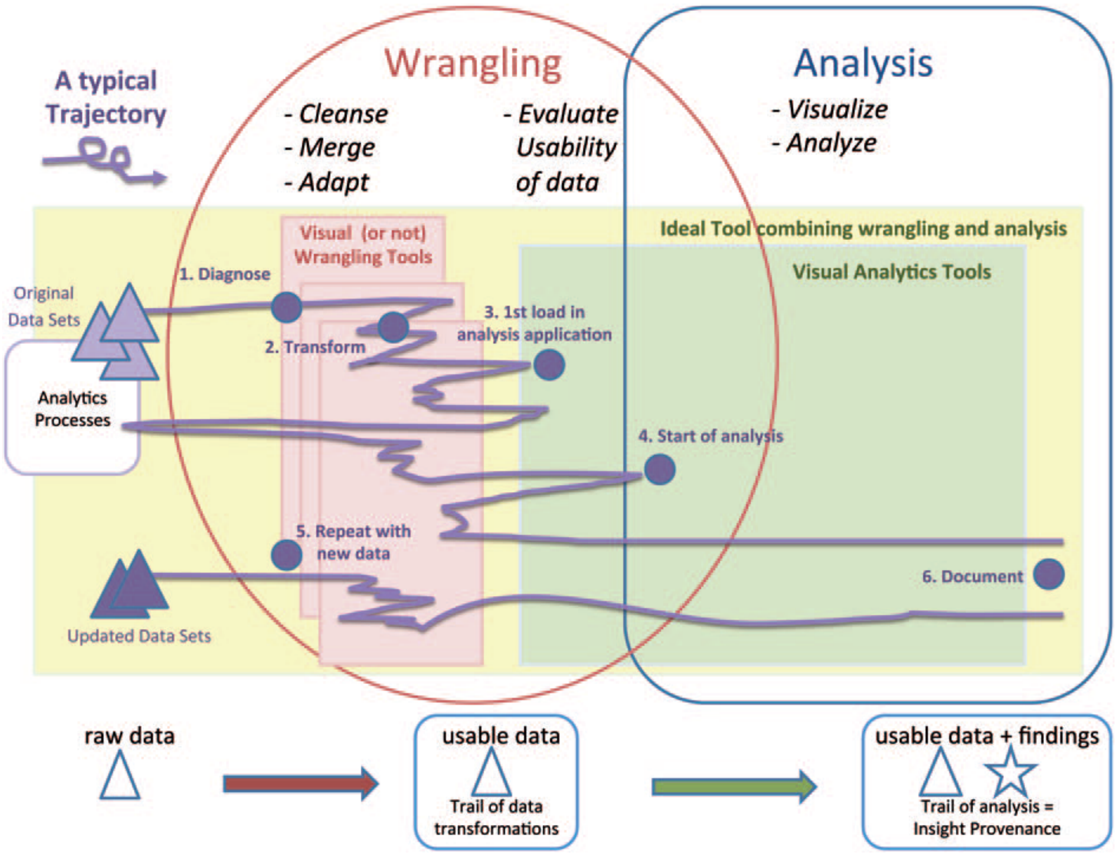


Figure 1.1: Iterative wrangling and analysis process. "The wrangling and analysis phases overlap. While wrangling tools tend to be separated from the visual analysis tools, the ideal system would provide integrated tools (light yellow). The purple line illustrates a typical iterative process with multiple back and forth steps. Much wrangling may need to take place before the data can be loaded within visualization and analysis tools, which typically immediately reveals new problems with the data." [7, p.273]

visualizations addressing specific problems like outliers. Human judgment is needed because data anomalies may indicate data quality problems, which have to be verified by analysts whether they are in fact errors [8].

Errors can then be cleaned, either automatically, semi-automatically or manually. Data Cleansing tools help to find erroneous data by means of defined rules [8]. An error can either be fixed with substitution and correlation or can be entirely rejected [5].

Often data has to be converted or redefined to get standardized values with specific patterns or types. Sometimes data has to be merged from different sources (e.g., a data set about employees is merged with data from subcompanies), which induces possible multi-source data problems, like wrong references or naming conflicts [8].

Each transformation can have an impact on data quality, so Data Profiling has to be performed after transformation. Therefore Data Wrangling is an iterative process, before data can finally be considered useful.

Data quality problems can be divided into schema level and instance level [6]. While schema level data problems can be solved by improving the schema design of, for instance, a database (e.g., not allowing null fields or specifying patterns for Strings), they can't fully avoid problems with the instance level, i.e., data contents. Users may provide dummy values for fields, which must be set, to satisfy the schema.

The results of classifying these data quality problems of different levels are data quality taxonomies. They help to identify different types of errors which can then be used by Data Cleansing tools. Most data quality taxonomies do not focus on time-oriented data [2, 10, 11] except for the taxonomy by Gschwandtner et al. [12] which includes error types like different or incorrect time zones or erroneous time intervals $(End \leq Start; duration \geq 0)$.

1.1.1 Information Visualization and Visual Analytics

Information Visualization (InfoVis) is defined as "the use of computer-supported interactive visual representations of abstract data to amplify cognition." [13, p.5] Since Data Profiling is about gaining insights about a data set, interactive visualizations are suitable for these kind of tools.

The combination of interactive visualization and automated analysis is called *Visual Analytics* (VA) [14]. VA solutions combine the strengths of computers and human respectively. While machines take over the calculation and data processing, humans provide cognition, visual intelligence, and decision making. Users benefit from "an effective understanding, reasoning and decision making on the basis of very large and complex data sets." [14, p.157] VA methods are suited for gaining an understanding of data and its quality, supporting users to "detect the expected and discover the unexpected." [14, p.157]

Existing Data Cleansing tools [3, 9, 8, 15] provide mechanisms to cleanse invalid data but without prior focus on time-oriented data sets. There is already a prototype being developed at the Technical University of Vienna which detects and annotates erroneous data entries. This thesis is about designing, implementing and evaluating a VA prototypical module of this prototype, visualizing detected errors and providing visualizations to find further data problems with a focus on complex time-oriented quality checks.

1.2 Research Questions

The following research questions are formulated to be investigated in this thesis.

Main Question:

- **RQ1**: How can VA methods help to identify and understand data quality problems of complex time-oriented data?

Sub Research Question:

- **RQ1.1**: How can already detected and annotated quality problems be visualized in an effective and adequate way?
- **RQ1.2**: What visualizations can help to find additional quality problems of time-oriented data?

1.3 Methodological Approach

The methodological approach consists of the following steps:

1. *Literature Review:*
   In a first step, a literature research was performed to gather theoretical knowledge about Data Cleansing, Profiling, and Wrangling.

2. *State of the Art Report:*
   Based on the findings of the literature review a detailed state-of-the-art report was written, outlining the currently existing tools and possibilities in Data Profiling.

3. *Conceptual Design of a VA Prototype:*
   The next step was a conceptual design phase of a VA prototype based on the results of the state-of-the-art report. We conducted an iterative design with regular feedback cycles. In our design phase we applied the Data-Users-Tasks design triangle [1] and focused on Munzner's nested model for visualization design [16]. We chose a suitable data quality taxonomy, on which we defined different error types, to be visually investigated. In addition, a suitable graphical user interface (GUI) has been designed.

4. *Implementation:*
   The prototype consisting of different visualizations was implemented in JAVA. We used a slightly modified version of the Prefuse Toolkit [17] which is continuously extended by the Information Engineering Group at the Vienna University of Technology.

5. *Qualitative Evaluation:*
   We evaluated our prototype with the help of a qualitative evaluation with a special focus on revealing insights. This provided the basis to answer our research questions and gave us feedback for reflection.

6. *Critical Reflection and Discussion:*
   We then reflect and discuss the results of the qualitative evaluation. Finally, we provide a conclusion which summarizes the scope of this work and state future work in this research area.

1.4 Structure of the work

This paper is structured as followed:

- **Chapter 2, Related Work:** In this chapter we cover existing open source and commercial Data Profiling products. We provide a comparison between their features and techniques. Additionally, we show a selection of visualization techniques for time-oriented data.
- **Chapter 3, Design:** This chapter covers the design process ranging from identifying the requirements, brainstorming up to refining our idea. This also results in a GUI design which will then be implemented.
- **Chapter 4, Implementation:** This section covers features and functionality of the prototype and gives insight about the technical implementation.
- **Chapter 5, Evaluation:** This section covers the technique, results and discussion of our qualitative evaluation with a special focus on revealing insights.
- **Chapter 6, Critical Reflection & Discussion:** Finally we conclude the work by summarizing findings in the previous chapters. Also we will provide future work within this research area.

CHAPTER 2

Related Work

2.1 Literature research

In this chapter, we will investigate existing data quality tools and visualizations for time-oriented data. The following results were discovered through my research, Google and Google Scholar search. Digital, scientific libraries (ACM [18], IEEE Xplore [19], and SpringerLink [20]) were also used, but they did not provide additional results than the ones already found with Google Scholar. The keywords in Table 2.1 were used for the previous mentioned search engines. We additionally investigated promising time-oriented visualization types with the help of TimeViz browser [21]. Furthermore, we scanned the reference lists of papers to discover further related work.

2.2 Data Quality Taxonomies

Whereas data quality is a broad term, this paper specifically focuses on data quality problems for time-oriented data. An example is a data set of water quality measurements

used keywords		
data quality	data wrangling	data cleansing
data profiling	data cleansing prototype	brushing and linking
google refine	data mining	data taxonomy
time-oriented data	time-dependent data	information visualization
visual analytics	statistics	dirty data

Table 2.1: Used keywords for literature research.

Data Quality Problem	**Example**
Illegal values	2010-13-32
Violated attribute dependencies	birth date 50 years ago, and age of 10 years
Uniqueness violation	two records with the same id
Referential integrity violation	reference to a missing data record

Table 2.2: Schema-related data quality problems defined by Rahm et al. [2]

Data Quality Problem	**Example**
Missing values	dummy values or *null*
Misspellings	"Viena" instead of "Vienna"
Cryptic values and abbreviations	"DBSTF" or "E"
Embedded values	"John Smith, New York"
Misfielded values	phone number in address field
Violated attribute dependencies	wrong zip code for city
Word transpositions	one user provides his name as "John S." and another one as "K. Miller"
Duplicated records	"John Smith" and "J. Smith"
Contradicting records	same records with different values
Wrong references	"John Smith" is referenced with department number 17, which is wrong

Table 2.3: Instance-related data quality problems defined by Rahm et al. [2]

from different stations. Beside quality checks like completeness or outliers, time-oriented data needs different data quality checks. To stay with the example of water quality measurements: Did every station measure every five minutes? Did a station stop working for a specific time period? Or, were there at least fifty measurements per hour?

Data quality can be defined in many different ways for different target groups. As an example, Olson [5] defines data quality as the satisfaction of data for its intended use due to accuracy, timeliness, relevance, completeness, understanding, and trust. Data quality can also be defined on the basis of data quality problems, which depend on the specific use case.

Rahm et al. [2] provide definitions about data quality problems, which are divided into schema-related and instance-related data quality problems. Schema-related problems (see Table 2.2) can be addressed by a better schema design (e.g., by defining constraints in a database). Instance-related problems (see Table 2.3) have no relation to the behind schema, and are actual errors in data.

When data sets are merged together by multiple sources, additional data quality problems arise: Referential Integrity constraints, wrong references, naming conflicts, heterogeneity of syntaxes and semantics, and different encoding formats.

Gschwandtner et al. [12] compared general data quality taxonomies for a definition of a data quality taxonomy for time-oriented dirty data. Their comparison points out additional single-source problems like unexpected low/high values, outdated temporal data, dummy entries, and inconsistent spatial data. They also defined additional data quality problems for time-oriented data, like wrong data formats, wrong time zones, implausible time ranges, and unexpected short/long gaps or intervals.

2.3 Data Quality Tools

This section gives an overview about the current state of the art in data quality tools, which can either be Data Profiling, Data Cleansing or Data Wrangling tools. The resulted data quality tools are described due to own testing, or by its documentation if it has not been accessible to test. In addition, each tool has a short summary about time-oriented data quality support.

A solution was selected when it supported either Data Profiling, Data Wrangling or Data Cleansing techniques. Solutions which were not accessible for testing were excluded if they had insufficient documentation about their features. Many results of the conducted scientific research deal with data quality algorithms (e.g., outlier detection, pattern matching) which were excluded in this work, since this paper does not focus on algorithms but on data quality tools.

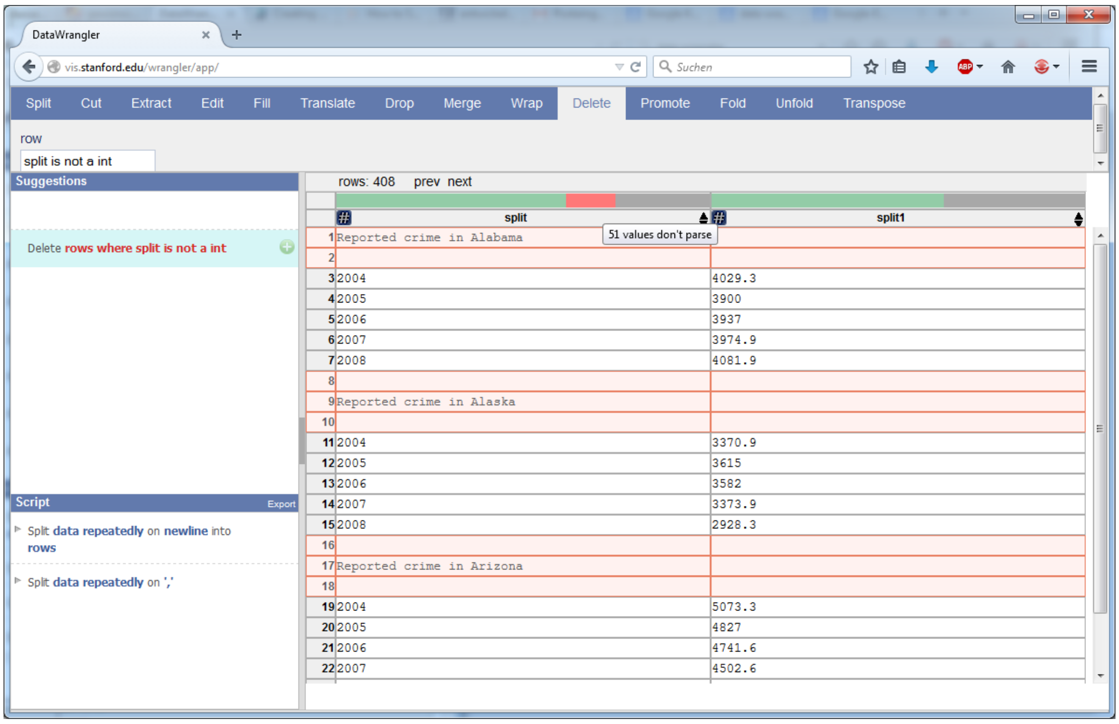
The final results consist of three scientific approaches (Wrangler [8], Profiler [15] and TimeCleanser [22]), two open source solutions (OpenRefine [9] and DataManager [23]), and four commercial products (DataCleaner [24], Data Match 2013 [25], Talend Open Studio [26] and Datamartist [27]).

2.3.1 Wrangler

Description

Wrangler [8] is a free accessible web application [28] developed at Stanford University for Data Wrangling. Due to its web architecture, Wrangler has a limit of 1.000 data records, and operation and calculations act slower compared to desktop applications. After importing a data set, Wrangler attempts to infer a suitable data type for every column. Based on the underlying data type, Wrangler tries to parse data and marks not parseable records as invalid. Every data record has one of the following states: valid, invalid (not parseable), or missing.

Those states are visualized in a data quality meter above every column (see Figure 2.1), where a green marking indicates valid values, red shows invalid values, and gray displays missing values. This data quality meter is linked with the tabular view. By clicking on it, corresponding data records will be selected and transformation suggestions will be shown. Transformations can be, f.i., removing, filling, splitting, and extracting, and are performed with *Natural language description*, a transformation language with interactive parameter selection.



Delete rows where split is not a int

Split data repeatedly on newline into rows

Split data repeatedly on ','

rows: 408 prev next

	split	split1
1	Reported crime in Alabama	
2		
3	2004	4029.3
4	2005	3900
5	2006	3937
6	2007	3974.9
7	2008	4081.9
8		
9	Reported crime in Alaska	
10		
11	2004	3370.9
12	2005	3615
13	2006	3582
14	2007	3373.9
15	2008	2928.3
16		
17	Reported crime in Arizona	
18		
19	2004	5073.3
20	2005	4827
21	2006	4741.6
22	2007	4502.6

Figure 2.1: GUI of Wrangler [8]. Action menu containing operations for the current selection (top). Transformation suggestions for current selection, and transformation history (left). Tabular view of wrangled data set (right). Above each column a data quality meter gives an overview about data quality.

Although the focus of this prototype lies on Data Wrangling, this system shows one approach for dirty data visualization inside a data table.

Time-oriented Data Support

Wrangler supports date types with the format *yyyy-MM-dd*, therefore time domain is not supported. The date is furthermore only checked by syntax, because Wrangler accepts invalid dates like *2000-13-32*.

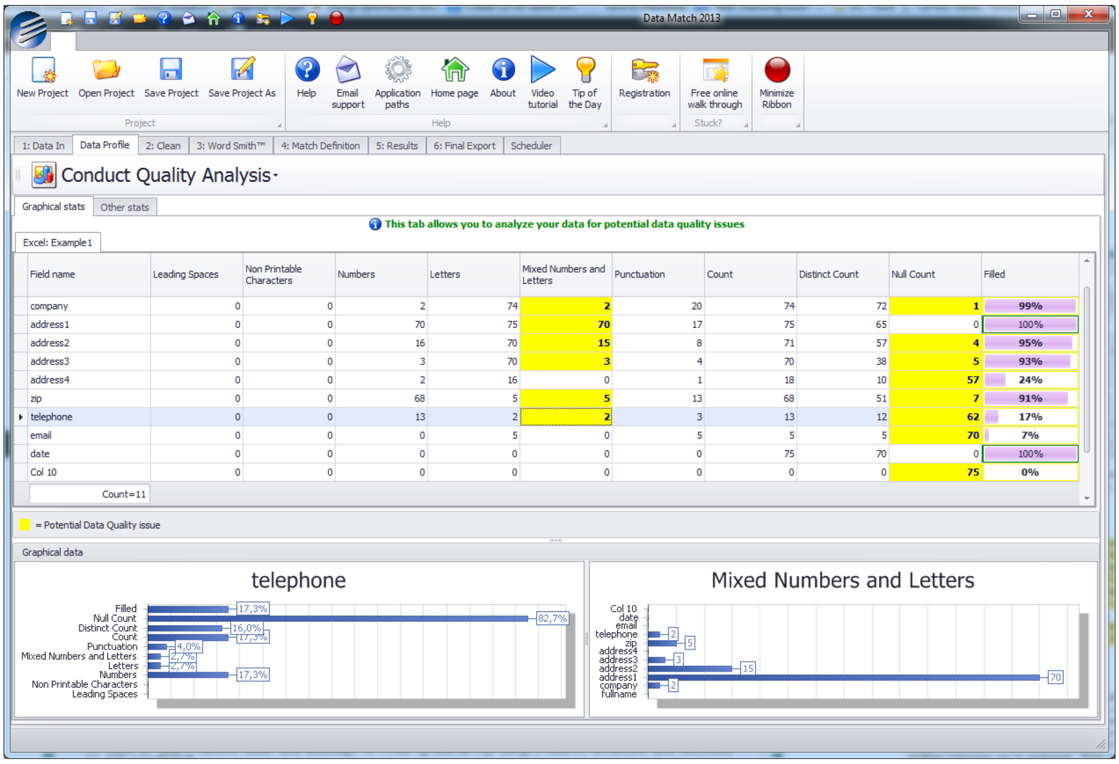
Wrangling a simple example with mixed date formats like *yyyy-MM-dd* and *yyyy.MM.dd* was relatively complex with the built-in *Natural language description*. A simple regular expression would have solved this problem much faster.

Although Wrangler supports date object, no statistics or visualizations are provided for Data Profiling purposes. The system can only display the syntactic validity of data records.

2.3.2 Data Match 2013

Description

Data Match 2013 [25] is a commercial product used for identifying data quality issues in data sets and defining rules which can then be performed to generate a cleaned data set. The focus lies on data deduplication, therefore duplicate data records can be identified with exact or fuzzy search. The matched data can then be merged due to a give rule (e.g., use the result with the highest value).



This tab allows you to analyze your data for potential data quality issues

Field name	Leading Spaces	Non Printable Characters	Numbers	Letters	Mixed Numbers and Letters	Punctuation	Count	Distinct Count	Null Count	Filled
company	0	0	2	74	2	20	74	72	1	99%
address1	0	0	70	75	70	17	75	65	0	100%
address2	0	0	16	70	15	8	71	57	4	95%
address3	0	0	3	70	3	4	70	38	5	93%
address4	0	0	2	16	0	1	18	10	57	24%
zip	0	0	68	5	5	13	68	51	7	91%
telephone	0	0	13	2	2	3	13	12	62	17%
email	0	0	0	5	0	5	5	5	70	7%
date	0	0	0	0	0	0	75	70	0	100%
Col 10	0	0	0	0	0	0	0	0	75	0%
Count=11										

🟨 = Potential Data Quality issue

Figure 2.2: Data Profiling View of Data Match 2013 showing statistics about data columns [25].

To identify possible data quality issues, Data Match 2013 provides a "Data Profile" view (see Figure 2.2), where means of missing and distinct values, and textual statistics are shown.

Displaying the completeness of the data set is informative about the data qualities state. Unfortunately the program provides no options to identify the corresponding data records where values are missing. An option to automatically fill those missing values is also not given, as well as directly manipulating the data set in tabular view.

Data Cleansing is reduced to textual functions like removing letters, numbers, symbols or white spaces. Additional features like spelling change (upper and lower case) as well as reversing text are supported to increase textual data quality.

After cleaning data and removing duplicates, the application can either generate a change report, or export the data set. Due to the trial version restriction it is impossible to tell in which output formats the data set can be exported.

Time-oriented Data Support

Although data records can not be set to a data type “Date”, the program seems to interpret them as Strings with a specific format, because the Data Profile view displays a median date value. However, the built-in matching algorithm processes date types as String, which does not satisfy desired Data Profiling results.

2.3.3 DataManager

Description

DataManager [23] is a free Data Wrangling software with integrated Data Profiling techniques. It uses a graph-based approach, where input-file-nodes can be linked with operation-nodes to generate transformed data sets (see Figure 2.3).

DataManager supports CSV files or database tables linked with database connections as input streams, which are visualized as input nodes. These nodes can then be linked with work nodes to transform the containing data set. Work nodes define operations on the data set, like filtering or sorting specific rows. The following wrangling operations are supported by DataManager:

- **Merge** two input streams into one output stream
- **Append** an input stream to another input stream
- Keep only **distinct** values of a specified attribute
- **Fill** empty fields with a specified rule
- **Filter** values based on an expression
- **Sort** the data set by rows
- **Calculate** values from existing data

The focus of DataManager lies on data transformation techniques, while Data Profiling functions help to identify data quality problems like missing values and value distribution to address those quality problems with suitable transformations. Data Profiling is limited on numerical values and can’t process data types like Strings, Dates or Times.

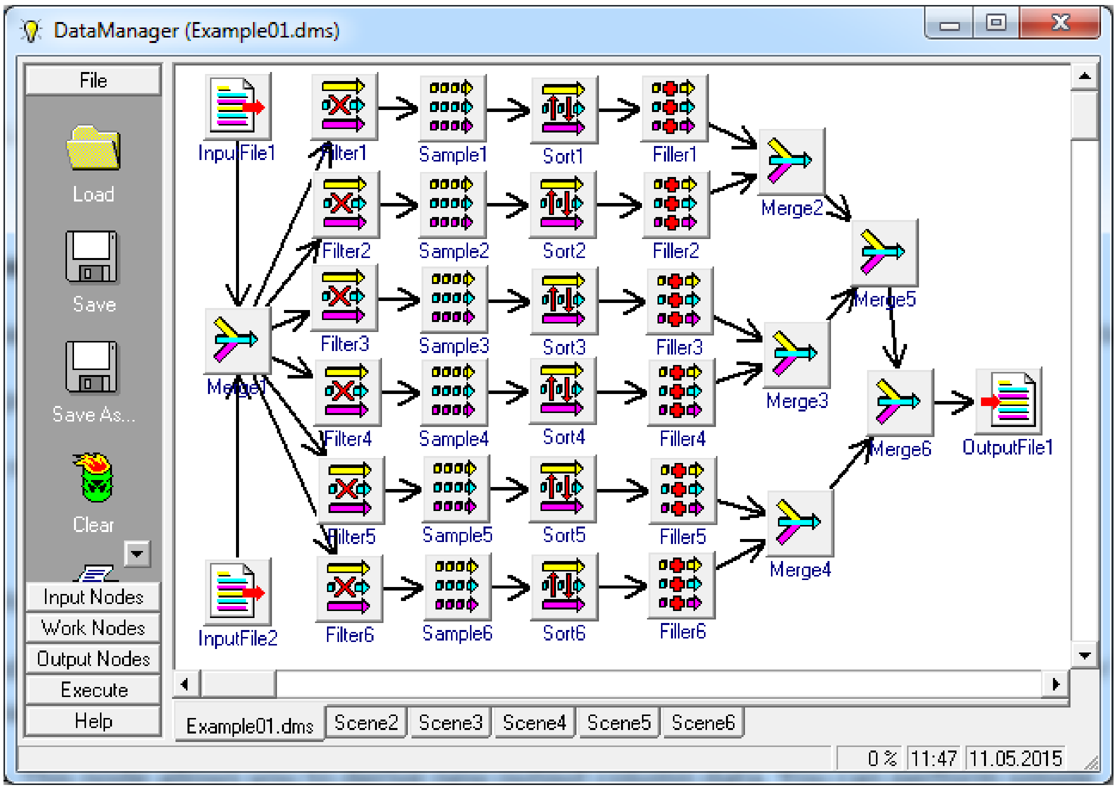


Figure 2.3: Example of DataManager merging and transforming two data sets into a merged output data set [23].

Time-oriented Data Support

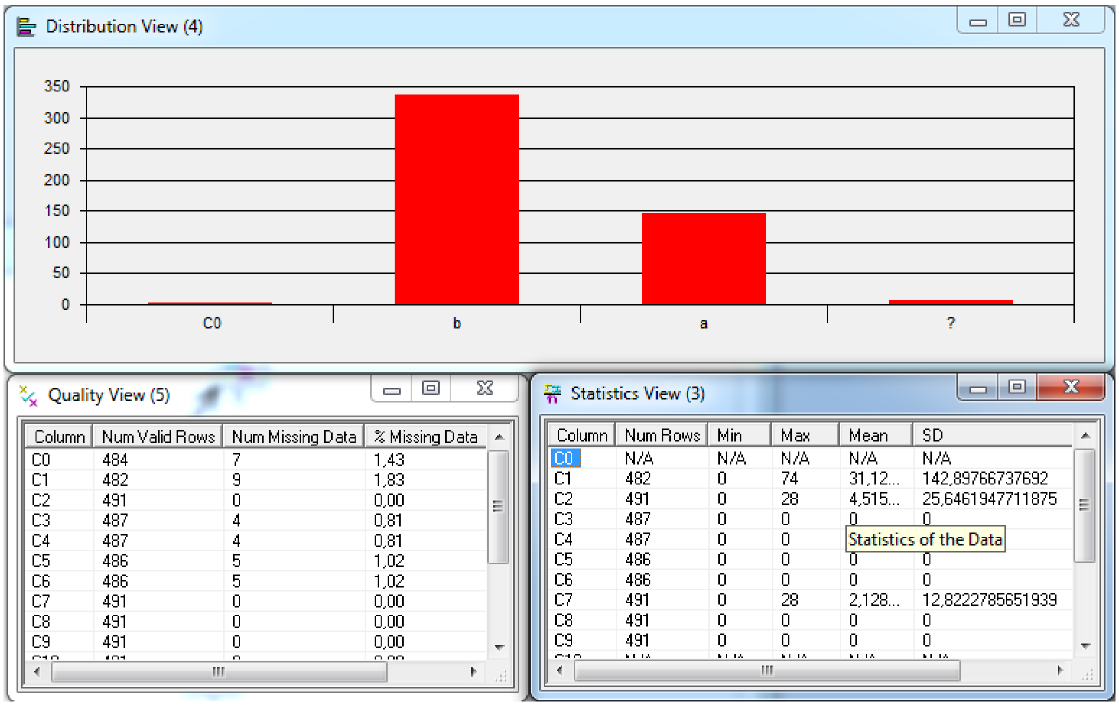
DataManager processes numerical and textual data types, however time-oriented data is not supported.

2.3.4 Profiler

Description

Profiler [15] is a visual analysis web application integrated with Wrangler's [8] data transformation engine. After importing a data set into the browser, users can interactively explore the data's quality. Five categories of anomalies are handled: missing, erroneous and inconsistent data, extreme values, and key violations. When clicking on a category in the anomaly browser (see Figure 2.5) Profiler automatically provides a number of different visualizations to give an overview of invalid data.

With the help of Profiler's invented suggestion engine, the system automatically provides visualizations so that users don't have to bother with the selection of adequate visualizations for different data attributes. Every visualization is a facet which can be



Column	Num Valid Rows	Num Missing Data	% Missing Data
C0	484	7	1,43
C1	482	9	1,83
C2	491	0	0,00
C3	487	4	0,81
C4	487	4	0,81
C5	486	5	1,02
C6	486	5	1,02
C7	491	0	0,00
C8	491	0	0,00
C9	491	0	0,00

Column	Num Rows	Min	Max	Mean	SD
C0	N/A	N/A	N/A	N/A	N/A
C1	482	0	74	31,12...	142,89766737692
C2	491	0	28	4,515...	25,6461947711875
C3	487	0	0	0	0
C4	487	0	0		
C5	486	0	0	0	0
C6	486	0	0	0	0
C7	491	0	28	2,128...	12,8222785651939
C8	491	0	0	0	0
C9	491	0	0	0	0

Figure 2.4: DataManager Analysis results [23]. Distribution View showing the distribution of values from the data set (top). Quality View giving an overview about completeness (left). Statistics View displaying numerical statistics (right). The column C0 is a date-time object which statistics are displayed as "N/A".

arranged by users needs. Profiler also "includes built-in support for primitive types — boolean, string, and numeric (int, double) — and higher-order types such as dates and geographic entities; e.g., state/country names, FIPS codes, zip codes." [15, p.4]

Figure 2.6 shows a binned scatter plot between Gross and Worldwide gross, where orange values were identified as outliers. Through selection, these values are highlighted with an orange marking, which then are also highlighted in other visualizations/facets. This technique is called brushing and linking [29].

Profiler is open source and available at Stanford's GitHub Repository [30]. However, the version from May, 2012 is not functional. We failed to fix the project setup due to the high amount of project dependencies and JavaScript errors, and therefore could not test it.

Time-oriented Data Support

Profiler suggests various visualizations for different data types. Numeric types are shown as histograms, while dates are aggregated to time units as days, weeks, months or years (see Figure 2.7). With the help of this visualization, users can identify the date range

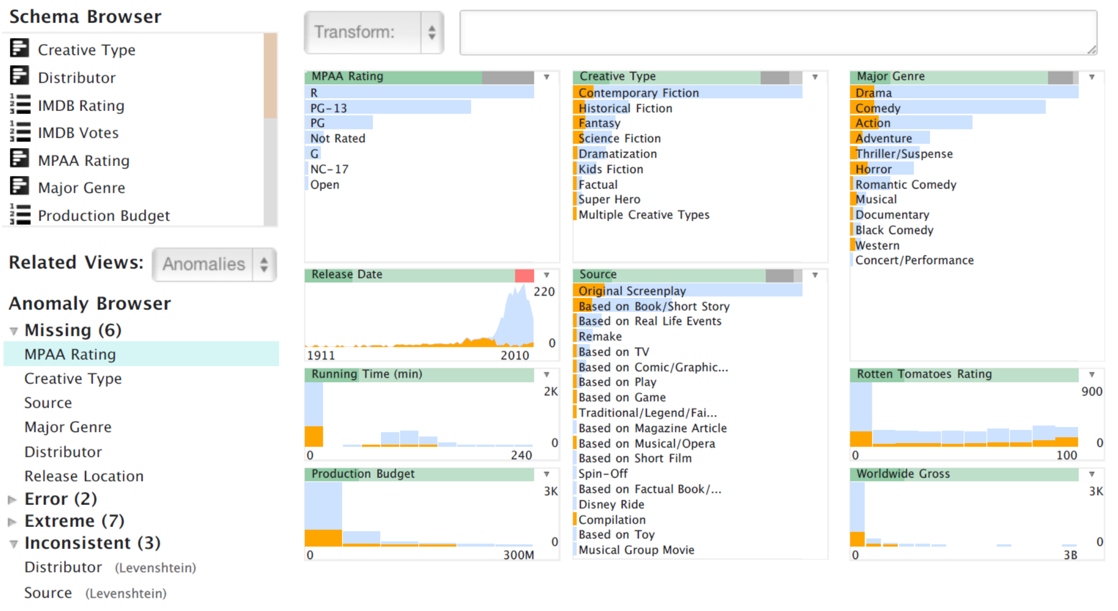


Figure 2.5: GUI of Profiler [15] showing visualizations of different data fields. The gray bar in the 'MPAA rating' window shows missing values. The red bar in the 'Release Date' window shows incompatible types. [15]

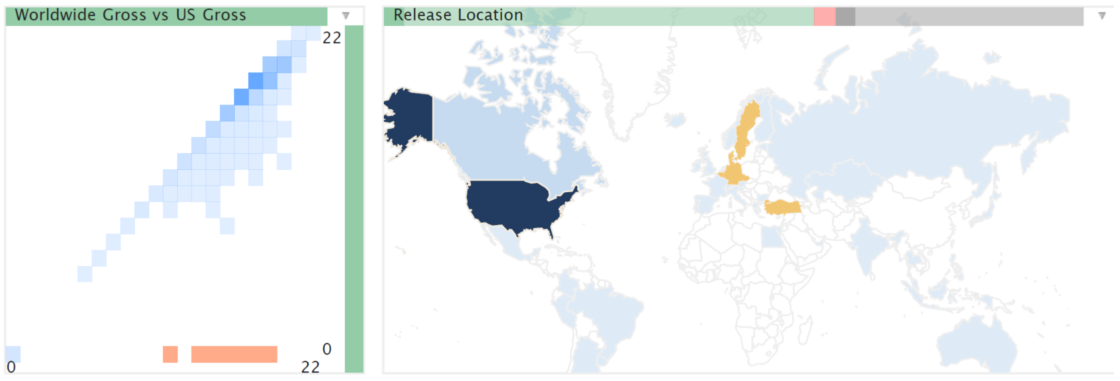


Figure 2.6: Map assessing 2D outliers in a binned scatter plot (left) in Profiler [15]. Linked highlights on the map (right) give an overview about the geological information. [15]

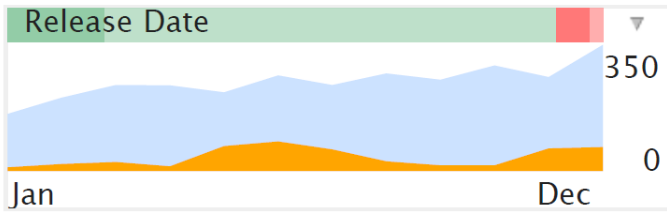


Figure 2.7: Distribution of release dates of a movie data set in Profiler [15].

and get an overview about value distribution over time. Due to missing tests, it is not possible to find out if Profiler's Date Object also supports data quality profiling on the time domain.

2.3.5 OpenRefine

Description

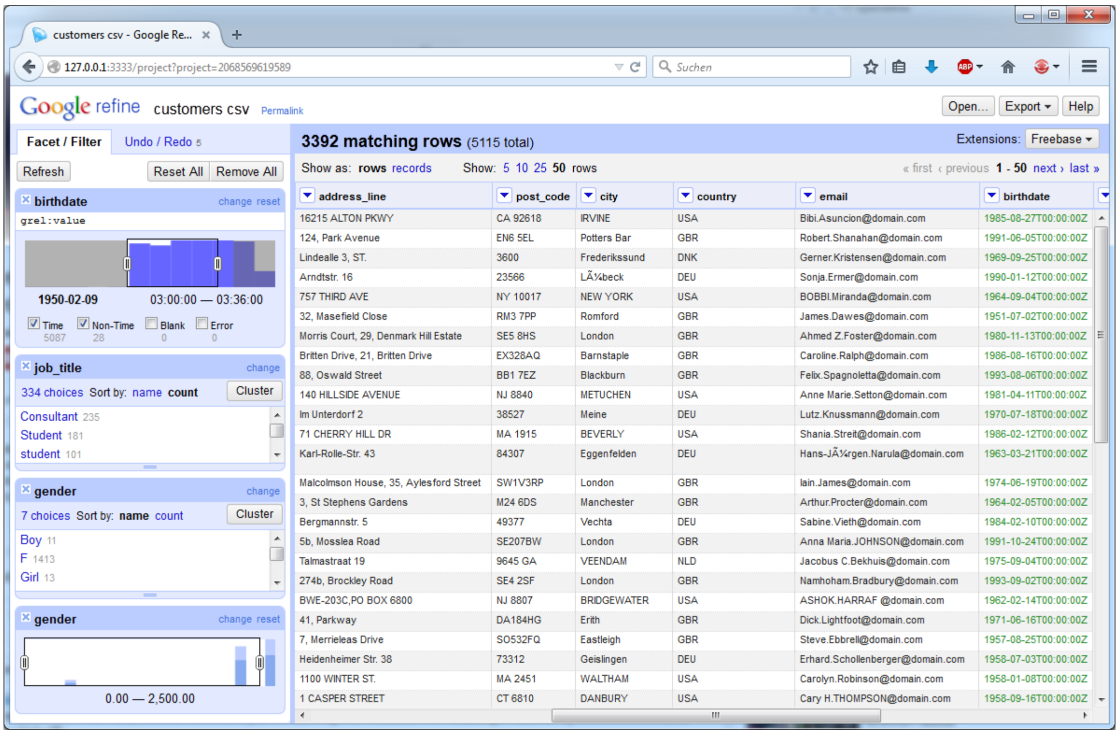
OpenRefine [9] is a web application, formerly developed by Google, which can be run on a local machine or web server. After importing a data set, data columns can be converted to desired data types, concluding: Text, Number or Date.

For every column facets can be defined, which are shown on the left side (see Figure 2.8). Those facets give an overview about the data quality and value distribution or categorical distribution. The value facets can be used as sliders for filtering, while categories can be activated for filtering. With *Google Refine Expression Language (GREL)*, Jython, and Clojure data records can be transformed. Unfortunately, OpenRefine does not support statistics about a column, and the *GREL* does not support calculations with more than one cell. Therefore no statistics like mean, average, minimum, and maximum can be displayed. Open Refine provides a history view, where users actions can be reverted.

Time-oriented Data Support

OpenRefine supports Date formats which contain date and time. It is represented as a JavaScript Date object and can be used in any JavaScript (respectively *GREL*) function for calculations and transformations.

The date facet shows whether invalid or missing values exist, and displays all valid dates as a bar chart. This facet shows the minimum and maximum date records, as well as date distribution (see Figure 2.8, left). With the help of sliders, a date range can be set, which filters the data records in the tabular view.



3392 matching rows (5115 total)

Show as: **rows** records Show: 5 10 25 **50** rows

address_line	post_code	city	country	email	birthdate
16215 ALTON PKWY	CA 92618	IRVINE	USA	Bibi.Asuncion@domain.com	1985-08-27T00:00:00Z
124, Park Avenue	EN6 5EL	Potters Bar	GBR	Robert.Shanahan@domain.com	1991-06-05T00:00:00Z
Lindealle 3, ST.	3600	Frederikssund	DNK	Gerner.Kristensen@domain.com	1969-09-25T00:00:00Z
Arndtstr. 16	23566	LÃ¼beck	DEU	Sonja.Ermer@domain.com	1990-01-12T00:00:00Z
757 THIRD AVE	NY 10017	NEW YORK	USA	BOBBI.Miranda@domain.com	1964-09-04T00:00:00Z
32, Masefield Close	RM3 7PP	Romford	GBR	James.Dawes@domain.com	1951-07-02T00:00:00Z
Morris Court, 29, Denmark Hill Estate	SE5 8HS	London	GBR	Ahmed Z.Foster@domain.com	1980-11-13T00:00:00Z
Britten Drive, 21, Britten Drive	EX328AQ	Barnstaple	GBR	Caroline.Ralph@domain.com	1986-08-16T00:00:00Z
88, Oswald Street	BB1 7EZ	Blackburn	GBR	Felix.Spagnoletta@domain.com	1993-08-06T00:00:00Z
140 HILLSIDE AVENUE	NJ 8840	METUCHEN	USA	Anne Marie.Setton@domain.com	1981-04-11T00:00:00Z
Im Unterdorf 2	38527	Meine	DEU	Lutz.Knussmann@domain.com	1970-07-18T00:00:00Z
71 CHERRY HILL DR	MA 1915	BEVERLY	USA	Shania.Streit@domain.com	1986-02-12T00:00:00Z
Karl-Rolle-Str. 43	84307	Eggenfelden	DEU	Hans-JÃ¼rgen.Narula@domain.com	1963-03-21T00:00:00Z
Malcolmson House, 35, Aylesford Street	SW1V3RP	London	GBR	Iain.James@domain.com	1974-06-19T00:00:00Z
3, St Stephens Gardens	M24 6DS	Manchester	GBR	Arthur.Procter@domain.com	1964-02-05T00:00:00Z
Bergmannstr. 5	49377	Vechta	DEU	Sabine.Vieth@domain.com	1984-02-10T00:00:00Z
5b, Mosslea Road	SE207BW	London	GBR	Anna Maria.JOHNSON@domain.com	1991-10-24T00:00:00Z
Talmastraat 19	9645 GA	VEENDAM	NLD	Jacobus C.Bekhuis@domain.com	1975-09-04T00:00:00Z
274b, Brockley Road	SE4 2SF	London	GBR	Namhoham.Bradbury@domain.com	1993-09-02T00:00:00Z
BWE-203C,PO BOX 6800	NJ 8807	BRIDGEWATER	USA	ASHOK.HARRAF @domain.com	1962-02-14T00:00:00Z
41, Parkway	DA184HG	Erith	GBR	Dick.Lightfoot@domain.com	1971-06-16T00:00:00Z
7, Merrieleas Drive	SO532FQ	Eastleigh	GBR	Steve.Ebbrell@domain.com	1957-08-25T00:00:00Z
Heidenheimer Str. 38	73312	Geislingen	DEU	Erhard.Schollenberger@domain.com	1958-07-03T00:00:00Z
1100 WINTER ST.	MA 2451	WALTHAM	USA	Carolyn.Robinson@domain.com	1958-01-08T00:00:00Z
1 CASPER STREET	CT 6810	DANBURY	USA	Cary H.THOMPSON@domain.com	1958-09-16T00:00:00Z

Figure 2.8: GUI of OpenRefine [9]. Facets giving overview and selection (left) for the tabular view of data records (right).

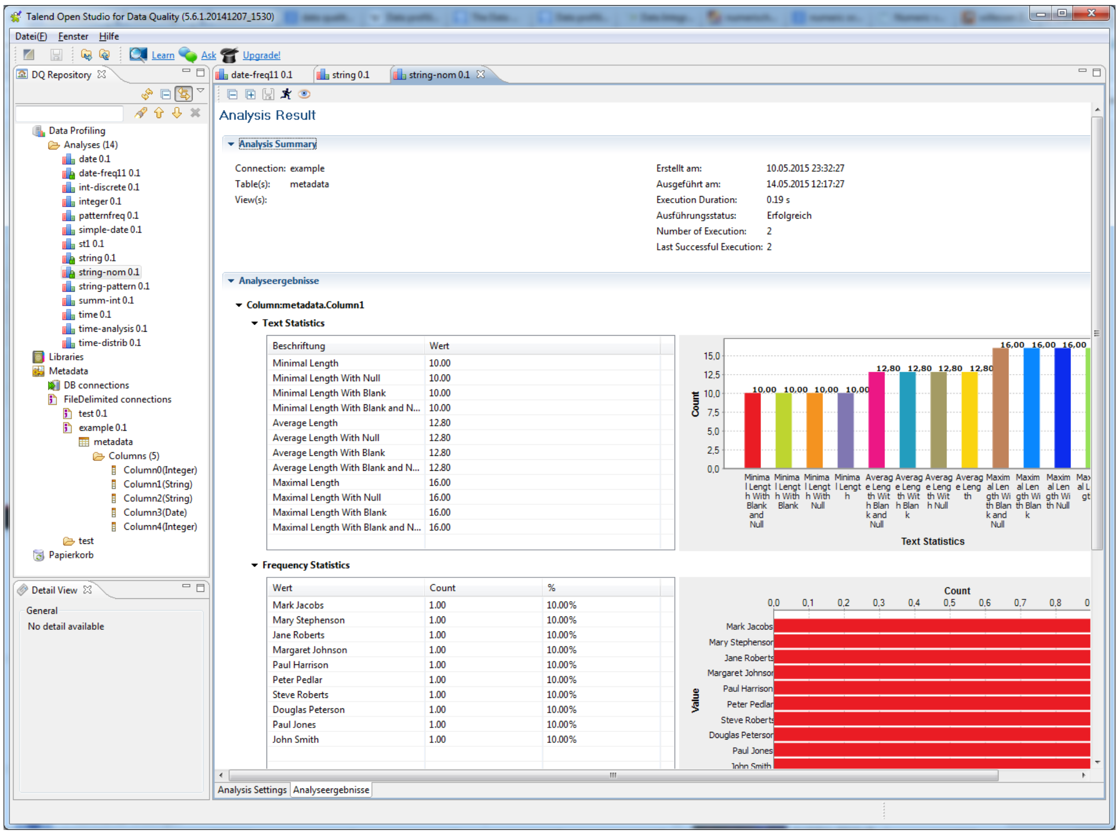
2.3.6 Talend Open Studio

Description

Talend Open Studio [26] is an open source Data Profiling application. Delimited files (e.g., CSV and Microsoft Excel) and database connections can be used as data source. The data set then is stored as relational tables, whereas columns are declared and converted to different data types (e.g., String, Boolean, Integer, Double, Date). Due to type declaration of each column, different quality profiling techniques can be performed. While numerical data types like integer and double can be analyzed with statistical methods, textual data types can be analyzed with blank spaces, words, and patterns.

Limited to database connections, Talend Open Studio provides analyses about the database structure, f.i., number of tables, rows per table, indexes, and primary keys. Match analysis helps to identify duplicates or gives an overview about defined groups of the data set (e.g., setting numerical ranges to group the data set). Talend Open Studio also provides correlation analyses (see Figure 2.10) where correlated values are displayed as a network graph, to discover data quality issues by looking at it.

Beside those analyses, Talend Open Studio provides statistics about numerical, textual and date types. Numerical values are analyzed in terms of missing data, minimum and



Analysis Result

Analysis Summary

Connection: example
Table(s): metadata
View(s):

Erstellt am: 10.05.2015 23:32:27
Ausgeführt am: 14.05.2015 12:17:27
Execution Duration: 0.19 s
Ausführungsstatus: Erfolgreich
Number of Execution: 2
Last Successful Execution: 2

Analyseergebnisse

Column:metadata.Column1

Text Statistics

Beschriftung	Wert
Minimal Length	10.00
Minimal Length With Null	10.00
Minimal Length With Blank	10.00
Minimal Length With Blank and N...	10.00
Average Length	12.80
Average Length With Null	12.80
Average Length With Blank	12.80
Average Length With Blank and N...	12.80
Maximal Length	16.00
Maximal Length With Null	16.00
Maximal Length With Blank	16.00
Maximal Length With Blank and N...	16.00

Frequency Statistics

Wert	Count	%
Mark Jacobs	1.00	10.00%
Mary Stephenson	1.00	10.00%
Jane Roberts	1.00	10.00%
Margaret Johnson	1.00	10.00%
Paul Harrison	1.00	10.00%
Peter Pedlar	1.00	10.00%
Steve Roberts	1.00	10.00%
Douglas Peterson	1.00	10.00%
Paul Jones	1.00	10.00%
John Smith	1.00	10.00%

Figure 2.9: GUI of Talend Open Studio [31]. Repository view containing analyses and data sets (left). Results of a textual analysis (right).

maximum values, mean and the median. Statistics about textual data types contain the length and even statistics about the length including and/or excluding white spaces.

Time-oriented Data Support

Talend Open Studio provides simple statistics (minimum, maximum, and missing values) about dates. However, a finer granularity (e.g., hours, minutes, and seconds) is not supported.

Furthermore, this application provides analyses about date distributions in case of years, months, weeks and weekdays. Figure 2.11 shows an example of a month distribution.

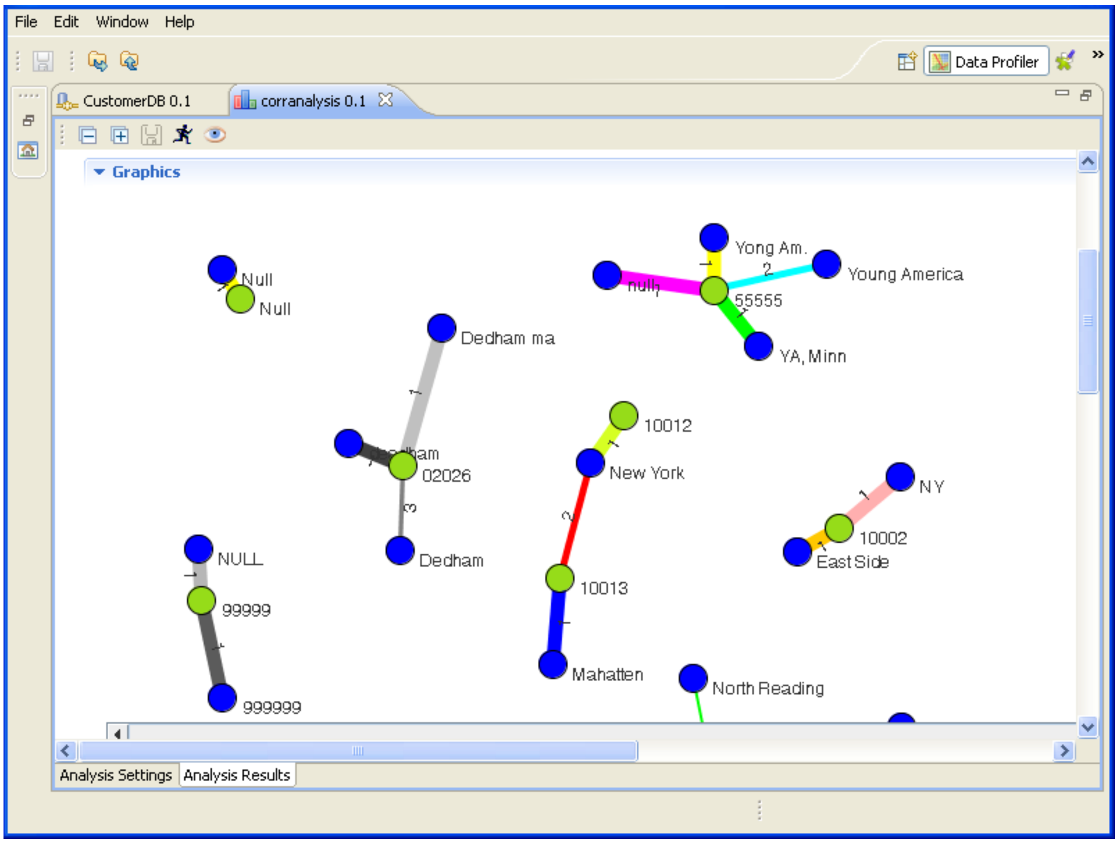
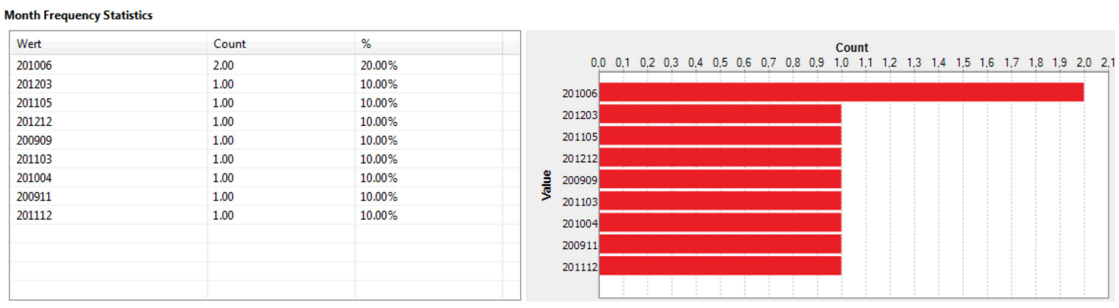


Figure 2.10: Nominal Correlation Analysis in Talend Open Studio displaying correlated values as a network graph to find data quality issues by looking at it [31].


Month Frequency Statistics

Wert	Count	%
201006	2.00	20.00%
201203	1.00	10.00%
201105	1.00	10.00%
201212	1.00	10.00%
200909	1.00	10.00%
201103	1.00	10.00%
201004	1.00	10.00%
200911	1.00	10.00%
201112	1.00	10.00%

Figure 2.11: Month distribution in Talend Open Studio [31].

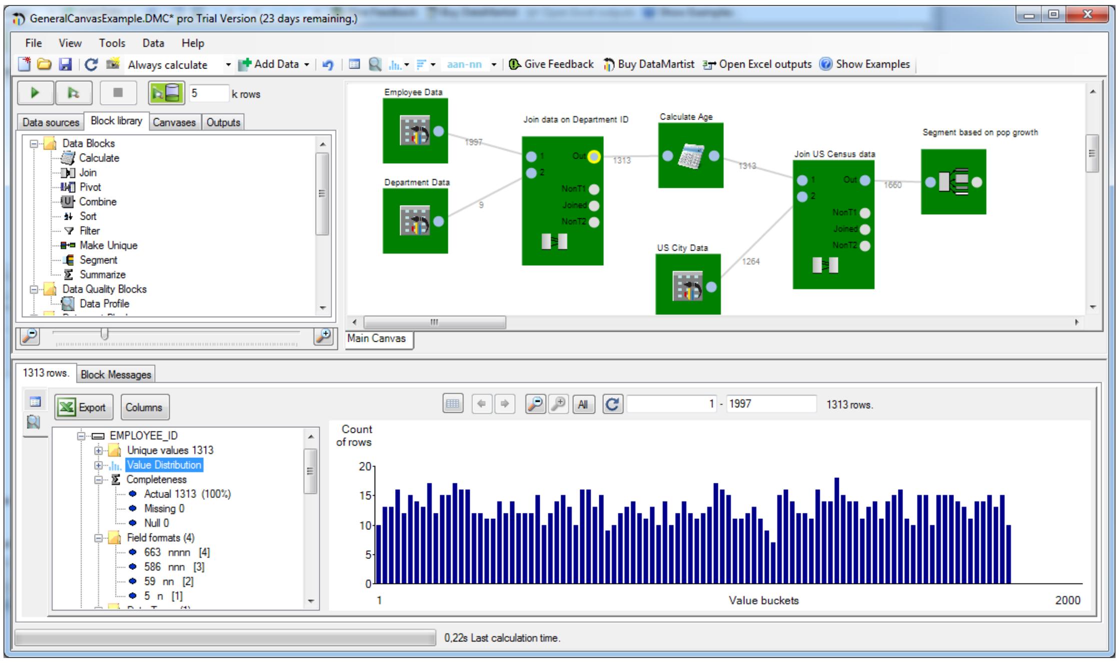


Figure 2.12: GUI of Datamartist [27]. Library and worksheet (top). Value distribution explorer (bottom) of employee IDs of the selected Join Stub (yellow circle).

2.3.7 Datamartist

Description

Datamartist [27] is another graph based approach for Data Profiling and wrangling. Every node - called "block" - can be linked together to transform data to get the desired (cleaned) output. Each block can consist of at least one input and output stub. Each of these stubs contain a data set which can be profiled.

Datamartist provides different profiling visualizations. Figure 2.12 shows the Value distribution explorer of employee IDs of a joined data set. In Figure 2.13 the category count explorer is shown, where the count of data records with the same category are visualized. The Field Format explorer gives an overview about the format of values (e.g., *nnn* for *586*, and *nn* for *59*. See Figure 2.12, bottom left).

Datamartist provides simple techniques like join, filter, sort, and calculate, as well as duplicate detection to transform and cleanse data.

Time-oriented Data Support

Datamartist supports the data types Date and DateTime on which operations can be performed, f.i., Week Day calculation, Date conversion, and whether the date is a weekday. With this, users have the possibility to transform data on the basis of time-oriented

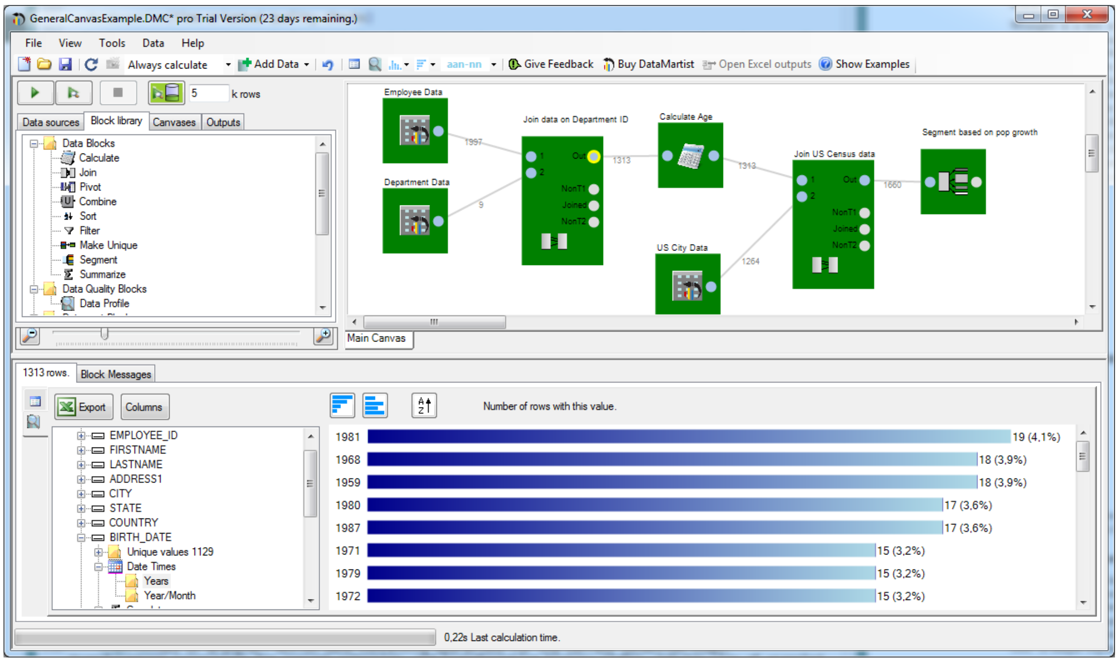


Figure 2.13: Category count explorer giving an overview about distribution in years from the field birth-date in Datamartist [27].

calculations. The program also supports a year and month distribution, but provides no finer granularity.

2.3.8 DataCleaner

Description

DataCleaner [24] is a commercial software, which has been tested with a demo version. It is also built up on a graph based approach, where data sources can be linked with converters, analyzers or transformations (see Figure 2.14). This application can be used as a data profiler, wrangler and cleansing tool.

After opening a data set from a database or delimited file, each column is initially interpreted as a String. Converters can be used to change columns to specific data types (Boolean, Date, Number, and String). After refining the data types, data quality analyses like duplication detection and pattern matching can be performed. Data Wrangling can be performed with filtering, coalescing, and calculating data records. DataCleaner provides predefined matchers for String values, such as URL, Country and email standardizing, which can be used to extract or split values into multiple columns (e.g., the email address into user name and domain).

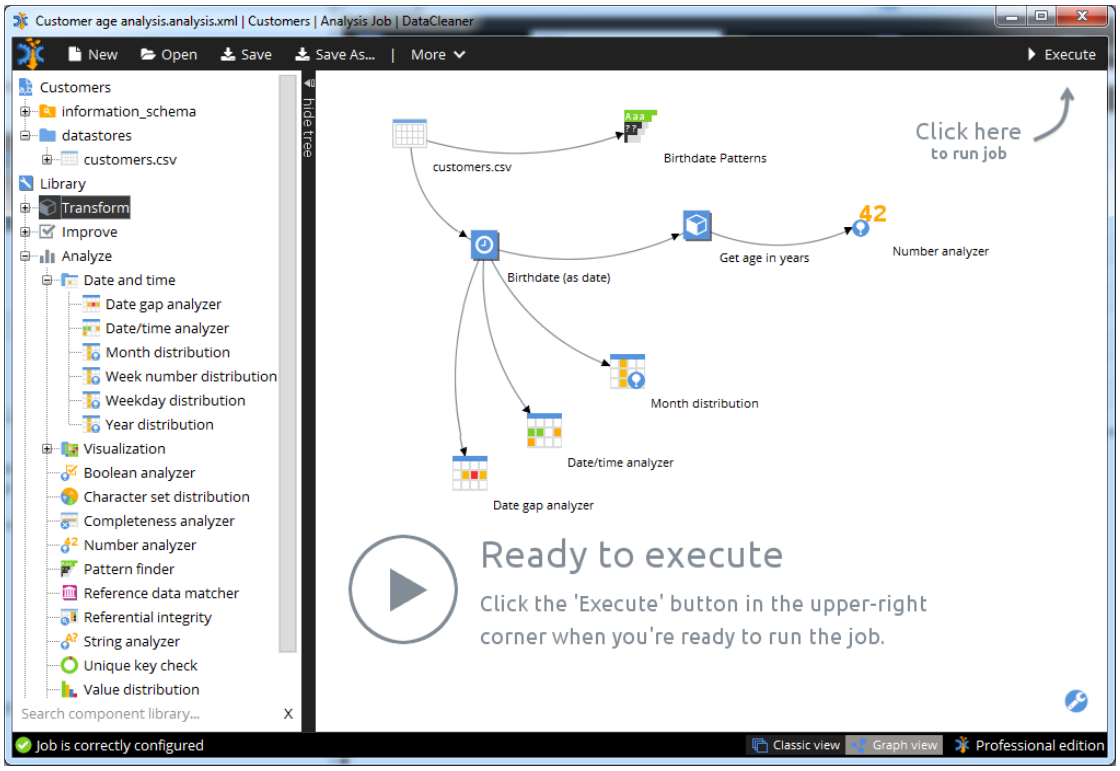


Figure 2.14: GUI of DataCleaner [24]. Overview about data stores, transformations, improvement and analysis tools (left). Graph-based work sheet (right).

Time-oriented Data Support

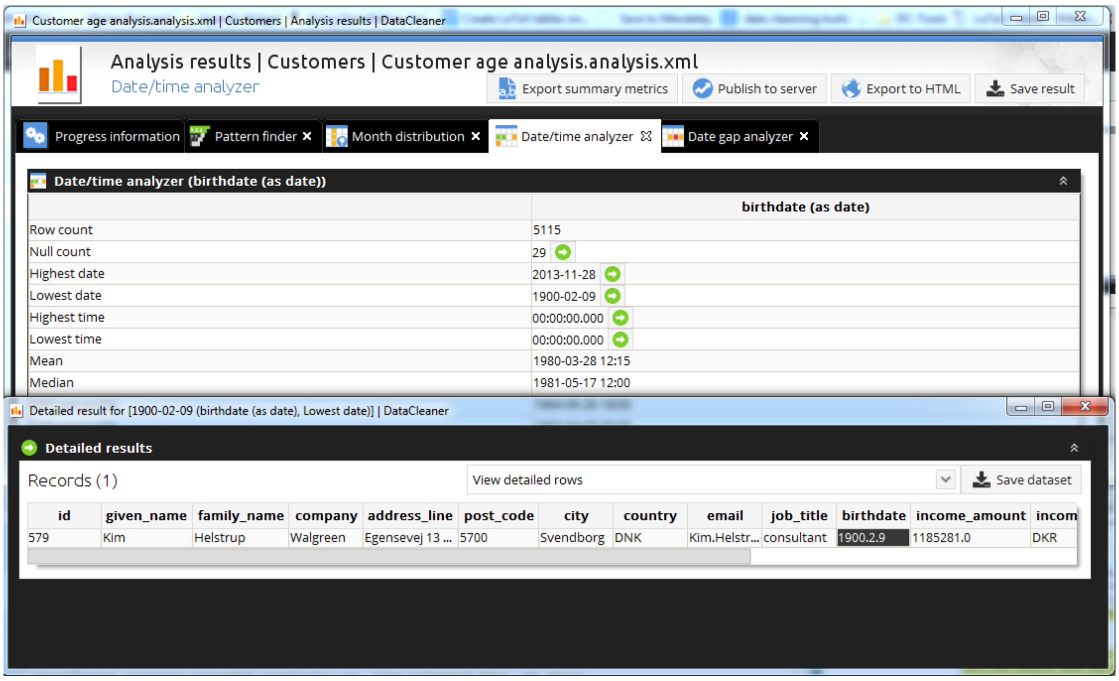
DataCleaner supports extended statistics on time-oriented data: highest and lowest date, highest and lowest time, mean and median (see Figure 2.15).

The program also provides a date gap analyzer which opens an interactive visualization, displaying gaps between dates which can be scrolled and zoomed. Another feature is a distribution over years, months, weeks or weekdays. With supportive calculations of date differences, period lengths, and timestamps, own quality checks can be defined, inspected or used for transformations (e.g., in a filter).

2.3.9 TimeCleanser

Description

TimeCleanser [22] is a desktop application prototype specialized on time-oriented Data Cleansing. This prototype has been developed over 3 years with a user-centered design process, aiming to address time-oriented data quality problems. It is not accessible since it was specially developed for a company, therefore information about its features were extracted by the corresponding research paper.


Customer age analysis.analysis.xml | Customers | Analysis results | DataCleaner

Analysis results | Customers | Customer age analysis.analysis.xml
Date/time analyzer

Date/time analyzer (birthdate (as date))

	birthdate (as date)
Row count	5115
Null count	29
Highest date	2013-11-28
Lowest date	1900-02-09
Highest time	00:00:00.000
Lowest time	00:00:00.000
Mean	1980-03-28 12:15
Median	1981-05-17 12:00

Detailed result for [1900-02-09 (birthdate (as date), Lowest date)] | DataCleaner

Detailed results

Records (1)

id	given_name	family_name	company	address_line	post_code	city	country	email	job_title	birthdate	income_amount	incom
579	Kim	Helstrup	Walgreen	Egensevej 13 ...	5700	Svendborg	DNK	Kim.Helstr...	consultant	1900.2.9	1185281.0	DKR

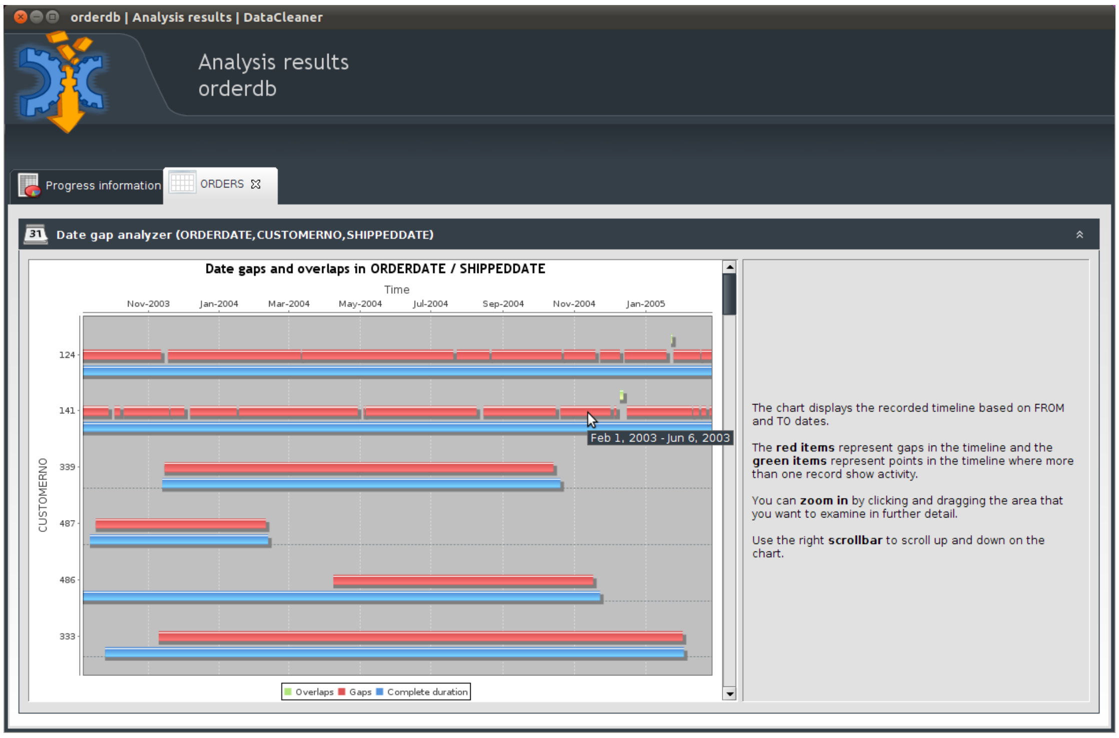
Figure 2.15: Date/Time Analyzer of DataCleaner giving statistics about time-oriented data (top). With selecting the green arrow symbol next to “Lowest date” details about the corresponding data record is displayed in a new window (bottom) [24].

Time-oriented Data Support

While other tools provide simple statistics about a single date field, in TimeCleanser users have to assign semantic roles to different data columns. “In particular, the user needs to declare which column contains ‘from’ values, which contains ‘to’ values (or timestamps, or durations, respectively), and which column contains the associated data values that are to be analyzed.” [22, p.5]

With the help of semantic roles, specialized visualizations with period length can be provided. Figure 2.17 shows a heat map of working hours of employees. The red circle indicates a possible data quality problem, where an employee worked over night. The analyst has to verify if this data record is plausible or not.

Another example is the visualization of values over time. In Figure 2.18 (top), raw values over times are displayed. We can see constant high values, which indicate an error. Figure 2.18 (center) display the difference between subsequent values, where analysts can see that values did not change over a period of time. Figure 2.18 (bottom) shows the length of intervals with constant values as bar plot. The outliers in this visualization also indicate a data quality problem. Beside visualizations to identify implausible values, TimeCleanser also provides temporal constraint checks, like duration constraints, gap constraints, and if entries for each ID cover a specified time span.


orderdb | Analysis results | DataCleaner
Analysis results
orderdb
Progress information
ORDERS
Date gap analyzer (ORDERDATE,CUSTOMERNO,SHIPPEDDATE)
Date gaps and overlaps in ORDERDATE / SHIPPEDDATE
Time
Nov-2003
Jan-2004
Mar-2004
May-2004
Jul-2004
Sep-2004
Nov-2004
Jan-2005
CUSTOMERNO
124
141
339
487
486
333
Feb 1, 2003 - Jun 6, 2003
🟩 Overlaps 🟥 Gaps 🟦 Complete duration

The chart displays the recorded timeline based on FROM and TO dates.

The **red items** represent gaps in the timeline and the **green items** represent points in the timeline where more than one record show activity.

You can **zoom in** by clicking and dragging the area that you want to examine in further detail.

Use the right **scrollbar** to scroll up and down on the chart.

Figure 2.16: Date gap analyzer displaying gaps from customers order dates and ship dates [32].

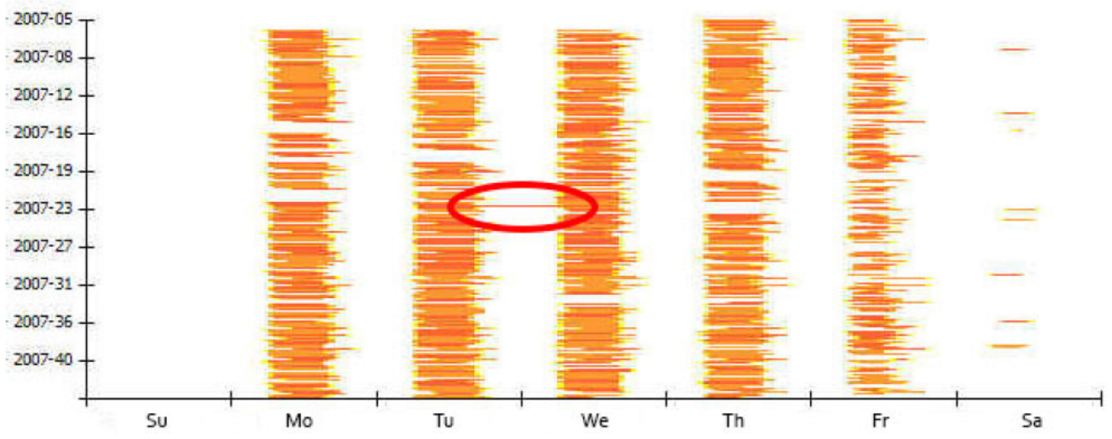


Figure 2.17: Heat map of working hours of employees in TimeCleanser [22].

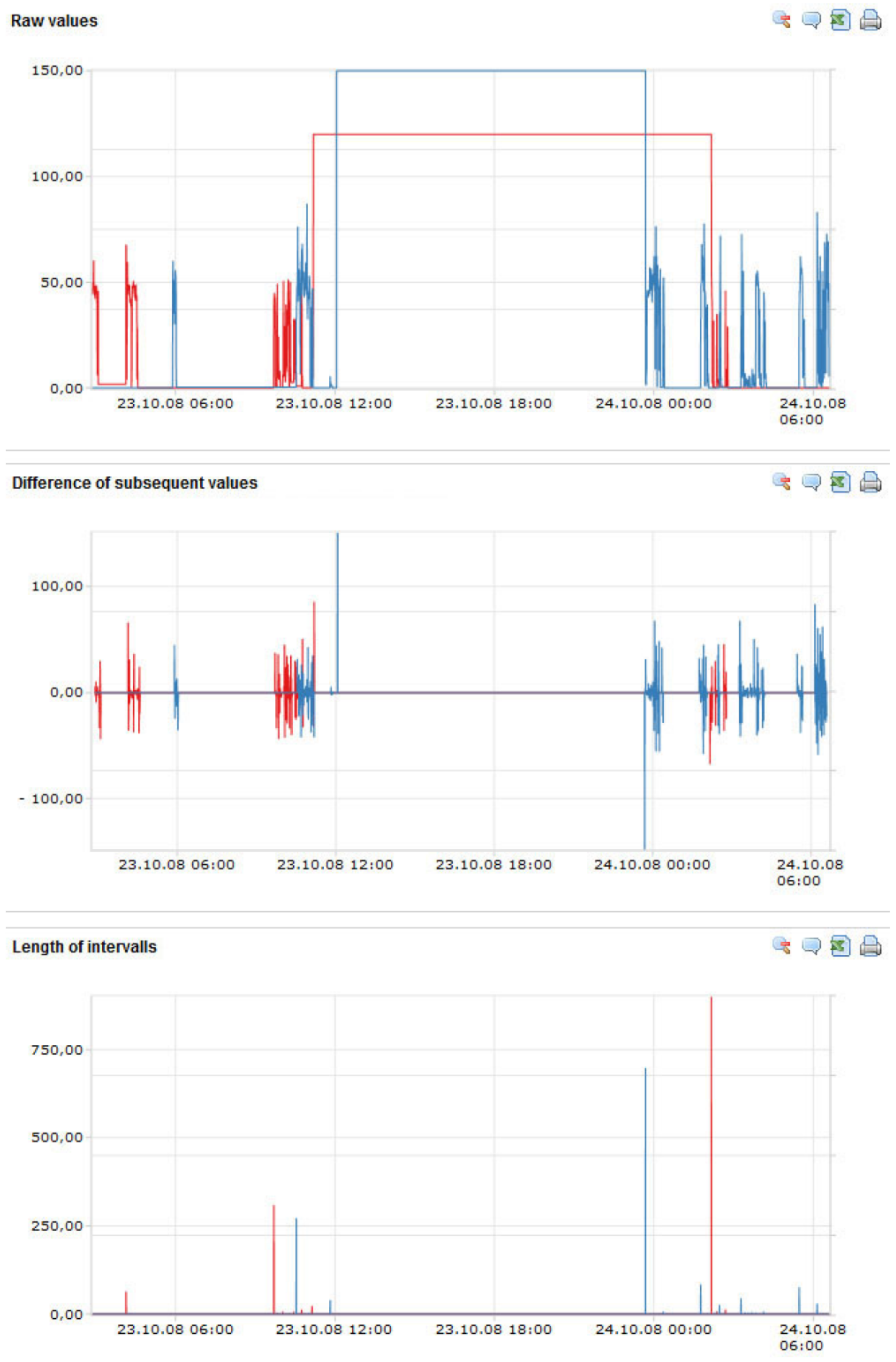


Figure 2.18: Overview of values over time (top). Difference plot of subsequent data values (center). Interval length as bars over time (bottom) [22].

2.3.10 Discussion

Table 2.4 gives an overview about Data Profiling techniques as a feature matrix. Each column represents a data quality tool, where rows define Data Profiling techniques. These features were selected on the basis of the programs features itself. When a tool provided a feature for a specific data quality problem, it was added to the list and compared with the other selected applications.

The following sections give a short summary about the possibilities and features of each application. Further information about each tool can be found in Section 2.3.

Wrangler

Looking at the table, you can see that Wrangler [8] only shows missing values. As an additional note the program also displays whether data is valid for a specified data type. Since Wranglers focus lies on Data Wrangling, the Data Profiling features are sufficient. Wrangler supports a date type, but processes it only as a text format (so dates like 2010-13-32 are possible).

Data Match 2013

Data Match 2013 [25] is focused on detection and removal of duplicate data records. It provides simple statistics for different data types, as well as date formats. However, the time domain is not processed and the duplicate detection function handles date objects as Strings.

DataManager

DataManager [23] is a graph-based Data Wrangling tool, which generally provides Data Profiling statistics tools. Unfortunately this program can only handle numeric and text data types. Time-oriented data is not supported.

Profiler

Profiler [15] is focused on Data Profiling and therefore provides many different visualizations for various data types with semantic roles. With this application missing, erroneous, and inconsistent data, extreme values, and key violations can be detected. Date objects can be processed, and visualized as histogram. It is unknown if the date type also contains the time domain.

OpenRefine

OpenRefine [9] looks like a combination of Wrangler [8] and Profiler [15] focused on wrangling a data set with the help of Data Profiling statistics and visualizations (facets in this case). Unlike Wrangler, OpenRefine supports date objects and visualize them as a histogram facet, which can be used for detecting outliers, getting an overview about

	Wrangler [8]	Profiler [15]	TimeCleanser [22]	OpenRefine [9]	DataManager [23]	DataCleaner [24]	Data Match 2013 [25]	Talend Open Studio [26]	Datamartist [27]
Statistics									
Missing Values	•	•		•	•	•	•	•	•
Missing Data (%)					•		•	•	•
Distinct Values		•				•	•	•	•
Numerical Statistics									
Minimum/Maximum		•			•	•	•	•	•
Mean					•	•	•	•	•
Median							•	•	
Standard Deviation					•	•			
Textual Statistics									
White Spaces						•	•	•	
Upper- and Lowercase						•			
Words						•			
Length								•	
Time-oriented Statistics									
Minimum/Maximum		•	•	•		•		•	•
Mean			•			•	•		
Features									
Value Distribution/Histogram		•		•	•	•		•	
Duplicate Detection		•	•	•		•	•	•	•
Pattern Matching				•		•		•	•
Correlation Analysis				•				•	
Time-oriented Visualizations									
Date Gaps			•			•			
Date Distribution		•	•	•		•		•	•
Period Length			•						
Time-oriented Constraints									
Period Length			•						
Time Gap			•						

Table 2.4: Comparison of Data Profiling techniques (• ...supported by application)

value distribution, and defining temporal ranges for a filter. Date types can be used in any calculations with the integrated *Google Refine Expression Language*.

Talend Open Studio

Talend Open Studio [26] is a Data Profiling tool, providing many analyses for different data types. Numerical data is analyzed with mathematical statistics, while textual data provides additional information about white spaces. A network graph can help to find data quality issues with correlated data. Although Talend Open Studio supports date as a data type, only year and month distribution views are provided. The time domain is not processed.

Datamartist

Datamartist [27] is another Data Wrangling tool with the powerful ability to profile the data set in each state, before and after a specified transformation. Value distribution visualizations, category count explorers, missing values, and pattern detection make it to a solid Data Wrangling tool with integrated Data Profiling techniques. Datamartist supports a Date and DateTime format, which can be used in calculations and therefore in filtering and cleansing techniques. It also provides a date distribution visualization (limited to years and months).

DataCleaner

DataCleaner [24] is another graph-based Data Wrangling software with integrated Data Profiling techniques. Pattern finding, statistics, and transformations combine it to a powerful tool. This program has the most statistics about DateTime objects: minimum and maximum date and time, as well as median. Besides date distributions aggregated with years, months, weeks or weekdays, DataCleaner provides a date gap analyzer, visualizing gaps from specified temporal values.

TimeCleanser

Previous mentioned data quality tools partially support time-oriented data, but most of them lack on finding temporal data quality problems (e.g., period length checks). Besides providing statistics about invalid temporal data, f.i., with a wrong date format, even valid entries can be implausible. Discovering implausible values can be checked with formal tests, f.i., if a delivery time of a product is longer than a week, this value can be declared as implausible. Often a formal definition for implausible values is impossible, because of hazy constraints. Due to contextual circumstances delivery times can exceed a specified duration, which can not automatically detected. Therefore insight of an analytic is needed.

TimeCleanser [22] is specialized on time-oriented data quality problems and does mainly provide time-oriented data quality checks. Beside statistic means, this prototype provides syntax checks, like duration constraints and gap constraints. With the help of semantic roles, data values can be set as points in time or intervals, with different settings. F.i., some time intervals may allow overlapping, while other overlapping intervals should be detected as erroneous. In addition, TimeCleanser provides visualizations like period length heat maps to identify implausible values, as well as value differences over time. All in all, this prototype provides the most techniques for time-oriented data quality checks.

2.4 Visualization Techniques

This section covers visualization techniques that are not specifically focused on data quality but on the visualization of time-oriented data. The following concepts were found with the help of TimeViz [21], an interactive online browser of visualization techniques which are covered in the book “Visualization of Time-Oriented Data” [33].

2.4.1 ThemeRiver

ThemeRiver [34] is a visualization of thematic trends of document collections. The visualization flows from left to right through time, getting thicker along with the strength of specific themes (see Figure 2.19). With ThemeRiver temporal patterns like trends or relationships may be discerned. The visualization technique behind the prototype can also be used for visualizing other quantitative data.

2.4.2 Tile Map

Tile Maps [35] help to visualize long term trends and seasonal patterns of time-oriented data. The main idea behind this concept is to arrange data based on different time granularities. A tile (cell) visualizes values of a day due to its lightness, whereas rows display *Days of Week* and columns represent *Weeks* (see Figure 2.20). Seasonal patterns and long term trends can be conveyed due to its two-dimensional representation as our experience to look at calendars.

2.4.3 Cycle Plot

Cycle Plots [36] are visualizations to discern seasonal and trend components in time-oriented data. The visualization consists of nested line plots inside a plot showing a seasonal pattern. In Figure 2.21 (left) the Cycle Plot shows sales over four *Weeks* combined with each corresponding *Day of Week*. With this visualization an upward trend is visible for Mondays, whereas sales are decreasing on Tuesdays. Generally the seasonal pattern reveals a sales peak on Wednesday.

2.4.4 Spiral Graph

Spiral Graphs [37] visualize time-series data along a spiral so that periodic structures can be identified. They support the comparison of several cycles, which length can be

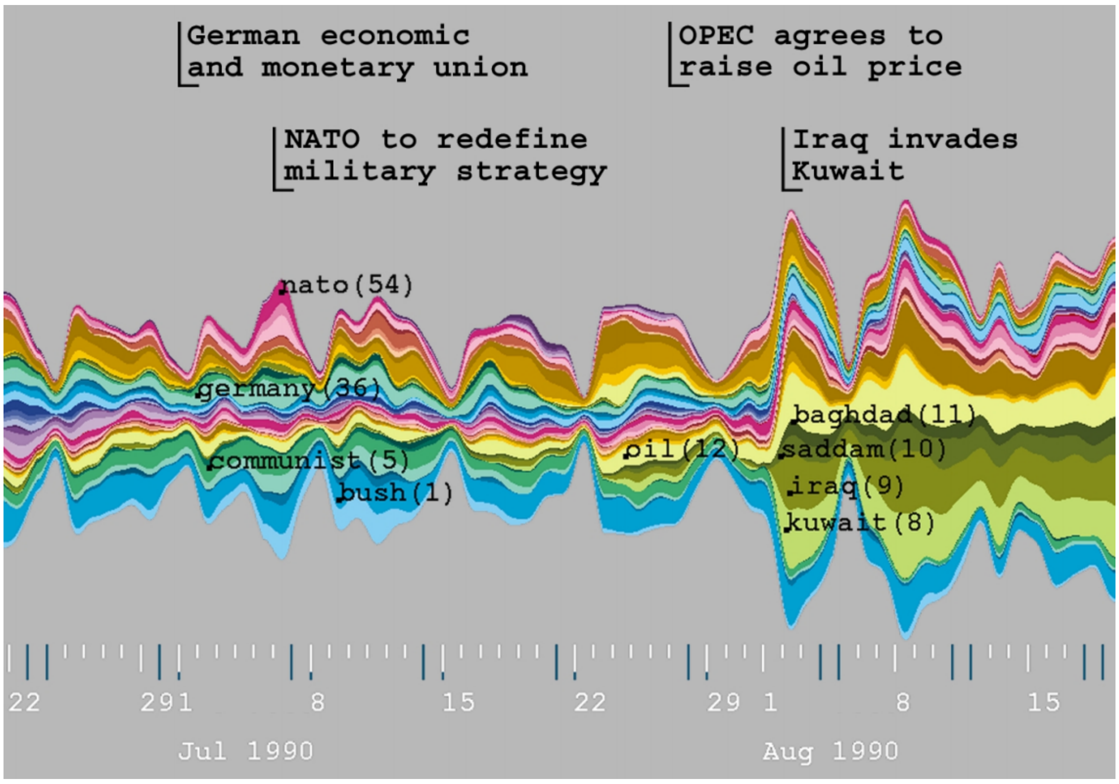


Figure 2.19: Example of a ThemeRiver showing the trend of different themes over time using a river metaphor [34].

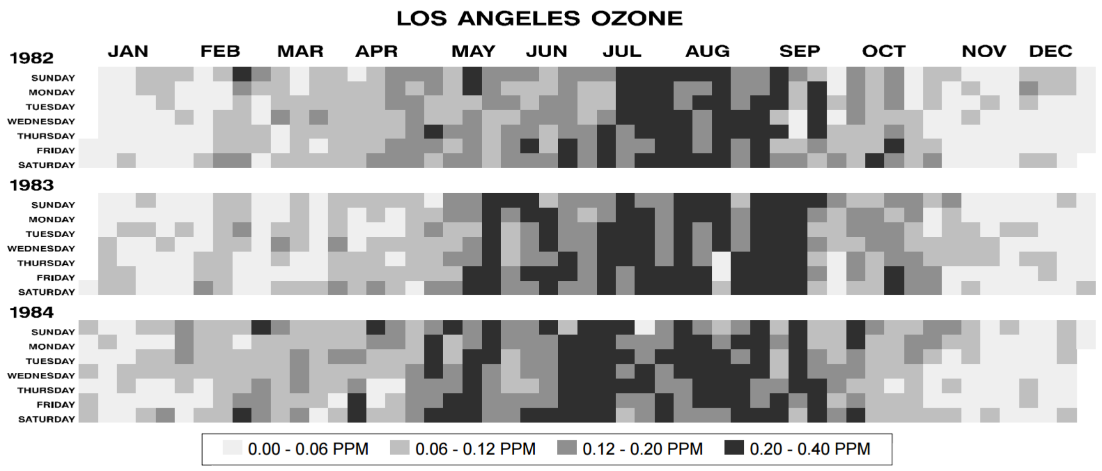


Figure 2.20: Example of a Tile Map displaying ozone concentrations with a different brightness inside tiles which stand for single days of a year [35].

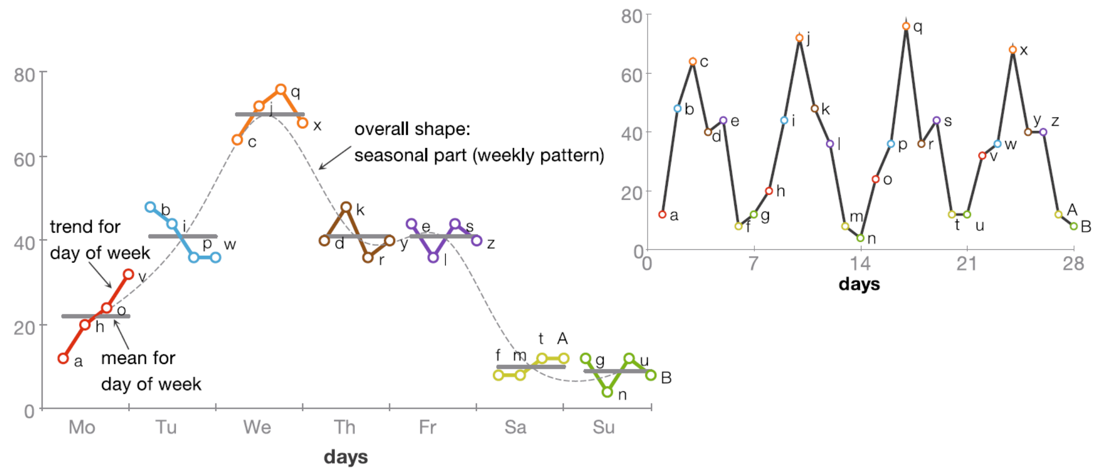


Figure 2.21: Cycle Plot combining different time granularities so that seasonal and trend components can be discerned (left), which is almost impossible with line plots (right) [36].

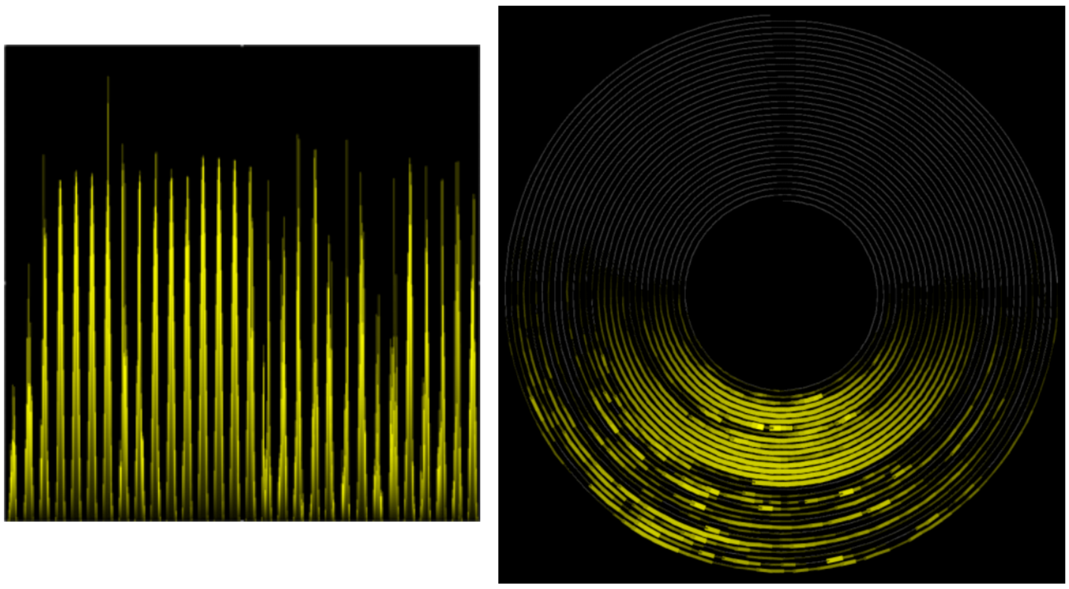


Figure 2.22: Example visualizations of a months sunshine intensity using a Spiral Graph (right), in comparison with a line chart (left) using the same data [37].

interactively set by the user. Figure 2.22 (left) shows an example of a Spiral Graph displaying sunshine intensity (quantitative data) with a twenty-four hour cycle.

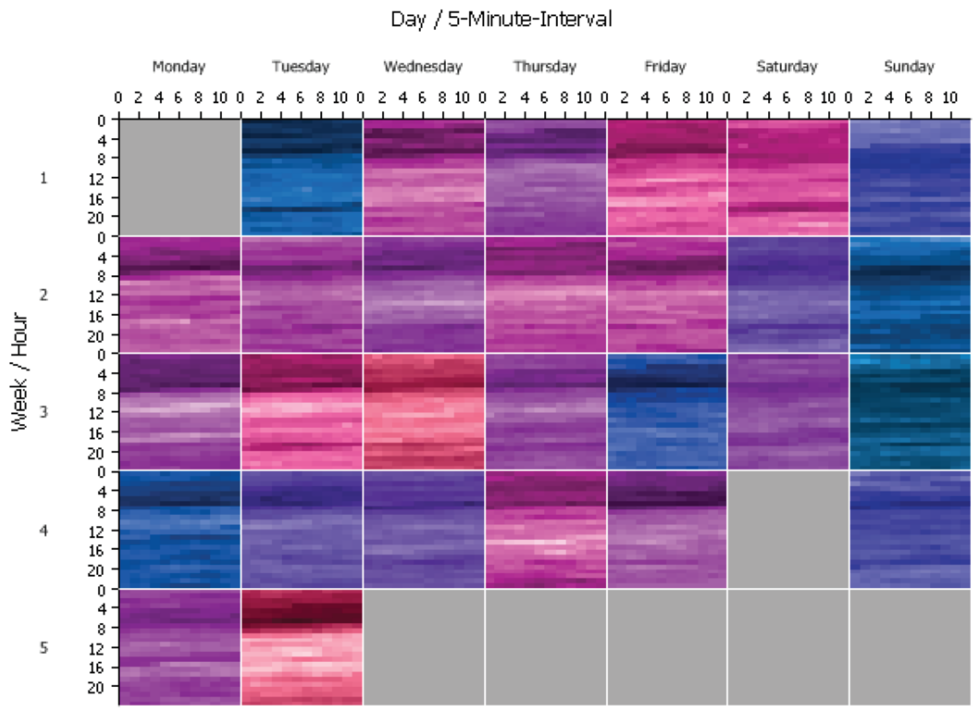


Figure 2.23: GROOVE Visualization showing the amount of police assignments [38]. Blocks representing one day are wrapped by weeks (of month). Inside a block hours are displayed as rows, where columns are five-minute-intervals. In this example on the fourth Saturday the data is missing.

2.4.5 GROOVE

GROOVE [38] (**Gr**anular **O**verview **Ove**rlay) is an interactive pixel-based InfoVis method using different time granularities (e.g. *Year*, *Week of Month*) to provide different representations time-oriented data. This visualization is configured with four time granularities, which then are rendered as a recursive grid layout (see Figure 2.23). Besides giving an overview about the data, GROOVE also provides detail information due to three possible overlays:

- **Color-based Overlay:** The value of each pixel has a special color (e.g. blue: low, red: high)
- **Opacity Overlay:** The average values per block can be displayed based on a blocks lightness (e.g. bright: low, dark: high)
- **Spatial Overlay:** Additionally users can interactively expand or collapse different levels of time granularities.

2.5 Conclusion

Most of our evaluated data quality tools cover techniques to profile or wrangle data with missing support of complex time-oriented constraints. Only four out of nine tools provide distribution visualizations aggregated by a specific time granularity. DataCleaner provides an time-oriented visualization showing time gaps between specific data entries.

Of all the evaluated tools, TimeCleanser is the only one supporting annotated data entries. It provides annotation of data entries based on time-oriented constraints and offers visualizations for durations and value changes over time. This prototype, however, mainly provides time-oriented data quality checks, making it difficult to discover further, general data quality problems.

CHAPTER 3

Design

3.1 Requirement Analysis

To come up with an appropriate concept we first had to identify the requirements of the prototype. *Grooviler* is planned to be a part of a workflow-based framework to cleanse and profile time-oriented data quality problems. In a previous step a prototypical application named *QualityTime* [39] was developed to detect and annotate data quality problems with the help of (semi-)automatic cleansing and profiling techniques. Based on the resulting data, *Grooviler* should be used in the next step to visualize the annotated data with a focus on time-oriented data quality problems. It is also planned to use it for identifying further data quality problems with the help of interactive visualizations.

The resulting prototype of *Grooviler* should therefore be capable of covering/profiling as many data quality problems as possible (R1) in addition to displaying already annotated data quality problems in an adequate way (R2). The prototype's concept solely concentrates on Data Profiling and therefore should not provide possibilities to wrangle or transform the underlying data. As well as *QualityTime*, *Grooviler* uses the taxonomy of time-oriented single source data quality problems by Gschwandtner et al. [12] as a basis for its requirements (R3).

To provide an overview about the underlying data and its scope, we want to provide simple statistics like minimum, maximum or mean of different columns (R4). Like most of our evaluated Data Profiling tools, we also want to provide histograms showing distribution of certain values.

Requirements

R1 **Display annotated data:** display already annotated data quality problems.
R2 **Data Profiling:** provide mechanisms to profile further data quality problems.
R3 **Data Quality Problem Coverage:** cover time-oriented single source data quality problems by Gschwandtner et al. [12] (if applicable).
R4 **Statistics:** display statistics about the data set or parts of it.

3.2 Design Models

3.2.1 Nested Model for Visualization Design

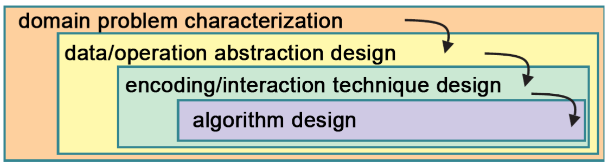
To ease the design and analysis of our prototype, we use the nested model for visualization and validation of Munzner [16]. She defines four levels of visualization creation, of which each one contains threats of an unsuitable design (see Figure 3.1 (a)). Each level results into the next level, with separate problems and risks.

On the first level, which is the characterization of domain, users and data have to be specified. It contains the risk that visualizations are designed for the wrong people or domain.

The second level is the abstraction of the domain to specific operations and data types. This is the basis for interaction techniques in the next level. The threat in this level is that chosen operations do not fulfill the needs of users.

The third level is about visual encoding holding the risk that the design does not communicate the abstraction to the user effectively. The innermost level is about the performance of implemented algorithms.

Figure 3.1 (b) shows approaches to validate the design of Grooviler. To define domain, data and operation characteristics of the first two levels, we will use the data-users-tasks triangle by Miksch et al. [1] (see Section 3.2.2). To validate our encoding/interaction design, we conducted iterative feedback cycles until we were satisfied with the result of our design (see Section 3.3). During implementation, we ensured the correctness or our algorithms with the help of built-in visualizations of Microsoft Excel [40].


domain problem characterization
data/operation abstraction design
encoding/interaction technique design
algorithm design
(a) Model of visualization creation by Munzner [16].

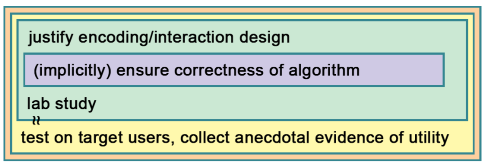

justify encoding/interaction design
(implicitly) ensure correctness of algorithm
lab study
test on target users, collect anecdotal evidence of utility
(b) Grooviler validation methods.

Figure 3.1: Grooviler validation methods by Munzner [16].

Munzner [16] diferentiates between lab studies measuring human errors for operation and tests on target users. While lab studies reveal usability problems (covering the encoding/interaction level), tests on target users provide the evidence of a systems utility (covering the abstraction level). In this work, we mixed both approaches, conducting an evaluation with target test users in a laboratory setting containing tasks. This delivers both, usability issues, and feedback, as well as the usefulness of our prototype measured with quantitative amounts of insights (see Chapter 5).

3.2.2 Data-Users-Tasks Design Triangle

For designing an appropriate, effective, and expressive VA approach, we need to consider the characteristics of data, users and tasks [1]. Only after addressing those three main aspects, we can think of adequate interactive visualizations (see Figure 3.2). Especially application designs that handle the complexity of time, need to take these considerations into account. Satisfying those three quality criteria results in effective, expressive, and appropriate VA methods.

Data

Grooviler should cover as many data types as possible. We do not plan on restricting any data type. In particular, we want to cover numerical, textual and time data types. Our prototype supports only tabular data.

Aigner et al. [33] summarizes design aspects of time-oriented data, which will now be discussed. We plan on using a discrete scale of time, so that it can be converted to a quantifiable unit (e.g., seconds). Date and time granularities should be offered as users are aware of the Gregorian calendar system (e.g., year, month, day). We handle intervals only as a span (duration) without anchored information about the beginning or end.

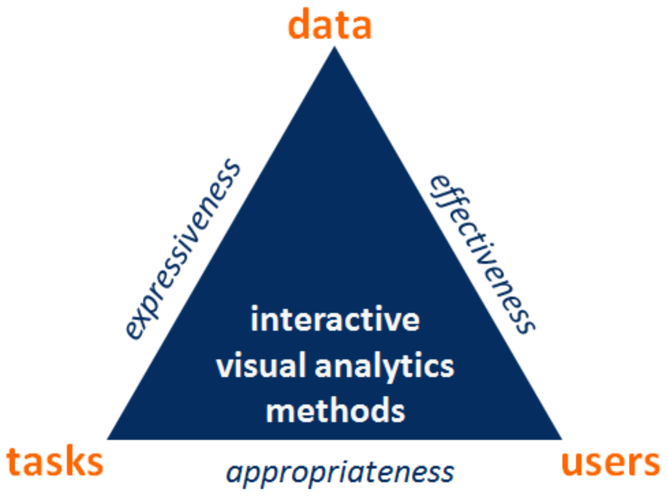


Figure 3.2: Data-Users-Tasks Design Triangle by Miksch et al [1].

Users

Our prototype should be used by any application domain. We define our target users as domain expert and data analyts, having a at least a minimal amount of technical experience in analysis tools. We also expect our users to have large displays (e.g., > 25" screen resolution) and computers containing a strong hardware (e.g., at least 8 GB RAM, Quad-Core CPU, SSD).

Tasks

General tasks were already covered in our requirement analysis. Users want to inspect already annotated data, as well as identifying further data quality problems. They also want to gain an understanding about the structure and content of underlying data. Specific user tasks and derived user interactions will be defined during the design process (see Section 3.3.2).

3.3 Design Process

First, we performed a brainstorming based on a broad range of visualization types with the help of the TimeViz Browser [21]. This is a scientific collection of different visualization techniques concentrating on time-oriented data. Then we assembled different ideas and approaches into a single InfoVis concept. The next step was to create a GUI design for the prototype. At last, we compared our approach with existing techniques found in our research.

3.3.1 Brainstorming

In addition to the Data Profiling solutions found in the related work, we took a step back and analyzed different visualization techniques. With the help of the TimeViz Browser [21], a collection of more than hundred scientific visualization techniques, we made a quick review about different visualizations whether they can be used for our time-oriented data quality problems or not. For this, every visualization technique was printed onto a small paper to get some inspiration for appropriate visualizations (see Figure 3.3).

The results of the first brainstorming iteration were 5 different approaches to display annotated dirty data and 8 techniques to discover further data quality problems.

Approaches to display annotated dirty data:

- **Stacked Bar Chart over Time:**
  This visualization stacks annotated data quality problems in groups on a bar chart which horizontal axis is ordered by a specific time column (see Figure 3.4).

Figure 3.3: Printed visualization techniques from the TimeViz Browser to do a rough evaluation and pre-selection.

- **Data Quality Tile Map:**
  A Tile Map [35] could be used to display the amount of data quality problems (e.g., missing values) expressed by its opacity (see Figure 2.20). Therefore users might discover seasonal causes for data quality problems (e.g., identifying a higher amount of dirty data each Monday because of periodic maintenance work). Users then could interactively set specific data quality annotations which are then displayed.

- **Annotation ThemeRiver:**
  Using a ThemeRiver [34] visualization (see Figure 2.19) would be an option to show the trend of data quality problems over time. Instead of themes the prototype could use different data quality problem types.

- **Bar Chart grouped by Date Granularity:**
  Annotated data could be grouped by a specific date granularity of a specific column. As an example dirty data could be grouped by hour intervals (*[00:00 - 01:00[, [01:00 - 02:00[, ...*) and be visualized over time in a bar chart. Stacked granularity

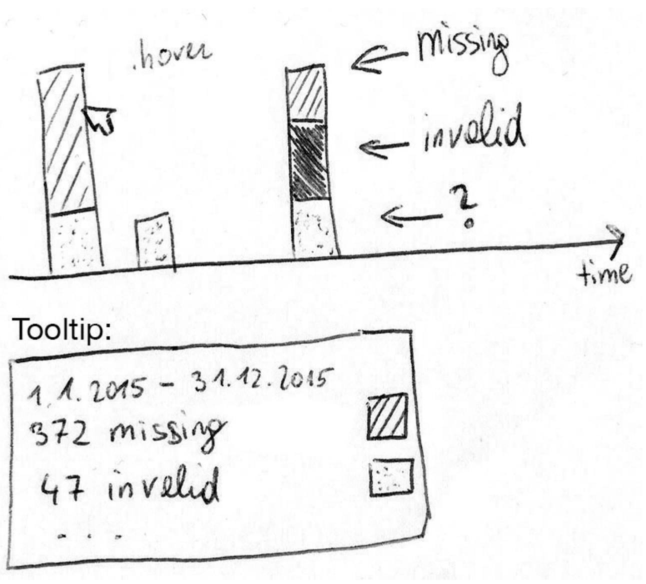


Figure 3.4: Stacked bar chart which stacks the amount of data quality problems in a bar shown over time.

grouping would also be possible, e.g. *Week Day + Hours* (*[Mo: 00:00 - 01:00[, ... [Fr: 23:00 - 23:59]*).

- **Annotation Spiral Graph:**
An interactive Spiral Graph [37] (see Figure 2.22) rendering the occurrence count of a specific data quality problem would show dirty data over time which occur over several cycles. A combination of multiple views with spiral graphs showing different data quality annotations could help to make a connection between data and its dirtiness over time in cyclic manner.

Approaches to find further data quality problems:

- **Duplicate detection:**
The prototype could provide a configuration page for users to set up aspects of data which define them as unique (e.g., ID). With this technique corresponding data entries would be identified and annotated as duplicates. These results then may be visualized in different date or time granularities (e.g., duplicates per year, duplicates per month, ...).

  This idea was dropped rather quickly since QualityTime [39] already supports elimination of duplicates.

- **Configurable Quantity Violation:**
Another idea was to provide a modular configuration for the users for time-oriented

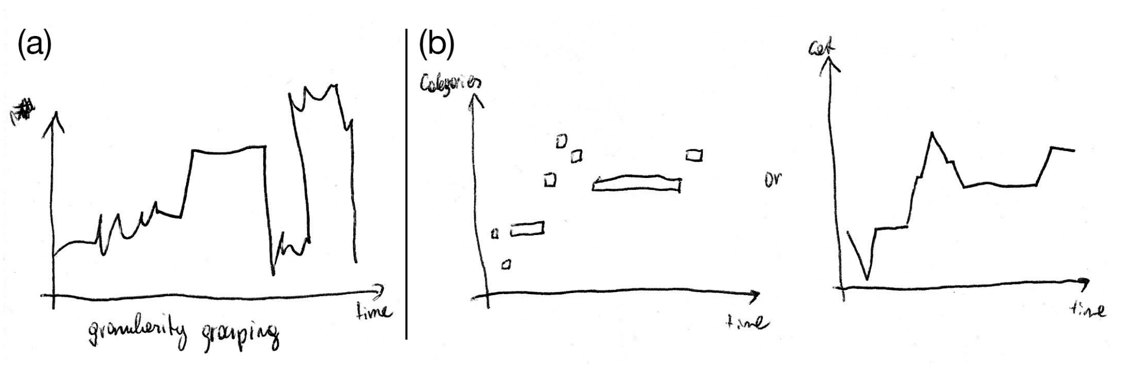


Figure 3.5: Sketch of a line chart displaying value changes over time (similar to TimeCleanser's visualization [22]) with (a) quantitative values and (b) two possible visualizations for nominal value changes over time.

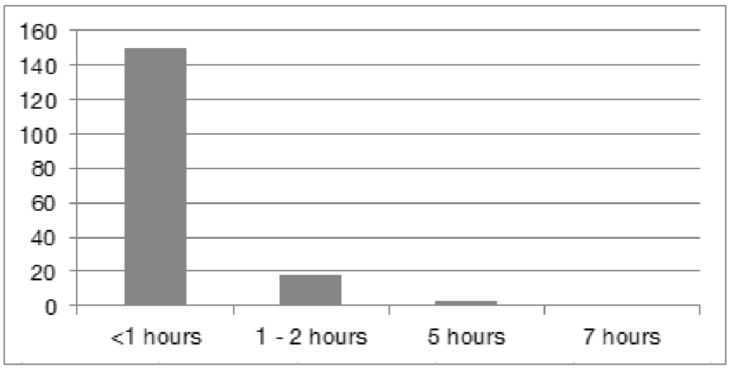


Figure 3.6: Sketch of a bar chart showing the distribution of duration interval length.

constraints. Setting cross-dependent constraints like a number which must be in a given range for a given time, would then be annotated as invalid (e.g., between 00:00 and 08:00 the number of customers must be 0).

- **Date Gap/Overlap Analyzer:**
  Providing a date gap analyzer as it is used in DataCleaner [24] (see Figure 2.16) could help to find potential data quality problems based on overlappings or gaps in time-oriented data.

- **Value Changes over Time:**
  TimeCleanser's [22] visualization for value changes over time (see Figure 2.18) can help to find unexpected value changes over time, which could also be adapted to nominal data (see Figure 3.5 (b)).

- **Duration Heatmap:**
  TimeCleanser's duration heatmap (see Figure 2.17) is an example of a technique helping users to visually identify implausible or wrong data.

- **Duration Length Distribution Bar Chart:**
  Interval lengths (durations) could also be displayed as a distribution with the help of a bar chart (see Figure 3.6). This bar chart would provide interactive possibilities like details on demand by showing information about corresponding data and brushing and linking.

- **Cycle Plot:**
  Although Cycle Plots [36] generally were intended to discern seasonal and trend components, they could be used to visualize trends of dirty data. This technique could also be used to identify seasonal patterns on data quality problems (e.g., missing values have an upward trend on Mondays). As we can see combining different temporal granularities can result in remarkable visualization results.

- **GROOVE:**
  Rendering different stacked time granularities in one visualization like in GROOVE [38] is another interesting approach. This type of visualization could help to find additional data quality problems, like missing values in a seasonal pattern (see Figure 2.23). It could also help to visualize annotated data quality problems by mapping a pixels opacity to the amount of a specific data quality problem.

Additionally the first brainstorming iteration resulted in notes about possible features of the prototype (where applicable):

- Display the data table containing the profiled data
- Integrate brushing and linking between visualizations and the data table
- Support temporal granularities
- Offer interactive switching of a temporal granularities
- Provide interactive setting of cycle lengths
- Integrate multiple views

3.3.2 Idea refinement

At this point, within the scope of this work, we decided to solely concentrate on building one and up to a maximum of two interactive visualizations. Hence the next step was to decide which type of visualization would fit best covering as many data quality problems as possible.

Looking at the results of the first brainstorming iteration (see Section 3.3.1), it seems that two-dimensional visualization approaches fit our previously defined requirements. They can be used to (1) display already annotated data (e.g., Data Quality Tile Map [35]) as well as (2) help the user identifying further data quality problems (e.g., Cycle Plot [36], Duration Heatmap [22] GROOVE [38]). Therefore we decided to use a heatmap visualization similar to a tile map, which axes can be freely configured.

Temporal Granularity			
Granularity	Minimum	Maximum	Next Zoom Granularity
Year	-	-	Month
Month	0	11	Day of Month
Week of Year	1	53	Day of Week
Week of Month	1	5	Day of Week
Day of Year	1	365	Hour of Day
Day of Month	1	31	Hour of Day
Day of Week	1	7	Hour of Day
Hour of Day	0	23	Minute
Minute	0	59	Second
Second	0	59	-
Millisecond	*0*	*999*	-

Table 3.1: Temporal Granularities provided by JAVA Development Kit from version 1.5 to 1.7 including their value range returned by the Calendar API. Milliseconds are not supported by the *Prefuse Toolkit.* The last column covers the next logical granularity if users want to zoom inside this granularity.

Axes

The axes of our heatmap visualization present specific columns of the data set, which can either hold data that is quantitative (e.g., sale, price), nominal (e.g., category) or temporal (e.g., birth date, time of purchase). Users should have the ability to interactively change and combine axes based on a data set's column and its data type. As an example, users might configure the row-axis with nominal data like *First Name* and the column-axis with temporal data like *Birth Date* grouped by years. This could show anomalies or trends of first names over time.

Quantitative axes have a linear range of a column's minimum to its maximum value. Temperatures might reach between -50 and 50° Celsius whereas prices may range between 0 and 1.000 €.

Nominal axes render categorical data of a specific column sorted in alphabetical order. The categories are taken from the columns containing textual data (String representation). The amount of categories depends on the amount of different nominal values.

Temporal axes based on a certain column containing date, time, or both, display data in a specific temporal granularity. The different types of temporal granularities are shown in Table 3.1. In our concept years are the highest granularity, therefore the range of years has to be calculated by the minimum and maximum year of the underlying data. Other granularities are constrained by one of their superior time granularities (e.g., a year consists of twelve months, a week consists of seven days).

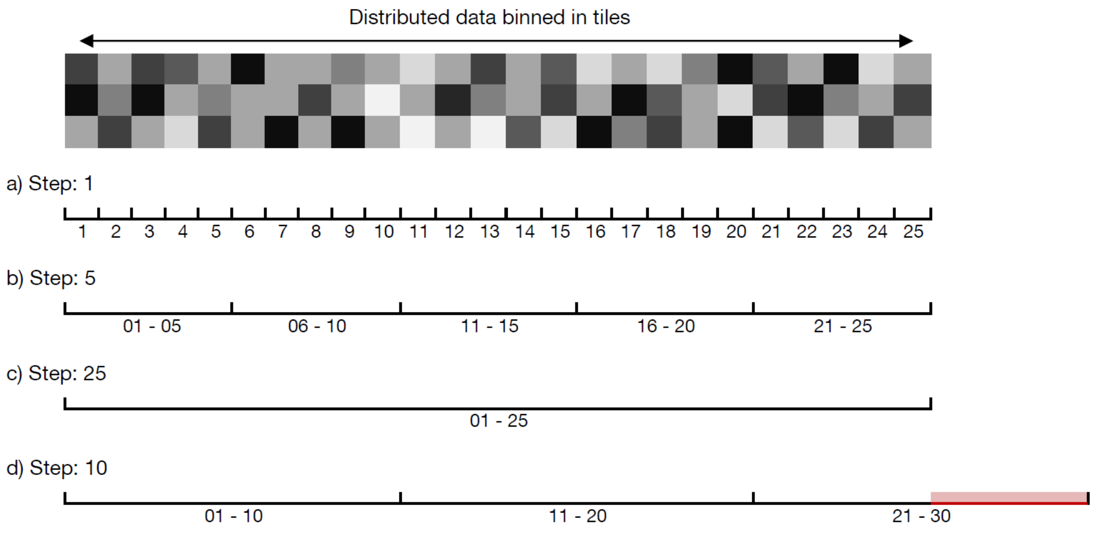


Figure 3.7: Example of different step-widths with a quantitative axis containing integer values from 1 to 25. With a step-width of 1, which is the most fine-grained step-width for integers, no binning has to be done (a). With a setting of a step-width like 5, or 25, which are divisible by the maximum range of 25 without a remainder, tiles are evenly distributed (b-c). Using an indivisible step-width like 10, results in a tile at the last position which ranges up to 30 although this tile only holds integers up to 25 (d).

Binning

Approaches using temporal axes, like GROOVE or Tile Maps, use the same date objects with different temporal granularities for column and row representation. The mapping of cells (tiles) or pixels with opacity or coloring is based on the data that falls in the corresponding time period. With our modular approach, letting users decide how to configure the heatmap's axes, our axis layout has to look up for the corresponding data type, based on the current axis configuration. Thus, binning depends on the respective data type of the current axis.

Nominal data can easily be grouped into different categories which then are evenly distributed along the axis. This also applies to temporal data grouped by a temporal granularity. However quantitative or numerical data, in particular floating point data, have no natural non-decomposable unit on which the axis can rely on.

Therefore quantitative axes need a so called "step-width" which defines the length of intervals used for grouping corresponding data records. Figure 3.7 shows a simple example with a quantitative axis containing integer values from 1 to 25 and different step-width configurations and its drawbacks. Using an indivisible step-width results in partly filled tiles, distorting the users perception while visually profiling the data.

We may now decide to only allow step-widths that are divisible without a remainder, but we want to keep the prototype as unrestricted in configuration as possible. Nevertheless, we plan on calculating a suitable step-width for quantitative data, that the user can interactively change.

As we can see in Table 3.1 temporal granularities also act like ordered nominal data due to their interval ranges. Therefore we could also provide a step-width to set interval ranges in temporal granularities. This could be a step-width of 12 with *Hour of Day* to differentiate forenoon and afternoon, or a step-width of 5 with *Minute*s. A step-width in temporal granularities could be described as “each x granularities” (e.g., each 12 hours, each 5 minutes).

In our prototype, step-widths must be within a temporal granularity’s interval range which can be found in Table 3.1. Although granularity configurations like 50 hours or 100 minutes may be useful in special situations, we restrict this configuration based on the calendar’s structure.

Using step-widths in temporal axes with granularities, provides another different, aggregated overview of the data. We want to offer a quick and responsive setting for step-widths so that users get immediate feedback of the impact of aggregating data and its changes. This could be accomplished with a slider component.

Cell Color Coding and Calculation

Figuratively speaking, with defining axes we offered different ways to change the width and height of tiles in our heatmap. Using quantitative data with scope of more than 10.000 and a step-width of 1 will result in a pixel-based visualization. Whereas nominal data with 5 different categories will result in a heatmap containing 5 big tiles.

The next step is to define a mapping between data and color. A tile of the heatmap contains a subset (list of tuples) of the data which falls into the boundaries given by the axes. We want to offer three different ways to generate a color based on the tuples of a tile.

- **Tuple Count:**
  The most simple coloration is setting the color based on the amount of tuples contained in a tile. This can help to give an overview about occurrences of tuples and their distribution.

- **Aggregate Tuples:**
  Another option is to set a tile’s color based on a single value which is grouped by an aggregate function. A specific tile could hold a subset of sales transactions on Mondays between 9 and 10 o’clock. We may now want to know the sum of sales during this period. Therefore we would aggregate the column “Sales” with the aggregate function *Sum*. This form of coloration could help to identify implausible values.

Therefore, we provide the following aggregate functions:

  - Count
  - Distinct Count
  - Median
  - Sum *
  - Mean *
  - Standard Deviation *
  - Minimum *
  - Maximum *

  Functions marked with an asterisk * can only be applied to numerical data. Count, Distinct Count, and Median can also be used for ordinal and temporal data.

- **Annotation Count:**
  To visualize data entries that were annotated with identified quality problems in a previous step, we want to provide a coloring based on the amount of tuples annotated with data quality problems. Users then could choose only to visualize an individual or any type of quality problem. With this, users can inspect the distribution of data quality problems across the data set.

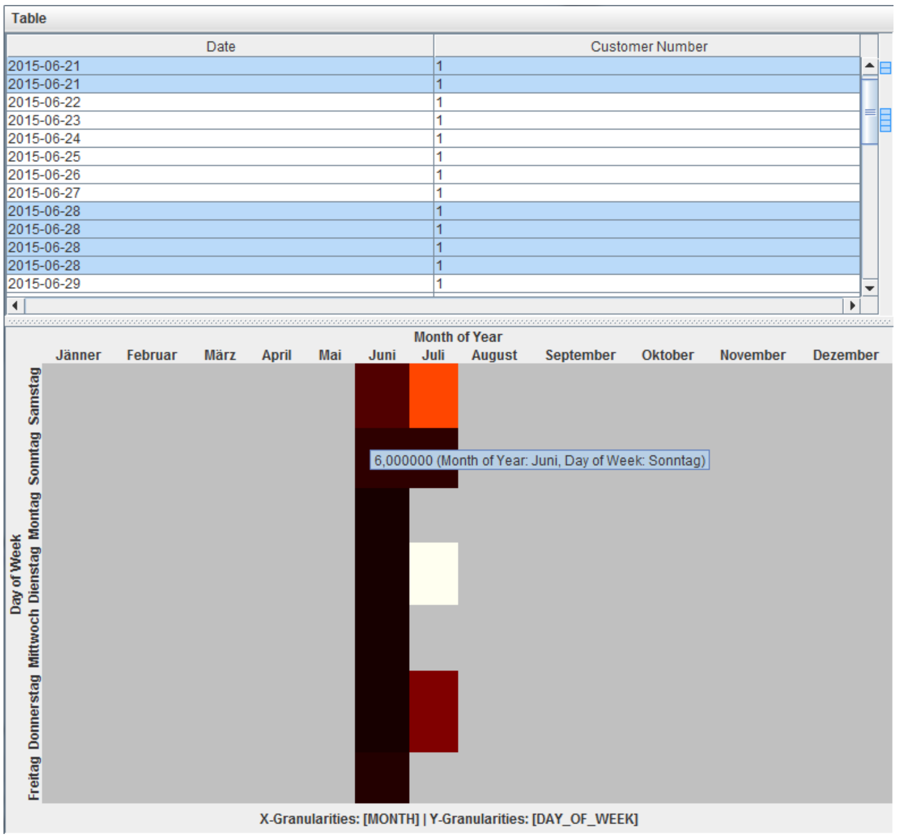
We might use several types of overlays as it was implemented in GROOVE. In our case, however, we decided to only use a single color coding to not overtax users using the prototype.

Interaction Operations

At this point, users can interactively configure horizontal and vertical axes based on a certain column's value, data type and step-width (if ordinal). They can also configure the type of aggregate functions used for heatmap cells (tuple count, tuple aggregation, or annotation count). To identify the scope of *Grooviler's* interaction operations, we decided to create a small proof of concept prototype (see Figure 3.8).

It is build with JAVA without any technical constraints as a disposable application. It consists of a data table implemented with a `JTable` and a heatmap arranged with a `MigLayout` as a grid layout. MiG Layout [41] is an open source JAVA layout manager. We could have used a `java.awt.GridLayout` which is included in JAVA Development Kit since version 1.0. However, its implementation can only deal with integer values, resulting in gaps between tiles when the heatmap's width is indivisible.

Thanks to a `JSplitPane`, the prototype already supports flexible panel dimensions of the table or heatmap panel. With this, the movable divider can be used to setup the desired height of the corresponding panels.



Table

Date	Customer Number
2015-06-21	1
2015-06-21	1
2015-06-22	1
2015-06-23	1
2015-06-24	1
2015-06-25	1
2015-06-26	1
2015-06-27	1
2015-06-28	1
2015-06-28	1
2015-06-28	1
2015-06-28	1
2015-06-29	1

6,000000 (Month of Year: Juni, Day of Week: Sonntag)

X-Granularities: [MONTH] | Y-Granularities: [DAY_OF_WEEK]

Figure 3.8: Screenshot of the first prototype serving as a proof of concept identifying possible interaction operations. Selecting a heatmap cell highlights the corresponding data table rows with interactive markers (known as TODO-markers from *Eclipse IDE* [42]) Hovering over a heatmap cell results in a tooltip providing information about the cell's calculated value and temporal granularities.

The disposable prototype is able to create a heatmap based on programmatically defined temporal granularities, which are shown in a status bar at the bottom of the screen. Hovering a tile results in a tooltip showing the amount of containing table entries. By clicking on a tile, the corresponding table rows get highlighted and annotated with markers. This markers are a starting point for *Grooviler* since they are not yet interactive. Otherwise, this first prototype does not provide any interaction operations. Also, the heatmap configuration view was not defined and implemented yet.

With the help of this proof of concept we refined the provided interaction operations. We additionally implemented the Visual Information Seeking Mantra by Shneiderman [43]:

1. *Overview:* This is already supported by the heatmap's appearance due to its axes, data types, and used values.

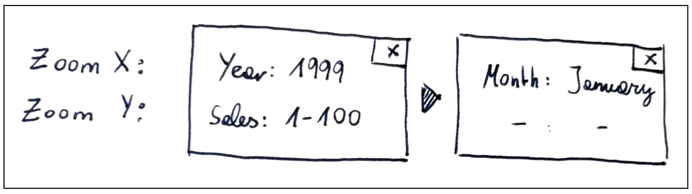


Figure 3.9: Sketch of a Zoom History. Each zoom action gets stored and displayed as an interactive element to the right of the prior zoom action. Each action can be undone by clicking the close-button in the right corner of each element.

2. *Zoom & Filter:* We accomplish this by offering a technique to zoom inside a heatmap cell. With numerical data this filters the cell's numerical interval range. Zooming inside temporal data switches to the next finest temporal granularity based on the parents granularity. As an example zooming inside a heatmap tile containing the year 1999 would display Months of the year 1999.

3. *Details-on-Demand:* We accomplish this by providing a separate view displaying statistics (e.g., box plots, value distribution) about a user selection.

4. *Relate:* Existing tools do not provide the possibility to identify corresponding data entries of a specific data quality problem. We bridge this gap with the help of interactive markers next to the data table's scroll bar (similar to TODO-markers implemented in *Eclipse IDE* [42]). When users select a specific heatmap tile, related table rows get highlighted and annotated with interactive markers. Clicking the marker causes the table to scroll automatically to the clicked table element.

5. *History:* We provide a simple history containing each zoom action (see Figure 3.9). With this, each zoom level (and filter operation) can be undone.

6. *Extract:* The extraction of sub-collections or query parameters is not supported by this prototype.

3.3.3 GUI Design

Now that we defined possible user interaction operations, we were able to design an appropriate GUI. We identified 4 areas which are needed to offer already discussed functionality: We need (1) a data table containing all data, (2) a heatmap visualization, (3) a panel containing statistic visualizations and (4) a section providing configuration elements for heatmap configuration.

The first attempt was using 4 different views arranged in a grid layout (see Figure 3.10). This GUI design has weaknesses using space inefficiently. Data tables often need a certain width due to their cell's contents. This layout would result in limiting the remaining space for our heatmap visualization. Another flaw with this approach is, that statistic

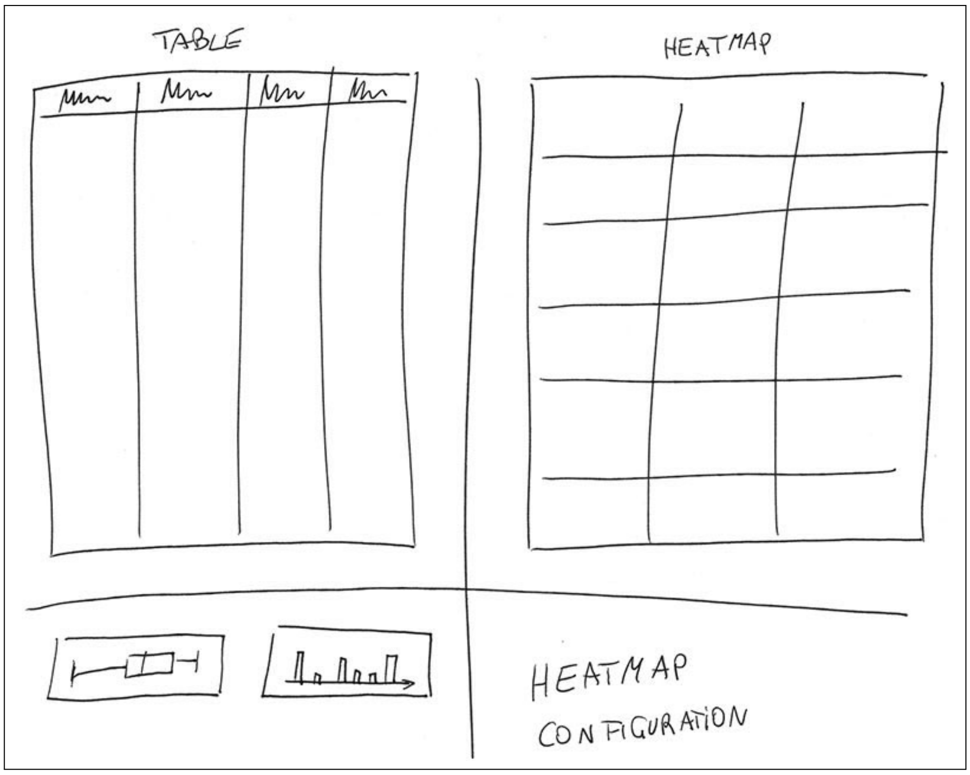


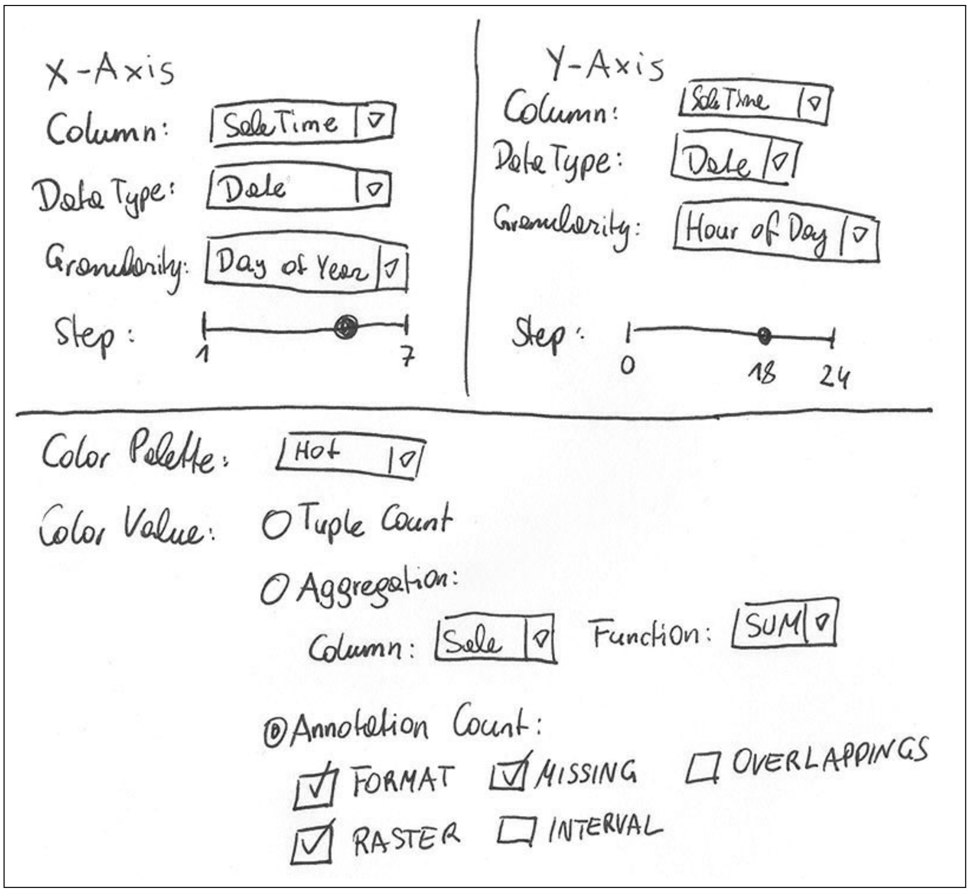
Figure 3.10: Sketch of the first GUI approach, showing a data table (upper left), heatmap visualization (upper right), statistic visualizations for selection (lower left), and heatmap configuration (lower right, see Figure 3.11).

visualizations would also need to be arranged as a grid layout. This visualizations are then limited in their size, especially in their height.

Also, the configuration panel for the heatmap (see Figure 3.10 lower right, and Figure 3.11) would need a lot of space. Therefore the statistics panel automatically adopts that height. When users have no active selection this would result in a large, empty area.

To address this problems we reorganized the layout (see Figure 3.12). First of all we moved the statistics panel on to the side into a separate panel containing a vertical scroll bar next to the main panel. We assume that users do not need to constantly reconfigure the heatmap visualization. Therefore we extracted the heatmap configuration view into an own separate window, since it takes up a significant amount of space. With this, it is up to the users to decide when and how to display the configuration window. This may be on top or besides the main window, or even on another monitor.

A panel containing the zoom history (see Figure 3.9) was missing in the first design. It is now arranged below the heatmap at the bottom of the main window. Also, we arranged the heatmap below the data table to give it enough width.


X-Axis

Column: SaleTime ▽

Date Type: Date ▽

Granularity: Day of Year ▽

Step: 1 ● 7

Y-Axis

Column: SaleTime ▽

Date Type: Date ▽

Granularity: Hour of Day ▽

Step: 0 ● 18 24

Color Palette: Hot ▽

Color Value: ○ Tuple Count

○ Aggregation:

Column: Sale ▽ Function: SUM ▽

◉ Annotation Count:

☑ FORMAT ☑ MISSING ☐ OVERLAPPINGS

☑ RASTER ☐ INTERVAL

Figure 3.11: GUI Sketch of the heatmap configuration containing configuration of the horizontal axis (upper left), configuration of the vertical axis (upper right), and configuration of the calculated/colored value (bottom).

To provide enough flexibility we provide adjustable panel dimensions. Users should be able to customize the width of the statistics panel, as well as the height of the data table and heatmap. Therefore we want to use movable dividers between the main panel and statistics panel, and between the data table and heatmap.

3.3.4 Feature Comparison

Before starting the implementation we compared our concept with the techniques found in our research about Data Profiling tools (see Table 2.4). The following list shows additional features offered by our current approach:

- *Value Distribution/Histogram:* by showing a boxplot about a selected subset of the data.

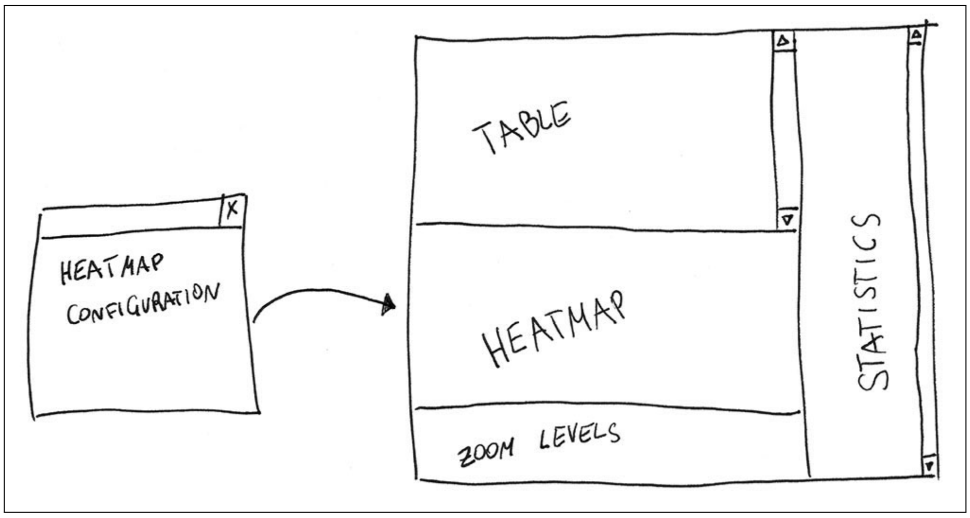


Figure 3.12: Sketch of the second and final GUI approach. The heatmap configuration is extracted into a separate window (left) separated from the main window (right). The statistics panel is now attached at the right side of the window. The main panel consists of the data table, heatmap and the zoom history.

- *Pattern Matching:* because *QualityTime* supports regular expression matching which annotates matching data.
- *Date Gaps:* using a separate column containing calculated date gaps which then can be visualized.
- *Date Distribution:* due to temporal granularity settings of each axis of the matrix.
- *Period Length:* by visualizing manually, pre-calculated lengths of intervals (e.g., in milliseconds).

CHAPTER 4

Grooviler

This chapter covers the implementation of the prototype, and gives an overview about project dependencies. After discussing architecture decisions we then cover each implementation of every visualization.

4.1 Technical Details and Implementation

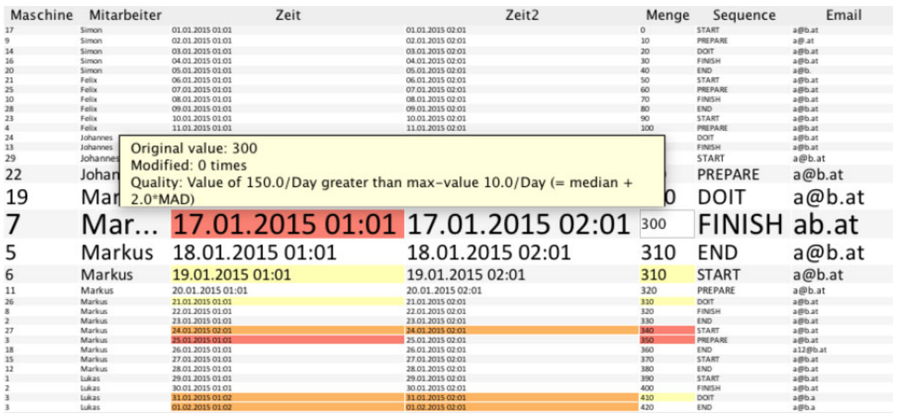
We will now give an overview about technical dependencies of *Grooviler*. Then we will cover the architecture of our prototype and explain each implemented visualization.

4.1.1 QualityTime Dependency

As already mentioned, *Grooviler* needs to work with a data set where erroneous data entries are already annotated. This is accomplished with another prototype called *QualityTime* [39] developed by Thomas Braunsberger at the *Vienna University of Technology*. It extends a prototype named *TimeProfiler* developed by Theresia Gschwandtner and Benjamin Klaus.

QualityTime offers the possibility to import given data in CSV (comma separated values) data format. It automatically identifies basic data quality problems on runtime (e.g., missing values). Contextual data quality issues can be found with specific configurable checks (e.g., identifying outliers, checking time raster). Faulty data is then annotated and highlighted in the provided fisheye table (see Figure 4.1).

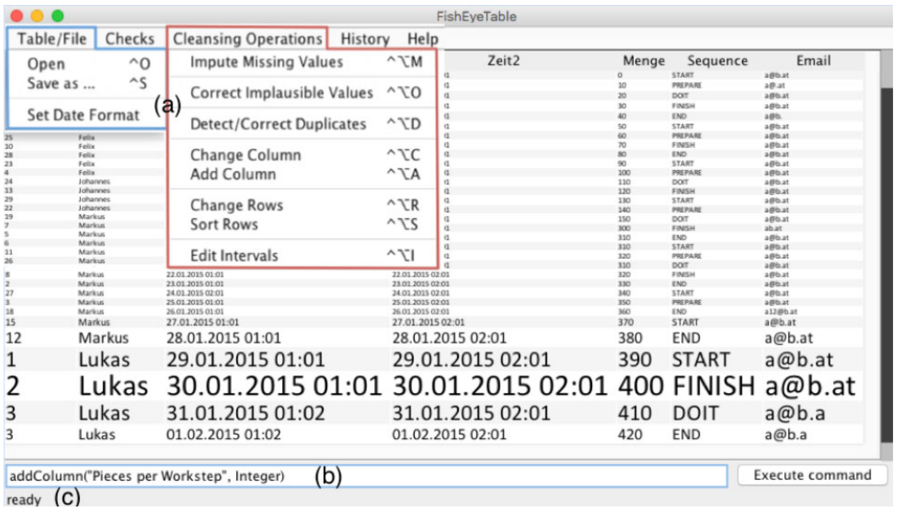
Also, this tool provides possibilities to wrangle data with the help of semi-automatic cleansing algorithms (see Figure 4.2). It also provides data transformation operations, i.e., adding columns, and performing calculations on different column values. Each step of the user is recorded and can therefore be undone.



Maschine	Mitarbeiter	Zeit	Zeit2	Menge	Sequence	Email
17	Simon	01.01.2015 01:01	01.01.2015 02:01	0	START	a@b.at
9	Simon	02.01.2015 01:01	02.01.2015 02:01	10	PREPARE	a@.at
14	Simon	03.01.2015 01:01	03.01.2015 02:01	20	DOIT	a@b.at
16	Simon	04.01.2015 01:01	04.01.2015 02:01	30	FINISH	a@b.at
20	Simon	05.01.2015 01:01	05.01.2015 02:01	40	END	a@b.
21	Felix	06.01.2015 01:01	06.01.2015 02:01	50	START	a@b.at
25	Felix	07.01.2015 01:01	07.01.2015 02:01	60	PREPARE	a@b.at
10	Felix	08.01.2015 01:01	08.01.2015 02:01	70	FINISH	a@b.at
28	Felix	09.01.2015 01:01	09.01.2015 02:01	80	END	a@b.at
23	Felix	10.01.2015 01:01	10.01.2015 02:01	90	START	a@b.at
4	Felix	11.01.2015 01:01	11.01.2015 02:01	100	PREPARE	a@b.at
24	Johannes				DOIT	a@b.at
13	Johannes				FINISH	a@b.at
29	Johannes				START	a@b.at
22	Johan				PREPARE	a@b.at
19	Mar			0	DOIT	a@b.at
7	Mar...	17.01.2015 01:01	17.01.2015 02:01	300	FINISH	ab.at
5	Markus	18.01.2015 01:01	18.01.2015 02:01	310	END	a@b.at
6	Markus	19.01.2015 01:01	19.01.2015 02:01	310	START	a@b.at
11	Markus	20.01.2015 01:01	20.01.2015 02:01	320	PREPARE	a@b.at
26	Markus	21.01.2015 01:01	21.01.2015 02:01	310	DOIT	a@b.at
8	Markus	22.01.2015 01:01	22.01.2015 02:01	320	FINISH	a@b.at
2	Markus	23.01.2015 01:01	23.01.2015 02:01	330	END	a@b.at
27	Markus	24.01.2015 02:01	24.01.2015 02:01	340	START	a@b.at
3	Markus	25.01.2015 01:01	25.01.2015 02:01	350	PREPARE	a@b.at
18	Markus	26.01.2015 01:01	26.01.2015 02:01	360	END	a12@b.at
15	Markus	27.01.2015 01:01	27.01.2015 02:01	370	START	a@b.at
12	Markus	28.01.2015 01:01	28.01.2015 02:01	380	END	a@b.at
1	Lukas	29.01.2015 01:01	29.01.2015 02:01	390	START	a@b.at
2	Lukas	30.01.2015 01:01	30.01.2015 02:01	400	FINISH	a@b.at
3	Lukas	31.01.2015 01:02	31.01.2015 02:01	410	DOIT	a@b.a
3	Lukas	01.02.2015 01:02	01.02.2015 02:01	420	END	a@b.a

Original value: 300
Modified: 0 times
Quality: Value of 150.0/Day greater than max-value 10.0/Day (= median + 2.0*MAD)

Figure 4.1: Example of annotated data quality problems in *QualityTime* [39]. The hovered row and its neighbors are displayed bigger than the rest due to its representation as a fisheye table. In this screenshot various data quality issues are highlighted with different background colors. Hovering a cell provides a detail on demand tooltip with a short description of the specific problem.



			Zeit2	Menge	Sequence	Email
				0	START	a@b.at
				10	PREPARE	a@.at
				20	DOIT	a@b.at
				30	FINISH	a@b.at
				40	END	a@b.
				50	START	a@b.at
25	Felix			60	PREPARE	a@b.at
10	Felix			70	FINISH	a@b.at
28	Felix			80	END	a@b.at
23	Felix			90	START	a@b.at
4	Felix			100	PREPARE	a@b.at
24	Johannes			110	DOIT	a@b.at
13	Johannes			120	FINISH	a@b.at
29	Johannes			130	START	a@b.at
22	Johannes			140	PREPARE	a@b.at
19	Markus			150	DOIT	a@b.at
7	Markus			300	FINISH	ab.at
5	Markus			310	END	a@b.at
6	Markus			310	START	a@b.at
11	Markus			320	PREPARE	a@b.at
26	Markus			310	DOIT	a@b.at
8	Markus	22.01.2015 01:01	22.01.2015 02:01	320	FINISH	a@b.at
2	Markus	23.01.2015 01:01	23.01.2015 02:01	330	END	a@b.at
27	Markus	24.01.2015 02:01	24.01.2015 02:01	340	START	a@b.at
3	Markus	25.01.2015 01:01	25.01.2015 02:01	350	PREPARE	a@b.at
18	Markus	26.01.2015 01:01	26.01.2015 02:01	360	END	a12@b.at
15	Markus	27.01.2015 01:01	27.01.2015 02:01	370	START	a@b.at
12	Markus	28.01.2015 01:01	28.01.2015 02:01	380	END	a@b.at
1	Lukas	29.01.2015 01:01	29.01.2015 02:01	390	START	a@b.at
2	Lukas	30.01.2015 01:01	30.01.2015 02:01	400	FINISH	a@b.at
3	Lukas	31.01.2015 01:02	31.01.2015 02:01	410	DOIT	a@b.a
3	Lukas	01.02.2015 01:02	01.02.2015 02:01	420	END	a@b.a

addColumn("Pieces per Workstep", Integer)

ready

Figure 4.2: Structure of *QualityTime*'s UI: (a) Menu bar to perform quality checks to annotate data or data transformations, (b) command line and (c) status bar [39].

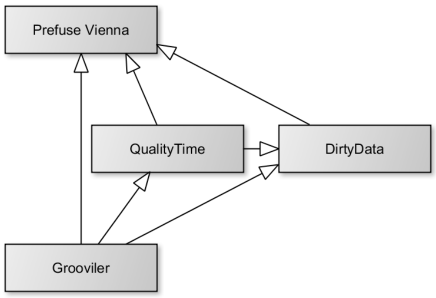


Figure 4.3: Project Dependencies of Grooviler: *QualityTime* and *Grooviler* both depend on *Prefuse Vienna* and *DirtyData. Prefuse Vienna* is a slightly modified version of *Prefuse* [17]. *DirtyData* is an extension of *Prefuse* providing a dirty data model, time quality checks and wrangling operations.

At the start of the implementation phase *QualityTime* had not been fully implemented. To avoid complicated merge problems, *Grooviler* is being developed in a separate project using the same project dependencies as *QualityTime* (see Figure 4.3). *Prefuse Vienna* is a slightly modified version of Prefuse [17] which is continuously extended by the Information Engineering Group at the *Vienna University of Technology. DirtyData* is also developed by the Information Engineering Group and is an extension of *Prefuse* providing data structures which can be annotated with data quality problems. It also offers time-oriented data quality checks, cleansing operations and a history of the user's action.

With this project setup we ensure that our prototype also complies with technical constraints given by *QualityTime.* To ensure that our prototype uses the same table as *QualityTime*, we reuse the singleton `Manager` of the *DirtyData* project. The `Manager` consists of the user's history and a model manager containing the data and attached observers. We also use the already implemented `TPColorManager` to provide the same coloring in our data table based on annotation types.

To simplify our test procedure we annotate data with random data quality problems at start-up. With this we can easily test our visualization based on already annotated data. Each row has a 50% chance that a random cell obtains a data quality annotation. 10% of those rows have the chance that each cell of that row get completely annotated. This gives us a highly annotated but randomly distributed data basis.

4.1.2 Architecture

To comply with the architecture of *QualityTime*, we apply the Model View Controller pattern [44]. At first we planned on implementing different controllers for each view. But since user interactions require to notify changes across different views, we ended up with a single controller, named `GroovilerController`. We applied the Singleton pattern [45] to ensure that it has only one instance.

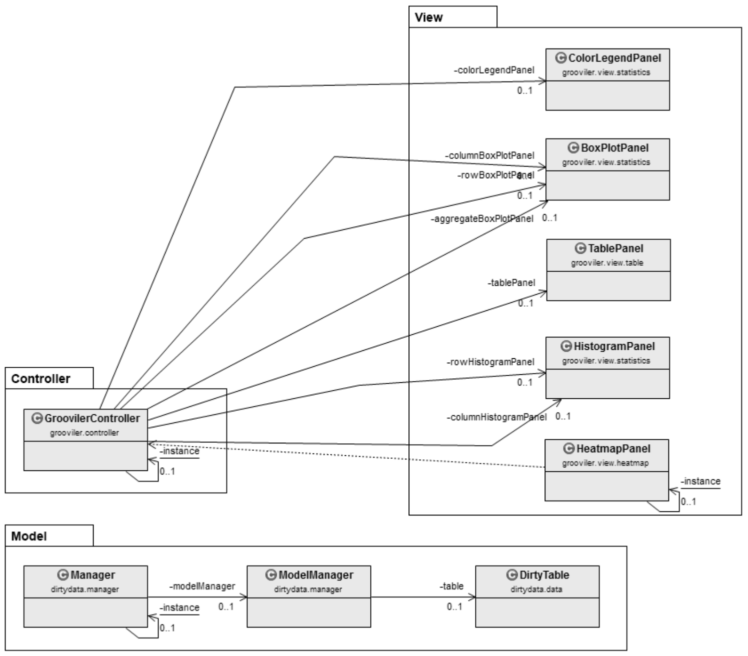


Figure 4.4: Parts of *Grooviler's* code visualized as an UML class diagram showing the Model View Controller principle. The model, controller and the heatmap panel are singleton classes, to reduce the expense with weaving them into other classes.

To simplify the dependency weaving, our heatmap panel also uses the Singleton pattern, because it is the most frequently changing component in our system. A disadvantage with that pattern is the missing reusability, which is irrelevant for our prototype.

Data Types

Data types are analyzed and set during application start-up by *Prefuse*. In our case we need the possibility to change the data type so that our configuration keeps flexible. Temporal data normally behaves ordinal. Due to our introduction of temporal granularities we need to extend the data types with a new one, namely `TEMPORAL`. With this, users can now decide to switch the representation of a column holding temporal data between ordinal and temporal (aggregated). To be downwardly compatible we only use this data type in *Grooviler* in our utility method that identifies the data type of a column.

Interestingly, *Prefuse* explicitly defines numerical data with a type named `NUMERICAL`. This is not to be confused with the data type `ORDINAL`. The difference can easily be shown with a simple example. We may have a data set containing the following numerical data: $[1, 3, 5, 2, 4, 6]$. When we set the data type to ordinal, the data set keeps the way written above. However, if it is changed to numerical, the data set gets a numerical sorting which results in the following data set: $[1, 2, 3, 4, 5, 6]$

There is another data type called `NOMINAL`, which is however merely used. *Prefuse* does not want to miss out on ordinal data, therefore it cautiously sets nominal data as ordinal. This does not have any side-effects though, since ordering is based on the occurrence in the data set.

In summary, we now have 4 data types available, whose representation can be changed at runtime through user input:

- `NUMERICAL`: data that is getting handled with numerical sorting. Can also be displayed as ordinal data.
- `ORDINAL`: any data using an ordinal scale (in our case also nominal data).
- `TEMPORAL`: data that contain a time or date. This data type is used for temporal granularities. Temporal data may also be represented as ordinal data.
- `NOMINAL`: data that has no ordinal scale. Merely in use, since it is typically defined as `ORDINAL` anyway and therefore ignored in our prototype.

Extensions for Temporal Granularities

To provide the possibility to use temporal granularities with *Prefuse* we needed to make further adaptions. The first change is to introduce a temporal data type which was already discussed in Section 4.1.2.

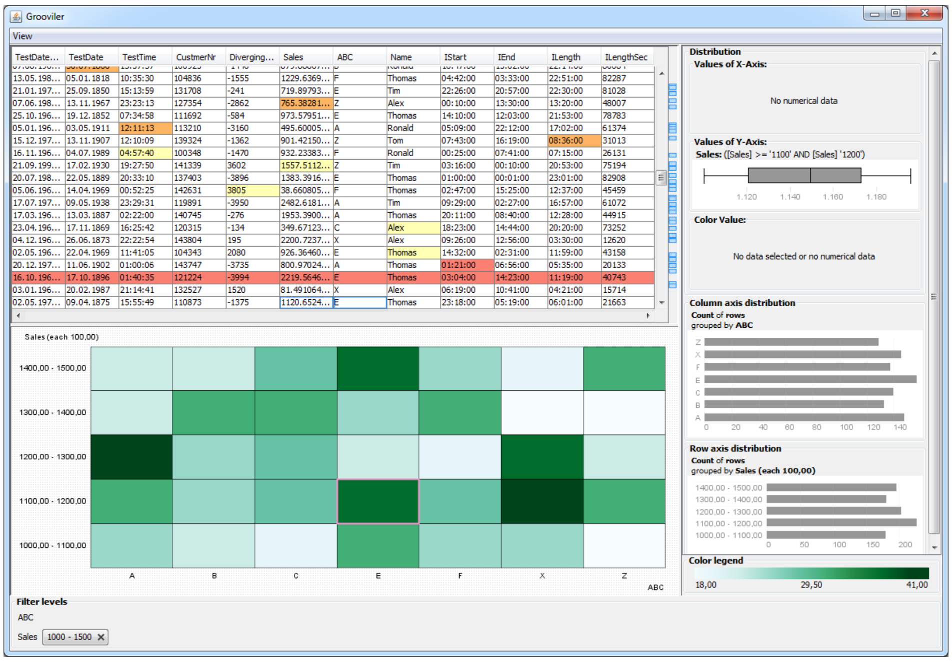
We also needed to provide a specific `ObjectRangeModel` which defines interval ranges based on a temporal granularity. We already covered the interval ranges of temporal granularities in Table 3.1, which we now implemented. The range model is an important concept in our heatmap visualization so that each cell knows its internal bounds.

Since we want to filter data with temporal granularities, we also had to extend the current `Predicate` implementations. We implemented a `TemporalGranularity Predicate` that is flexible enough to handle different temporal granularities and a specific range. With this we can define a filter containing a range (e.g., *January - March*, *Years: 1990 - 2000*, *Tuesday - Thursday*).

Prefuse Extensions

We mainly commited source code to our own prototype project, but sometimes we have to adapt the *Prefuse Vienna* itself. The `DataLib` includes serious bugs when it comes to data operations on tuple sets. In some cases nominal data is treated as ordinal data causing unexpected effects in sorting of axis labels.

Additionally, we provide extensions which are not dependent on data structures of *Grooviler* and can therefore be reused by other developers. We contribute a new layout and renderer for box plots which is covered in Section 4.1.3. We also added the data type `TEMPORAL` to the *Prefuse* Constants and a helper method inverting colors. Further, we introduces an `AggregateAxisLayout` which can be used as an underlying data source for tile-based two-dimensional visualizations (e.g., heatmaps).



TestDate...	TestDate	TestTime	CustmerNr	Diverging...	Sales	ABC	Name	IStart	IEnd	ILength	ILengthSec
13.05.198...	05.01.1818	10:35:30	104836	-1555	1229.6369...	F	Thomas	04:42:00	03:33:00	22:51:00	82287
21.01.197...	25.09.1850	15:13:59	131708	-241	719.89793...	E	Tim	22:26:00	20:57:00	22:30:00	81028
07.06.198...	13.11.1967	23:23:13	127354	-2862	765.38281...	Z	Alex	00:10:00	13:30:00	13:20:00	48007
25.10.196...	19.12.1852	07:34:58	111692	-584	973.57951...	E	Thomas	14:10:00	12:03:00	21:53:00	78783
05.01.196...	03.05.1911	12:11:13	113210	-3160	495.60005...	A	Ronald	05:09:00	22:12:00	17:02:00	61374
15.12.197...	13.11.1907	12:10:09	139324	-1362	901.42150...	Z	Tom	07:43:00	16:19:00	08:36:00	31013
16.11.196...	04.07.1989	04:57:40	100348	-1470	932.23383...	F	Ronald	00:25:00	07:41:00	07:15:00	26131
21.09.199...	17.02.1930	19:27:50	141339	3602	1557.5112...	Z	Tim	03:16:00	00:10:00	20:53:00	75194
20.07.198...	22.05.1889	20:33:10	137403	-3896	1383.3916...	E	Thomas	01:00:00	00:01:00	23:01:00	82908
05.06.196...	14.04.1969	00:52:25	142631	3805	38.660805...	F	Thomas	02:47:00	15:25:00	12:37:00	45459
17.07.197...	09.05.1938	23:29:31	119891	-3950	2482.6181...	A	Tim	09:29:00	02:27:00	16:57:00	61072
17.03.196...	13.03.1887	02:22:00	140745	-276	1953.3900...	A	Thomas	20:11:00	08:40:00	12:28:00	44915
23.04.196...	17.11.1869	16:25:42	120315	-134	349.67123...	C	Alex	18:23:00	14:44:00	20:20:00	73252
04.12.196...	26.06.1873	22:22:54	143804	195	2200.7237...	X	Alex	09:26:00	12:56:00	03:30:00	12620
02.05.196...	22.04.1969	11:41:05	104343	2080	926.36460...	E	Thomas	14:32:00	02:31:00	11:59:00	43158
20.12.197...	11.06.1902	01:00:06	143747	-3735	800.97024...	A	Thomas	01:21:00	06:56:00	05:35:00	20133
16.10.196...	17.10.1896	01:40:35	121224	-3994	2219.5646...	E	Thomas	03:04:00	14:23:00	11:19:00	40743
03.01.196...	20.02.1987	21:14:41	132527	1520	81.491064...	X	Alex	06:19:00	10:41:00	04:21:00	15714
02.05.197...	09.04.1875	15:55:49	110873	-1375	1120.6524...	E	Thomas	23:18:00	05:19:00	06:01:00	21663

Figure 4.5: GUI of *Grooviler*. Table showing the underlying data (top) aggregated in a heatmap (center). Statistics panel (right) including a color legend (lower right). Panel showing the zoom history/filter levels (bottom).

4.1.3 GUI

Figure 4.5 gives an overview of *Grooviler's* GUI. The main panel is split with a split pane into two separate panels. The left component holds the table containing the underlying data and the heatmap visualization. The right view renders statistics about the data set and selections. At the bottom we render the filter levels which are set during zoom operations.

Table Panel

The table panel consists of a `JTable` with a custom cell renderer. It sets the background color, as well as the blue border in case of an annotated or highlighted cell. To comply with the standards of *QualityTime* we use the same background colors by re-using the `TPColorManager`.

On the right side of the table's scroll bar we place a table indicator component containing interactive markers. Each marker represents an annotated vertical position (row) inside the data table. The position gets calculated with ratio calculation between the item number in relation to the amount of items. By clicking a marker the table automatically scrolls to that specific item.

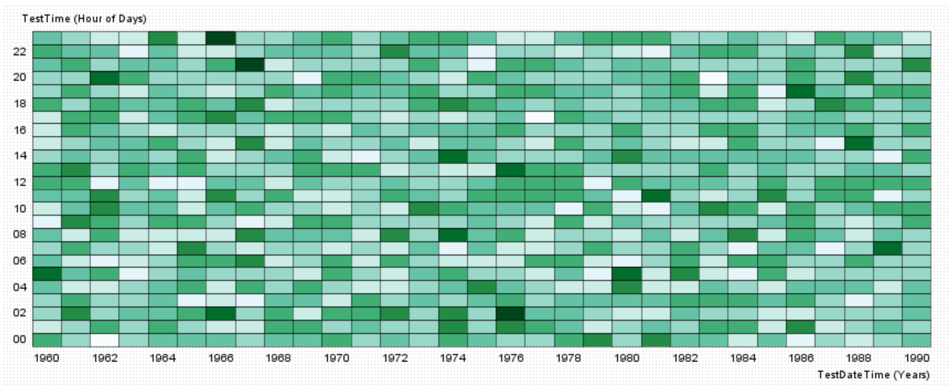


Figure 4.6: Heatmap Visualization in *Grooviler*. In this example tuples get aggregated in hour of days (column *TestTime*) and years (column *TestDateTime*). Axis labels are shown at the left and bottom of the heatmap and display the labels based on the current aggregation. Not every axis label is shown due to a configured spacing that prevents overlapping labels.

Heatmap

The configurable heatmap aggregating values is the core of our prototype and consists of several layouts and renderers. As a basis we use an `AggregateTable` where items can hold aggregate items within. Our two-dimensional approach needs the possibility to specify the x- and y-position of each item. Instead of re-implementing already existing data structures with horizontal and vertical positions, we decided to use the row-index of the aggregate table. Based on the column count, each position can easily be recalculated:
$rowIndex = floor(index/columnCount)$
$columnIndex = index - columnCount * rowIndex$

Layout The default implementation of axis layouts calculates visual items based on its data type. This would result in unevenly distributed visual items based on the underlying data. Therefore we have to implement our own axis layout. This `AggregateAxisLayout` handles underlying data as evenly distributed ordinal data. It additionally calculates the width and height of visual items based on the total amount of tiles. A shape renderer takes these dimensions as a reference to draw rectangles with the given size.

The implementation of the axis layouts for the underlying `AxisLabelLayout` had to be adapted as well. Our `NominalAxisLayout` handles ordinal data nominally. The *Prefuse Toolkit* automatically uses an ordinal layout for data types which are not explicitly defined as nominal. This sometimes results in illogical, alphabetical sorting, like ascending month names (e.g., *April to September*) or wrong sorted bins (e.g., $[1-5[, [10-15[, [5-10[$).

Since our visual items have a fiducial in the lower left corner, the default implementation for the layout of axis labels is not sufficient. Using the default `AxisLabelLayout`

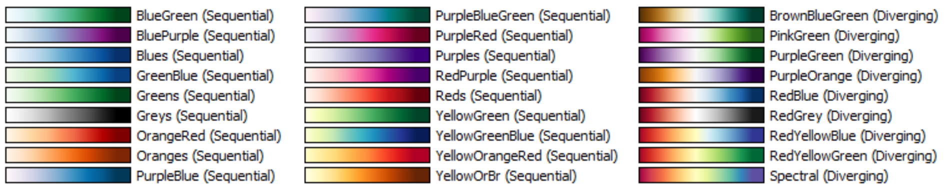


Figure 4.7: Color Palettes in *Grooviler* taken from ColorBrewer [46]. The first 18 color palettes can be used for sequential data. The 9 palettes in the right column can be used for diverging quantitative values.

would result in axis labels at the lower left corner of each heatmap tile. Therefore we extend the class and slightly modify it to a `CenteredAxisLabelLayout` which relocates each label to the center of a tile.

The built-in axis label layout provides the possibility to define a specific spacing in pixels. It is used to prevent overlappings by taking the spacing as minimum width between origins of labels. Due to our configurability labels may need a different spacing. Therefore we calculate the spacing based on the longest occurring label.

Because of the missing possibility to render a simple text onto a *Prefuse* visualization, we further created a `AxisDescriptionLabelLayout`. Based on the underlying axis label layout, a given text will be rendered above (for y-axes) or beyond (for x-axes) axis labels. The axis description is computed based on the current axis configuration with the help of the `BinLabelHelper` utility class.

Coloring Coloring is done with an extension of `DataColorAction`. This class provides the possibility to map values of a certain column to a color. We extend it to an `AggregateData ColorAction` which takes a color configuration as constructor argument. Each tile calculates its value based on the containing aggregate items and the setting of the coloring. This can be the count of tuples, annotation count or a specified aggregation function. The relation between the bounds of the color palette and mappable values is done in the `setup()` method.

Next to the heatmap we provide a color legend panel, displaying the minimum, medium and maximum value of the heatmap. On every update action in the heatmap the `ColorLegendUpdateAction` notifies the color legend panel with new values to update accordingly.

We implemented a total of 27 different color palettes provided by the online tool ColorBrewer [46]. 18 of them can be used to display sequential values while 9 color palettes are suited to display diverging values. Since we only visualize quantitative data in our heatmap, we do not implement the given 8 color palettes for qualitative data by ColorBrewer. Our used color schemes cover simple gradients up to a spectrum scheme ranging from red to yellow, white, green and finally blue. Figure 4.7 gives an overview about all provided color palettes.

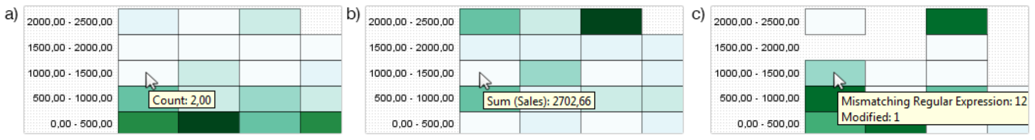


Figure 4.8: Heatmap Tooltip in *Grooviler*. The tooltip is shown when users hover above a certain cell in the heatmap. It shows the current tuple count (a), aggregation function (b) or data quality types with corresponding values (c).

Sometimes it is difficult to differentiate between heatmap cells that contain low values and ones that are empty (and therefore not rendered). Most color palettes treat low values as a very light color, sometimes converging to white (see Figure 4.8 (c)) To address this problem, we introduced a dotted background pattern to distinguish between those cases.

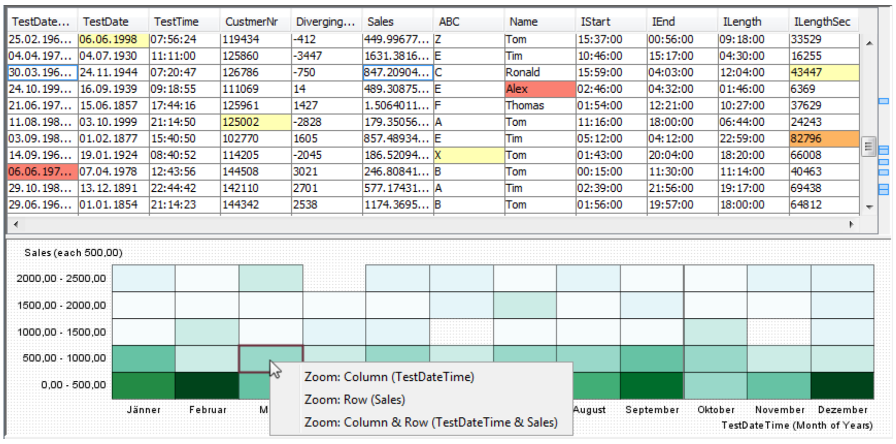
Tooltip We implement a detail on demand technique by adding a tooltip to our heatmap (see Figure 4.8). This shows a heatmap tile's label, which is set during our color action initializing. At that moment we have the access to underlying aggregated data and the result based on aggregated items.

Dependent on the heatmap's configuration the tooltip shows appropriate information. We try to display it as descriptive as possible. As an example the annotation count shows a textual distribution over different annotation types (see Figure 4.8 (c)).

Highlighting We provide a `ControlListener` to respond to user's mouse input. When a user left-clicks on a certain cell in the heatmap, the appropriate cell and its relevant data is getting highlighted (see Figure 4.9). Thereby the table gets annotated with interactive blue markers next to the data table which can be used to jump to a corresponding table row. Finally we provide a feedback for the user by rendering a border around the selected cell. The border color is an inversion of the cell's color to be independent of freely configurable color palettes.

Also, by selection, some table's cells get marked with a blue, inner border. These annotated cells show the columns which are used in the heatmap visualization. In our example in Figure 4.9 the user selected the cell with Sales between 1.000 and 1.500 € in February. The data table marks the column *TestDateTime* because it is used as horizontal value and the *Sales* column because of its configuration as vertical axis. The *ILengthSec* column (abbreviation for *Interval Length in Seconds*) is marked because of the heatmap's configuration to show data quality annotations.

Zooming Our `ControlListener` also reacts to a user's right-click. It opens a context menu that contains the possibility to zoom inside the row, column or both (see Figure 4.9). To support users in their selection, we additionally provide the field name which is used in the respective column or row.



TestDate...	TestDate	TestTime	CustmerNr	Diverging...	Sales	ABC	Name	IStart	IEnd	ILength	ILengthSec
25.02.196...	06.06.1998	07:56:24	119434	-412	449.99677...	Z	Tom	15:37:00	00:56:00	09:18:00	33529
04.04.197...	04.07.1930	11:11:00	125860	-3447	1631.3816...	E	Tim	10:46:00	15:17:00	04:30:00	16255
30.03.196...	24.11.1944	07:20:47	126786	-750	847.20904...	C	Ronald	15:59:00	04:03:00	12:04:00	43447
24.10.199...	16.09.1939	09:18:55	111069	14	489.30875...	E	Alex	02:46:00	04:32:00	01:46:00	6369
21.06.197...	15.06.1857	17:44:16	125961	1427	1.5064011...	F	Thomas	01:54:00	12:21:00	10:27:00	37629
11.08.198...	03.10.1999	21:14:50	125002	-2828	179.35056...	A	Tom	11:16:00	18:00:00	06:44:00	24243
03.09.198...	01.02.1877	15:40:50	102770	1605	857.48934...	E	Tim	05:12:00	04:12:00	22:59:00	82796
14.09.196...	19.01.1924	08:40:52	114205	-2045	186.52094...	X	Tom	01:43:00	20:04:00	18:20:00	66008
06.06.197...	07.04.1978	12:43:56	144508	3021	246.80841...	B	Tom	00:15:00	11:30:00	11:14:00	40463
29.10.198...	13.12.1891	22:44:42	142110	2701	577.17431...	A	Tim	02:39:00	21:56:00	19:17:00	69438
29.06.196...	01.01.1854	21:14:23	144342	2538	1174.3695...	B	Tom	01:56:00	19:57:00	18:00:00	64812

Figure 4.9: Heatmap Interaction Operations Zooming and Linking in *Grooviler*. In our example the user left-clicked, and therefore highlighted the cell with Sales between 1.000 and 1.500 € in March. With a right-click the zoom menu shows the possibilities to zoom inside the row, column or both.

Rule	**Error Message**
When the underlying data type is ordinal and the axis shows only one ordinal bin	You can't zoom deeper into a single ordinal bin
When the underlying data is temporal and configured with temporal granularity *Seconds*	Milliseconds are not supported
When the underlying data is numerical and the row/column/cell contains only a single value	Zooming into a single numeric value is not supported

Table 4.1: Zoom Rules of the *Grooviler* Heatmap when zooming is **not** possible.

Of course users can't zoom endlessly. Table 4.1 show the rules when it is **not** possible to zoom. These rules are applied to the row and column respectively. The zoom function gets disabled if one of the rules apply.

The following actions are performed during zooming:

- Clear the current selection and visualizations based on selection.
- Set a filter to the range of the selected heatmap tile (e.g. *0 - 1.000*, *Year: 1999*, *Months: 4 - 6*).
- Switch to the next logical temporal granularity (only with temporal data, see Table 3.1).
- Add a filter component to the zoom filter panel.
- Update the heatmap with applied filters.
- Update corresponding histogram panels with applied filters.

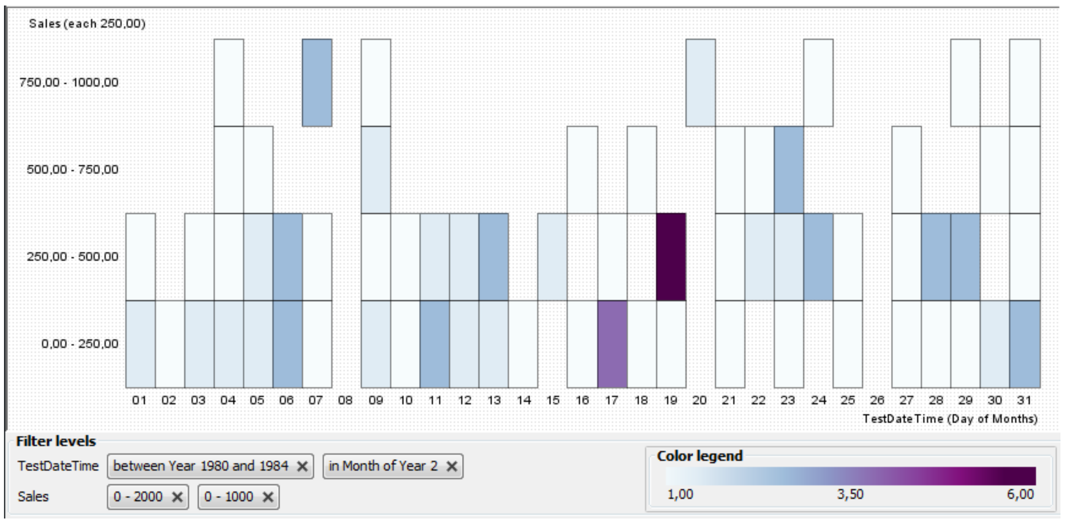


Figure 4.10: Example of nested zoom operations in *Grooviler*. The user zoomed into April of year 1976 with sales between 0 and 1.000.

It is possible to perform nested zooms (see Figure 4.10). In this example, the user first zoomed into the year ranging from 1980 to 1984 and sales between 0 and 2.000. Then he/she zoomed into the second month of the year (February) and sales between 0 and 1.000.

As a side note to Figure 4.10: Although the filter is set to 0 - 1.000 you may notice that the vertical axis only shows values up to 900. This is due to our initial step calculation resulting in 3 tiles with a step-width of 300. Our calculation tries to find a balance between not showing too many tiles and giving expressive values as interval ranges. In this example the maximum value is 829. We could have used a step-width of 400, also provided by 3 tiles but this would result in a distorted visualization (see Figure 3.7). Our technique is a mixture of decade-based rounding with a dynamic control variable based on the value's span.

Statistics Panel

The Statistics panel consists of 3 box plot visualizations and 2 histograms. Each box plot shows a selected subset of the data grouped by column, row and both. Both histograms summarize the value distribution of the current heatmap configuration of each axis respectively.

Box Plot We implement a *Prefuse* visualization for a basic box plot [47] without hiskers. It is rendered horizontally so that our remaining space inside the statistics panel is efficiently used. For this, we develop a `BoxPlotLayout` which constructs a single `VisualItem` holding the positions of the data's quartiles. The `BoxPlotRenderer` then takes this `VisualItem` as a reference for the box plots shapes. It draws rectangles

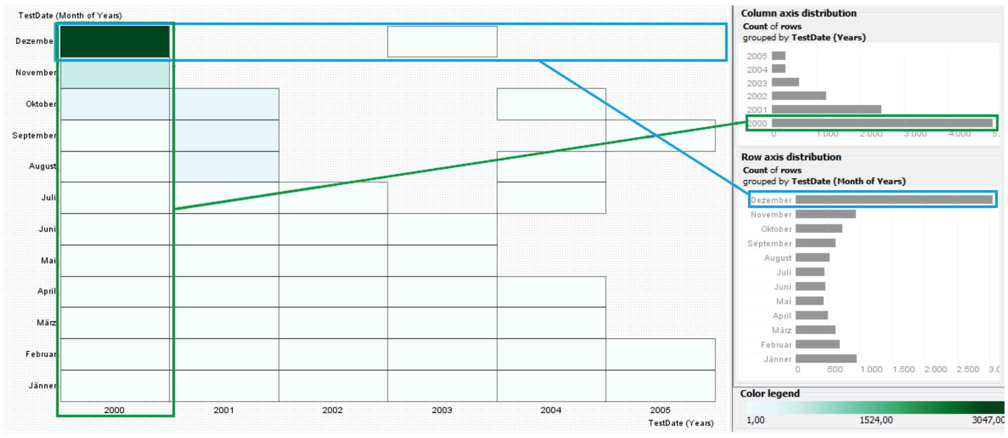


Figure 4.11: Histogram Visualizations in *Grooviler*. The upper histogram shows the distribution of the horizontal axis. All values of year 2000 is summarized in a separate histogram bar (green border). The lower histogram renders the vertical axis distribution. Values occurring in December are summarized in a specific bar in the histogram (blue border). Each histogram consists of a title which explains what the histogram is visualizing.

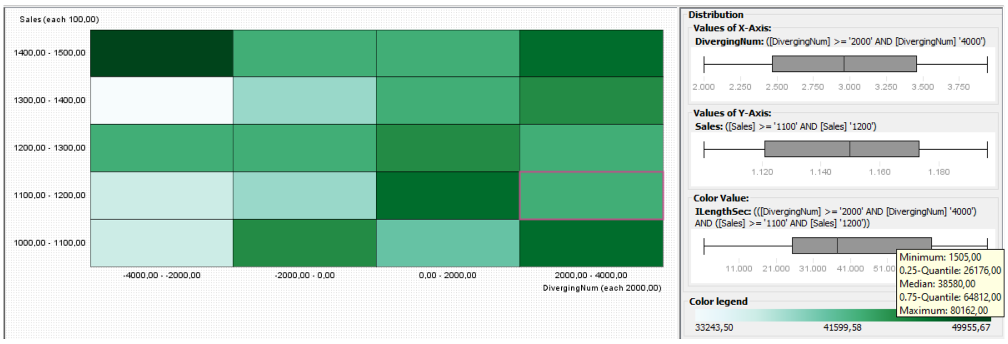


Figure 4.12: Box Plot Visualization in *Grooviler*. The first box plot shows the value distribution of the selected region of the horizontal axis. The second box plot gives an overview about the distribution of the selected vertical values. The third box plot gives an overview about the calculated color field inside the cell.

and lines at the corresponding positions. The height of the box plot is calculated based on the layout bounds giving a 5% padding.

Our implementation of box plots is independent of *QualityTime* or *Grooviler* and is therefore added to *Prefuse Vienna*. Like other *Prefuse* visualizations, the `BoxPlotLayout` requires a group that contains underlying data and a field name of a numerical column. Optionally an `AxisLabelLayout` combined with an `AxisRenderer` can be provided to display axis labels. We delivered a demo application with *Prefuse Vienna* to show its usage.

Our prototype holds 3 different box plot visualizations (see Figure 4.12). The first one shows the value distribution of the selected region of our horizontal axis. In this example the horizontal axis is defined with the numerical column *DivergingNum*. The box plot now only shows values of the column *DivergingNum* which occur in our selected range.

The second box plot works in the exactly same way, except that it takes the vertical axis into account. The third plot gives an overview about the calculated values. In Figure 4.12) we defined the calculation and coloration based on the mean of the column *ILengthSec*. Therefore we now only show the distribution of *ILengthSec* values which fall both inside the selected region of horizontal **and** vertical axis. We now have an overview about Interval length in seconds which *DivergingNum* is between 2.000 and 4.000 and Sales is between 1.100 and 1.200.

The leading description title of each box plot reveals what the box plot is about. A details on demand tooltip shows the exact numbers.

Histogram The histogram visualization consists of nominal layouts so that categories get correctly ordered. The shape renderer is slightly modified, so that rectangles are drawn from the left side to each visual-item's Position. We calculate the height of the visualization with the amount of bars and a padding between them. Axis labels of the horizontal axes use a calculated spacing based on the width of the longest label.

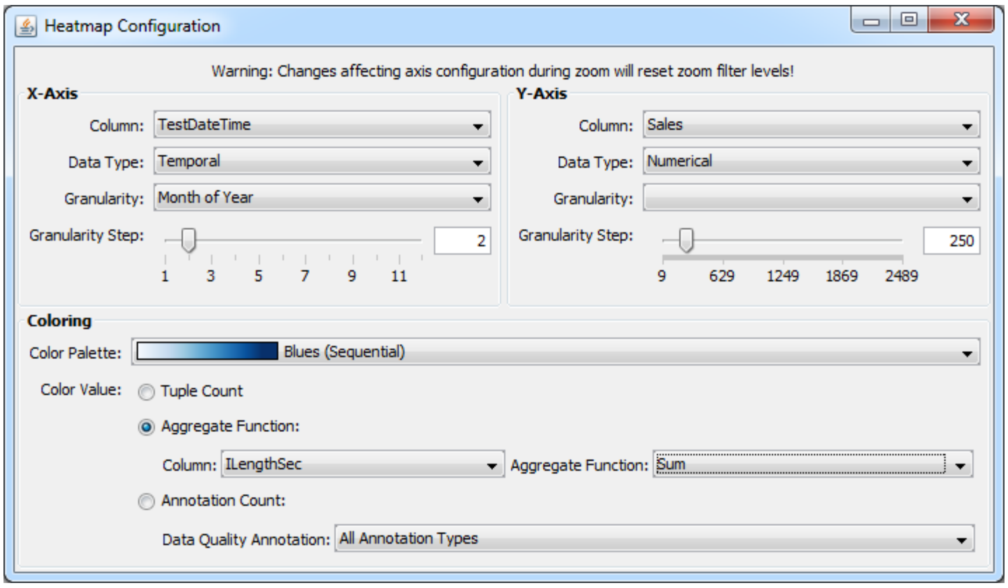
The Statistics Panel renders two histograms (See Figure 4.11). The first histogram is giving an overview about the heatmap's row distribution. The second histogram shows the distribution of the column axis. At first we planned to support the user's visual mapping by providing a horizontal and vertical histogram for each axis respectively. We abandoned this idea due to missing horizontal space. Users want to get an overview about the data distribution. A small horizontal histogram with a scroll bar would interfere with this approach.

However, users might now have problems associating the histograms to its particular axis. To address this concern we add a description title on top of each histogram which explains where the rendered data comes from. In the description we cover the following information:

- Aggregation function (e.g., Count, Sum, ...)
- Calculation field (e.g., tuple row, *Sales*, ...)
- Field and aggregation for grouping

Therefore we might have descriptions like:

- **Count** of **rows** grouped by **TestDate (Years)**
- **Sum** of **Sales** grouped by **Customer Count (each 500,00)**
- **Count** of **all annotations** grouped by **Game Category**



Heatmap Configuration

Warning: Changes affecting axis configuration during zoom will reset zoom filter levels!

X-Axis

Column: TestDateTime

Data Type: Temporal

Granularity: Month of Year

Granularity Step: 2

Y-Axis

Column: Sales

Data Type: Numerical

Granularity:

Granularity Step: 250

Coloring

Color Palette: Blues (Sequential)

Color Value: Tuple Count [✓ Aggregate Function:] Annotation Count:

Column: lLengthSec Aggregate Function: Sum

Data Quality Annotation: All Annotation Types

Figure 4.13: Heatmap Configuration Frame of *Grooviler*. Configuration of each axis by column, data type, granularity and step-width (top). Calculation can be set as tuple count, aggregation function, or annotation count (bottom).

Filter Zoom Level Panel

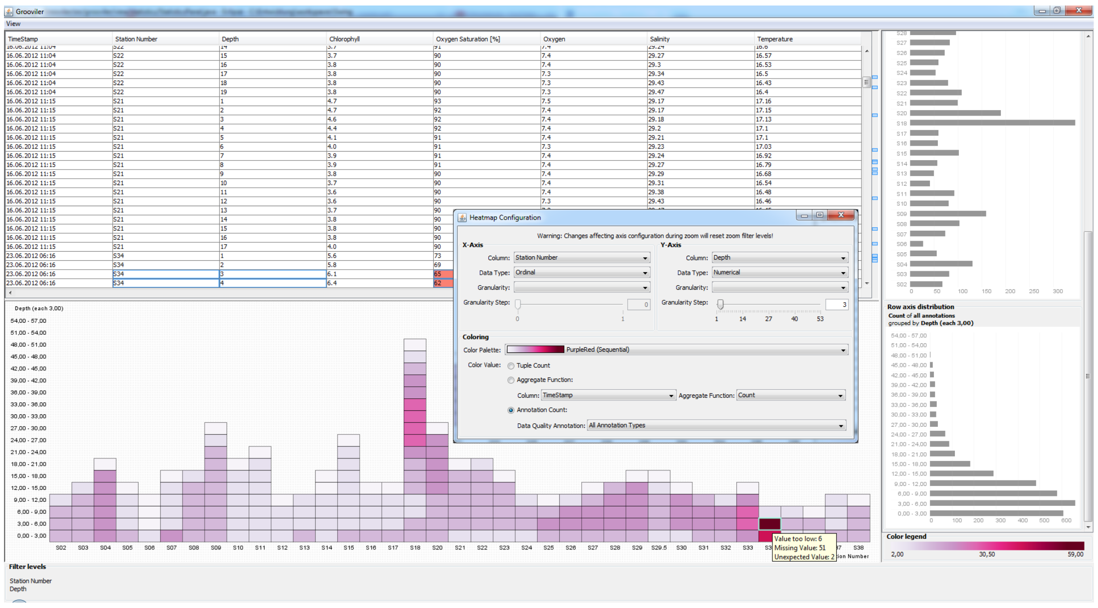
The filter levels panel below the heatmap shows the applied filters as interactive buttons (see Figure 4.10 bottom). They can be used to clear a specific filter level of each axis. With this, users have a flexible way to remove filters based on their needs.

Heatmap Configuration

We implement the configuration inside a separate frame, so that users can decide its visibility and position. Therefore, we apply the Singleton pattern, so it is ensured that there exists only one configuration.

To provide a fast transition between user's configuration and causing visual changes, we do not implement an apply button. Hence, the visualization needs to adapt on every configuration change. Additionally, most of the components inside the frame are dependent on each other. As an example, changing the column of the x-axis results in updating the combo box for data types, which results in (re)setting a pre-calculated step-width. Our simple modification changed the data basis and enforces a new rendering of the heatmap and corresponding histogram visualizations.

With this in mind, we adopt each configuration element and only perform necessary visualization adaptions. As an example, with the change of color calculation, we only update the color configuration, labels and histograms. The rest of the heatmap remains untouched.



TimeStamp	Station Number	Depth	Chlorophyll	Oxygen Saturation [%]	Oxygen	Salinity	Temperature
16.06.2012 11:04	S22	15	3.7	90	7.4	29.27	16.57
16.06.2012 11:04	S22	16	3.8	90	7.4	29.3	16.53
16.06.2012 11:04	S22	17	3.8	90	7.3	29.34	16.5
16.06.2012 11:04	S22	18	3.8	90	7.3	29.43	16.43
16.06.2012 11:04	S22	19	3.8	90	7.3	29.47	16.4
16.06.2012 11:15	S21	1	4.7	93	7.5	29.17	17.16
16.06.2012 11:15	S21	2	4.7	92	7.4	29.17	17.15
16.06.2012 11:15	S21	3	4.6	92	7.4	29.18	17.13
16.06.2012 11:15	S21	4	4.4	92	7.4	29.2	17.1
16.06.2012 11:15	S21	5	4.1	91	7.4	29.21	17.1
16.06.2012 11:15	S21	6	4.0	91	7.3	29.23	17.03
16.06.2012 11:15	S21	7	3.9	91	7.4	29.24	16.92
16.06.2012 11:15	S21	8	3.9	91	7.4	29.27	16.79
16.06.2012 11:15	S21	9	3.8	90	7.4	29.29	16.68
16.06.2012 11:15	S21	10	3.7	90	7.4	29.31	16.54
16.06.2012 11:15	S21	11	3.6	90	7.4	29.38	16.48
16.06.2012 11:15	S21	12	3.6	90	7.3	29.43	16.46
16.06.2012 11:15	S21	13	3.7	90			
16.06.2012 11:15	S21	14	3.8	90			
16.06.2012 11:15	S21	15	3.8	90			
16.06.2012 11:15	S21	16	3.8	90			
16.06.2012 11:15	S21	17	4.0	90			
23.06.2012 06:16	S34	1	5.6	73			
23.06.2012 06:16	S34	2	5.8	69			
23.06.2012 06:16	S34	3	6.1	65			
23.06.2012 06:16	S34	4	6.4	62			

Heatmap Configuration

Warning: Changes affecting axis configuration during zoom will reset zoom filter levels!

X-Axis
Column: Station Number
Data Type: Ordinal
Granularity:
Granularity Step: 0

Y-Axis
Column: Depth
Data Type: Numerical
Granularity:
Granularity Step: 3

Coloring
Color Palette: PurpleRed (Sequential)
Color Value: ( ) Tuple Count
( ) Aggregate Function:
Column: TimeStamp Aggregate Function: Count
(●) Annotation Count:
Data Quality Annotation: All Annotation Types

Value too low: 6
Missing Value: 51
Unexpected Value: 2

Figure 4.14: *Grooviler* configured with station numbers in relation to depth. Coloring is done with the amount of any data quality annotation.

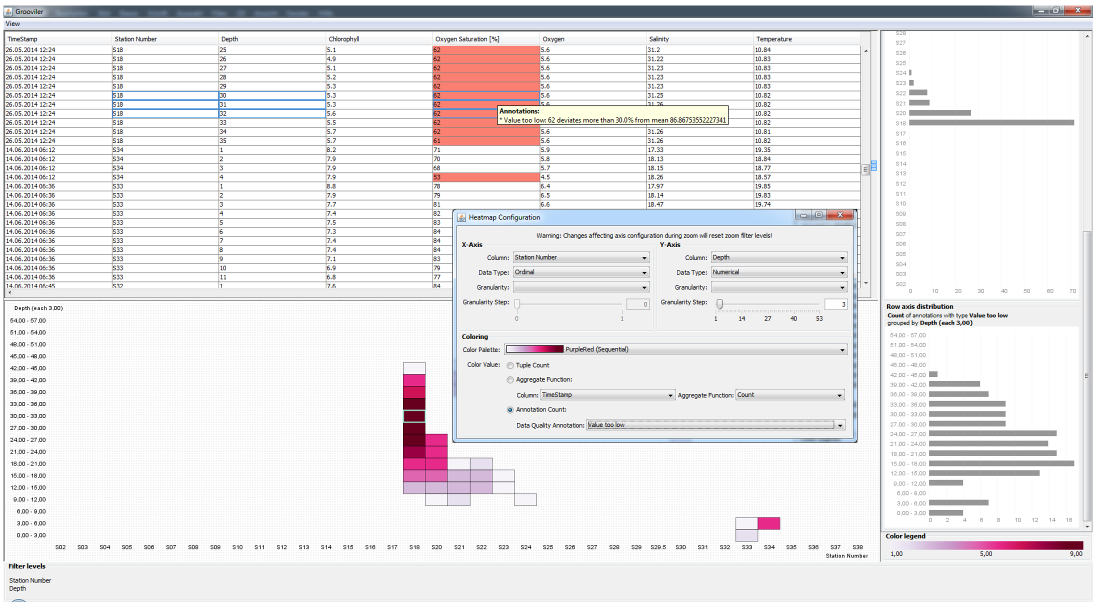
4.2 Use Cases

Since *Grooviler* is designed flexible, we will now cover two examples that act as possible use cases. The first example shows how *Grooviler* can visualize already annotated data entries to find out possible causes and relations. Then we will take a look on an example where we can visually identify a specific data quality problem thanks to the heatmap visualization.

For our use cases we will use an adapted data set from U.S. Geological Survey containing measurements about water quality of San Francisco Bay [48]. It contains measurements at specific times, depths and stations prodiving information about temperature, salinity and oxygen production. This data set is also used in our evaluation and therefore explained more in detail in Section 5.2.

4.2.1 Displaying Already Annotated Data Quality Problems

In this use case we want to take a look at the data set which is already annotated with different types of annotations. *Grooviler* provides the flexibility to let us decide how we want to render annotations in a heatmap. In our case we want to narrow down the error-prone measurement stations. We have an idea that data quality problems might occur at measurements with a specific depth. Therefore we configured the heatmap with station numbers in relation to depth. The coloring is based on occurrences of all data quality annotations (see Figure 4.14).



TimeStamp	Station Number	Depth	Chlorophyll	Oxygen Saturation [%]	Oxygen	Salinity	Temperature
26.05.2014 12:24	S18	25	5.1	62	5.6	31.2	10.84
26.05.2014 12:24	S18	26	4.9	62	5.6	31.22	10.83
26.05.2014 12:24	S18	27	5.1	62	5.6	31.23	10.83
26.05.2014 12:24	S18	28	5.2	62	5.6	31.23	10.83
26.05.2014 12:24	S18	29	5.3	62	5.6	31.23	10.83
26.05.2014 12:24	S18	30	5.3	62	5.6	31.25	10.82
26.05.2014 12:24	S18	31	5.3	62	5.6	31.26	10.82
26.05.2014 12:24	S18	32	5.6	62			10.82
26.05.2014 12:24	S18	33	5.5	62			10.82
26.05.2014 12:24	S18	34	5.7	62	5.6	31.26	10.81
26.05.2014 12:24	S18	35	5.7	61	5.6	31.26	10.82
14.06.2014 06:12	S34	1	8.2	71	5.9	17.33	19.35
14.06.2014 06:12	S34	2	7.9	70	5.8	18.13	18.84
14.06.2014 06:12	S34	3	7.9	68	5.7	18.15	18.77
14.06.2014 06:12	S34	4	7.9	53	4.5	18.26	18.57
14.06.2014 06:36	S33	1	8.8	78	6.4	17.97	19.85
14.06.2014 06:36	S33	2	7.9	79	6.5	18.14	19.83
14.06.2014 06:36	S33	3	7.7	81	6.6	18.47	19.74
14.06.2014 06:36	S33	4	7.4	82			
14.06.2014 06:36	S33	5	7.5	83			
14.06.2014 06:36	S33	6	7.3	84			
14.06.2014 06:36	S33	7	7.4	84			
14.06.2014 06:36	S33	8	7.4	84			
14.06.2014 06:36	S33	9	7.1	83			
14.06.2014 06:36	S33	10	6.9	79			
14.06.2014 06:36	S33	11	6.8	77			
14.06.2014 06:45	S32	1	7.6	84			

Annotations:
* Value too low: 62 deviates more than 30.0% from mean 86.86753552227341

Heatmap Configuration

Warning: Changes affecting axis configuration during zoom will reset zoom filter levels!

X-Axis
Column: Station Number
Data Type: Ordinal
Granularity:
Granularity Step: 0

Y-Axis
Column: Depth
Data Type: Numerical
Granularity:
Granularity Step: 3

Coloring
Color Palette: PurpleRed (Sequential)
Color Value: Tuple Count
Aggregate Function:
Column: TimeStamp Aggregate Function: Count
[✓ Annotation Count:]
Data Quality Annotation: Value too low

Figure 4.15: *Grooviler* configured with station numbers in relation to depth. Coloring is done with the amount of data quality annotations with the type “Value too low”.

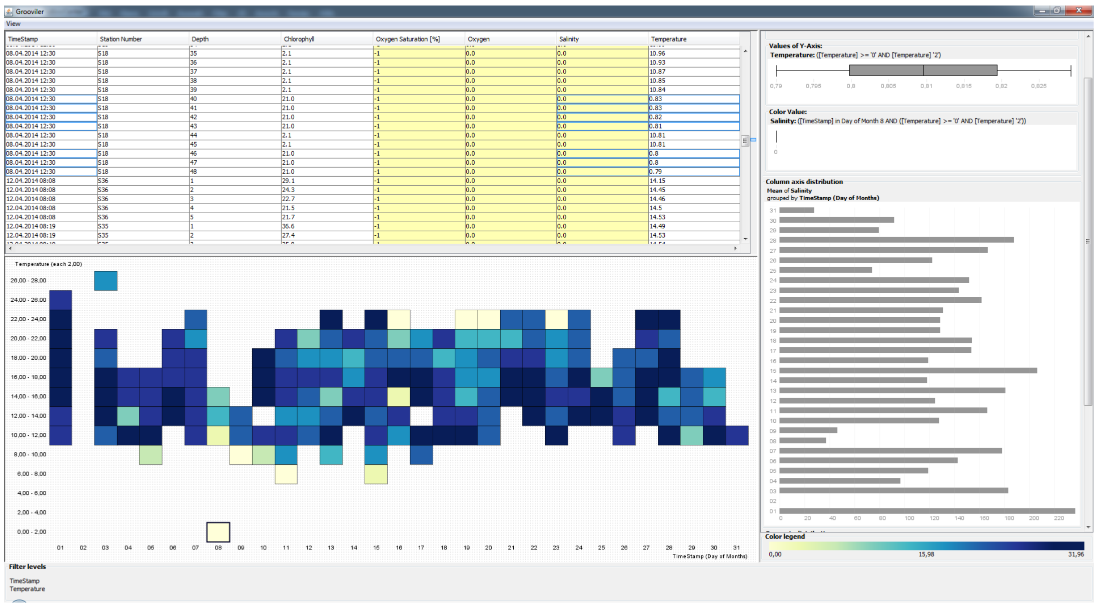
Based on the heatmap visualization it seems that station number `S18` seems to contain the highest amount of annotated entries. A side check with the histogram confirms our impression. The most concentrated data quality problems seem to occur in 3 to 6 meters depth in station `S34`. The tooltip tells us that most of the dirty data in this area results from missing values.

We will leave the missing values of station `S34` for later and want to investigate other data quality problems. So we switch the data quality problems in the coloring combo box and find an interesting pattern with the annotation “Value too low”. We can see that for this annotation station `S18` is showing more annotated entries. By clicking on the heatmap cell for this station and depths between 30 and 33 *Grooviler* highlighted the according data entries inside our table.

We can see that the annotated entries are about oxygen saturation, and additionally see that adjacent table entries also contain that annotation. The tooltip gives us additional information about the current data quality annotation. In this case the oxygen saturation seems to be lower than 30% of the calculated mean.

4.2.2 Identifying Further Data Quality Problems

For identifying further data quality problems, we might have assumptions leading us to desired configurations. For example, in the above use case we thought of problems occurring at specific depth, which lead to a heatmap configured with a depth-axis.



TimeStamp	Station Number	Depth	Chlorophyll	Oxygen Saturation [%]	Oxygen	Salinity	Temperature
08.04.2014 12:30	S18	35	2.1	-1	0.0	0.0	10.96
08.04.2014 12:30	S18	36	2.1	-1	0.0	0.0	10.93
08.04.2014 12:30	S18	37	2.1	-1	0.0	0.0	10.87
08.04.2014 12:30	S18	38	2.1	-1	0.0	0.0	10.85
08.04.2014 12:30	S18	39	2.1	-1	0.0	0.0	10.84
08.04.2014 12:30	S18	40	21.0	-1	0.0	0.0	0.83
08.04.2014 12:30	S18	41	21.0	-1	0.0	0.0	0.83
08.04.2014 12:30	S18	42	21.0	-1	0.0	0.0	0.82
08.04.2014 12:30	S18	43	21.0	-1	0.0	0.0	0.81
08.04.2014 12:30	S18	44	2.1	-1	0.0	0.0	10.81
08.04.2014 12:30	S18	45	2.1	-1	0.0	0.0	10.81
08.04.2014 12:30	S18	46	21.0	-1	0.0	0.0	0.8
08.04.2014 12:30	S18	47	21.0	-1	0.0	0.0	0.8
08.04.2014 12:30	S18	48	21.0	-1	0.0	0.0	0.79
12.04.2014 08:08	S36	1	29.1	-1	0.0	0.0	14.15
12.04.2014 08:08	S36	2	24.3	-1	0.0	0.0	14.45
12.04.2014 08:08	S36	3	22.7	-1	0.0	0.0	14.46
12.04.2014 08:08	S36	4	21.5	-1	0.0	0.0	14.5
12.04.2014 08:08	S36	5	21.7	-1	0.0	0.0	14.53
12.04.2014 08:19	S35	1	36.6	-1	0.0	0.0	14.49
12.04.2014 08:19	S35	2	27.4	-1	0.0	0.0	14.53

Values of Y-Axis:
Temperature: ([Temperature] >= '0' AND [Temperature] '2')

Color Value:
Salinity: ([TimeStamp] in Day of Month 8 AND ([Temperature] >= '0' AND [Temperature] '2'))

Column axis distribution
Mean of Salinity
grouped by TimeStamp (Day of Months)

Figure 4.16: Random configuration in *Grooviler* showing a suspicious pattern in temperatures around 0.0° Celsius. It also reveals potential missing measurements of the second day of month.

Another possibility is to explore the data set using random configurations with different granularities or step-widths. This lead us to a configuration seen in Figure 4.16 which doesn't really make sense. Why would you want to set the day of month for measurements in relation with temperatures? However, it revealed a suspicious pattern which leads to an ongoing investigation.

The heatmap reveals one or more outliers at the eight day of month around 0.0° Celsius. By clicking on it, we relate to the corresponding highlighted table entries. Adjacent table entries have temperatures about 10° Celsius, so this is likely to be a data quality problem. We can also spot possible outliers about 27° Celsius on the third day of month.

Looking at the histogram in the statistics panel we revealed another anomaly, which can also be seen at the heatmap. No measurements were made at the second day of month.

A side effect of working with *Grooviler* is the possibility to gain knowledge of the data set. In this example, coloring is based on the mean of salinity. Based on the heatmap visualization we can see that temperature and salinity do not seem to correlate. Otherwise the coloring of the heatmap would result in a more gradient pattern instead of this flecked one.

CHAPTER 5

Evaluation

To evaluate the usefulness of our prototype and to gain answers to our research questions, we performed a qualitative study with a special focus on revealing insights. We modified a data set for our testing purposes and let 3 test users execute 5 tasks covering different aspects of *Grooviler*. We thereby observed participants using our prototype to evaluate which types of insights they gained. We also gathered feedback about our approach for reflection and discussion.

5.1 Sample

For our evaluation we recruited 3 target users without any knowledge of our prototype. They are all working in IT area and have an education in science and technology. As they use visualizations mainly for presentation purposes, their experience with interactive visualizations is limited. For data analysis they only used *Microsoft Excel* [40] to identify duplicates or filter data for management needs.

5.2 Data set

We used an adapted data set from U.S. Geological Survey containing measurements about water quality of San Francisco Bay [48]. For the sake of simplicity we adapted the data so that we only include the following columns instead of the original 26 columns: Timestamp, Station Number, Depth, Chlorophyll, Oxygen Saturation [%], Oxygen, Salinity, Temperature.

The original data set contains measurements since year 1969. For this user study we only provided five years (2012 – 2016) to avoid performance issues. With this limited data we also wanted to support users to understand the underlying data. The given test data set was already annotated with different data quality problem annotations like "Missing value", "Value too high", or "Unexpected Value".

5.3 Task

The tasks were designed with a focus on one or more requirements and try to cover each functionality of our prototype:

- **Task 1: Changes over time**
  Identify measurement stations which are most error-prone. Which stations improved or decreased their measurement quality over the last years?

- **Task 2: Identifying possible causes**
  There is a significant amount of data quality problems within some of the stations. Identify the exact time when these problems occur (Hint: Zoom). Can you find correlations between data quality problems across different stations?

- **Task 3: Annotated Data Quality Problems**
  Identify already annotated data quality problems. Can you identify patterns where or when specific problems occur?

- **Task 4: Statistics Panel**
  Configure the x-axis with the column "Timestamp", as temporal data type with the Granularity "Day of Month" and a Step of 1. Set the y-axis with the column "Temperature", with a step-width of 5. What can you tell about the measurement distribution when you look at the heatmap in the Statistics Panel?

- **Task 5: Statistics**
  Task 5a: Identify the coldest and hottest month.
  Task 5b: Also identify the following statistics of temperature and salinity: minimum, maximum, mean, median, standard deviation.

The first two tasks provide possibilities to identify further data quality problems (R2). Task 3 is designed to evaluate the visualization techniques of already annotated data entries (R1). Task 4 and 5 cover statistical features of *Grooviler* (R4). The requirements to cover specific types of data quality problems (R3) is not covered in this evaluation and will be seperately discussed in the next chapter.

Due to the prototype's flexibility we did not want to restrict users in their exploration. Therefore we encouraged them to take their time for exploring the prototype between or during tasks.

5.4 Procedure

The study took place in a quiet meeting room, in which the developer and respective participant were present. The developer acted as observer for taking notes and as support for logical and technical questions. The test session started with an introduction of the prototype, its GUI, and the test data set used for this session. Afterwards, we conducted a semi-structured interview to learn about their experience with visualization techniques and experience in data analysis. We then used audio recording software and encouraged the test user to think aloud while performing the tasks mentioned above.

After they completed the tasks, we asked about their impression of our prototype, the usefulness of its visualizations and interaction techniques. We also questioned them about possible improvements, missing features or visualizations and what could be improved in general.

5.5 Data Analysis

Our collected qualitative data consists of observation, audio recordings and interview notes. We analyzed it to answer our main research question (RQ1), how interactive VA methods can help to increase data quality of complex time-oriented data.

Users of our prototype make sense of data and its containing quality problems and therefore might gain insights. That is why we wanted to analyze the effectiveness of our prototype visualizing already existing data quality problems (RQ1.1). In order to answer whether our prototype can help to find additional data quality problems (RQ1.2) we also studied insights.

We therefore chose Klein´s five categories [49] for gaining insights, and categorized our qualitative data accordingly:

- **Connections**: These insights happen if a connection of events provides new information. For example, displaying values in different constellations due to various axis-configurations might result in insights.
- **Coincidence**: Coincidence insights result from repetitive patterns without obvious connections. For example, reoccurring non-changing values in the data set might indicate a data quality problem.
- **Curiosity**: A curiosity insight is triggered by a single event instead of repetition. A value outlier might catch a users attention and therefore leads to a curiosity insight.
- **Contradiction**: Specific events causing doubts can result in contradiction insights. Data entries containing unrealistic values could be an example.
- **Creative Desperation**: These insights happen when users require to find new ways out of inescapable traps. For example, changing the visualization´s configuration if no further quality problems can be found.

5.6 Results

All tasks have been solved by all participants with an average duration of 60 minutes. The duration for the pre- and post-interviews took about 50 minutes.

In total, we spotted 56 insights which vary per user between 15 and 24. Users investing more time into exploration gained more insight about the data and its quality problems. One user concentrated on nearly solely task-solving, which resulted in the lowest amount of insights. Another participant understood each task as a starting point and dug into the data set, resulting in the highest amount of insights.

Most of the insights were curiosity insights (37.5%) and contradiction insights (28.6%). Connection insights (14.3%) and coincidence insights (10.7%) played a negligible role. Creative desperation only occurred in 5.4% of the cases.

Most insights were gained with the help of *Grooviler's* heatmap visualization (66,1%). The table panel resulted in 19 of 56 insights (33.92%). By using the statistics panel we only discovered 4 insights (7.1%). Please note that percentages by visualization type sum up to more than 100%. This is because of connection insights which eventually include more than one visualization type.

5.6.1 Connections

Connection insights, with a total of 8, occurred randomly during the test session. We can differentiate between two types of connection insights. The first one is gaining an insight with a connection between different views. For example, selecting a heatmap cell inspecting related data table entries or cross-checking between the histogram view and heatmap view.

The second type results from connections between values, for example a participant noted: "station S18 only measures exactly 27.0° Celsius at that specific day". We discovered 5 connection insights between views and 3 between values.

5.6.2 Coincidence

We found 8 coincidence insights that occurred during any task. Repetitive values over a given time indicate data quality problems. A test user stated: "it seems that the measurement sensor was stuck because of many on-following measurements with 17.1° Celsius temperature".

Another type of coincidence insight resulted from reoccurring annotations of data quality problems. With this, users gained an insight about a specific station which generally was error-prone. All coincidence insights occurred inside the table visualization.

5.6.3 Curiosity

The most insights were curiosity insights with an amount of 21 out of 56 discovered insights. Of those 21 curiosities, 16 (76.2%) were about color differences or cell-rendering inside the heatmap visualization. As an example, a participant stated that "Station S18 is most error-prone due to its dark coloring". Another participant noted: "due to the missing cells, we can see that there are no measurements on the second day of month ever".

3 curiosity insights were gained while exploring the table view discovering colored table entries. 2 insights were gained by inspecting the histogram visualization. A test user stated: “it makes sense that the histogram shows less measurements on 31st day of month, because of months that have less than 31 days (e.g., February)”.

5.6.4 Contradiction

In total, we found 17 contradiction insights. Of those, 14 (82.3%) resulted from color differences or cell-rendering of the heatmap visualization. A participant stated: “temperatures more than 20°C in December seem like a measurement error”. Another participant noted: “there are measurements with more than 100% of oxygen saturation, which seems to be unrealistic.”.

The other 3 out of 17 contradiction insights (17.7%) resulted from values inside the data table causing doubts. A test participant noted: “it seems odd that there are 7 measurements about 0° Celsius between regular 10° Celsius measurements”.

5.6.5 Creative Desperation

Creative desperation insights occurred only 3 times. For example, a participant randomly scrolled through the table and found annotated values. Those were hour outliers which he would not have found because he did only set up the heatmap with date boundaries.

The other 2 creative desperation insights happened while randomly setting up seemingly illogical relations in the heatmap. With this, participants found implausible negative salt values.

5.7 Feedback

Our post-interview delivered us a broad range of feedback about our prototype and its usefulness. We will separate feedback into positive and negative, and group them by the amount of occurrence.

Positive Feedback from every Participant

- **Coloring:** Expressive coloring and dynamic color palettes, so that even marginal values can be made visible (by changing it to a spectral color palette).
- **Heatmap flexibility:** Anything can be set into relation with different grouping or granularity.
- **Overview:** Depending on configuration, the heatmap can give a good overview about the data (e.g., temperature flow over years).
- **Connections:** Looking up parts of the data set by clicking on specific heatmap cells in combination with our table markers.

Positive Feedback from 2 Participants

- **Tidy UI:** Anything is visible, nothing is hidden inside cascaded tabs, or overlapped by other elements. Everything fits on the screen, providing a good overview.
- **Statistics:** Statistical values could be found easily for parts of the data set (e.g., mean salinity for stations over years).
- **Filter Levels:** Filter levels giving users feedback about what they see. Intuitive buttons to remove specific filter levels.

Positive Feedback from 1 Participant

- **Box Plot:** The box plot gives a good overview about the selected sub-set of data.
- **Draggable Columns:** Columns are draggable and can therefore be reordered.
- **Details on Demand:** Tooltips providing the specific numbers behind colored parts of our visualization.
- **Colored Table:** Colored data table cells based on the underlying data quality problem annotation.

Negative Feedback from every Participant

- **Bad Performance:** *Grooviler* was evaluated on a slower personal computer than it was developed on. This resulted in bad performance so that freezes occurred. Users also sometimes performed interactions twice because of the not responding application.
- **Zooming:** Each user missed a functionality to zoom back out of specific zoom/filter levels.

Negative Feedback from 2 Participants

- **Disappearing Tooltips:** Tooltips automatically disappeared after 3 seconds giving users not enough time to process displayed information.
- **Small Descriptions:** The description of heatmap axes have a small font-size which is hard to read.

Negative Feedback from 1 Participant

- **Missing Histogram Interactivity:** One test participant missed the interactivity of our histogram visualizations. He expected them to also highlight respective values inside the heatmap and data table.

- **Confusing Label Padding:** When labels begin to overlap our prototype automatically hides some of the labels. A test user was confused and did not know what type of data he selected due to a hidden label field.
- **Hide Statistics Panel:** One user missed a functionality to minimize the statistics panel.

CHAPTER 6

Critical Reflection & Discussion

Based on our collected qualitative data, we discuss whether the requirements are fulfilled and answer our research questions. Then we discuss possible improvements and conclude our work with benefits and shortcoming of this work.

6.1 Requirement Coverage

In total, *Grooviler* fulfills the requirements which where defined in Section 3.1. Our prototype is possible to display already annotated data (R1) and provides visualization techniques to profile data further (R2) with the help of statistical information (R4). However, our prototype only covers 11 out of 17 data quality problems defined by Gschwandtner et al. [12] (R3).

6.1.1 R1 - Display annotated data

We already described a use case for evaluating already annotated data quality problems in Section 4.2.1. *Grooviler* supports the configuration of annotation occurrences for heatmap rendering and coloring. User can configure the calculation for any annotation type or filtering them by a specific type.

Figure 6.1 shows an example of this visualization type. In our evaluation we specified Task 3 to cover the visualization and exploration of already annotated data (see Section 5.3). Users inspected the heatmap visualization colored with annotation count with all annotations and only for specific types. As we can see in Figure 6.1, generally 2016 has a lower amount of data quality problems. Unexpected values occurred only in the years 2014 - 2016 and too low values were often produces by station number S18 in year 2014.

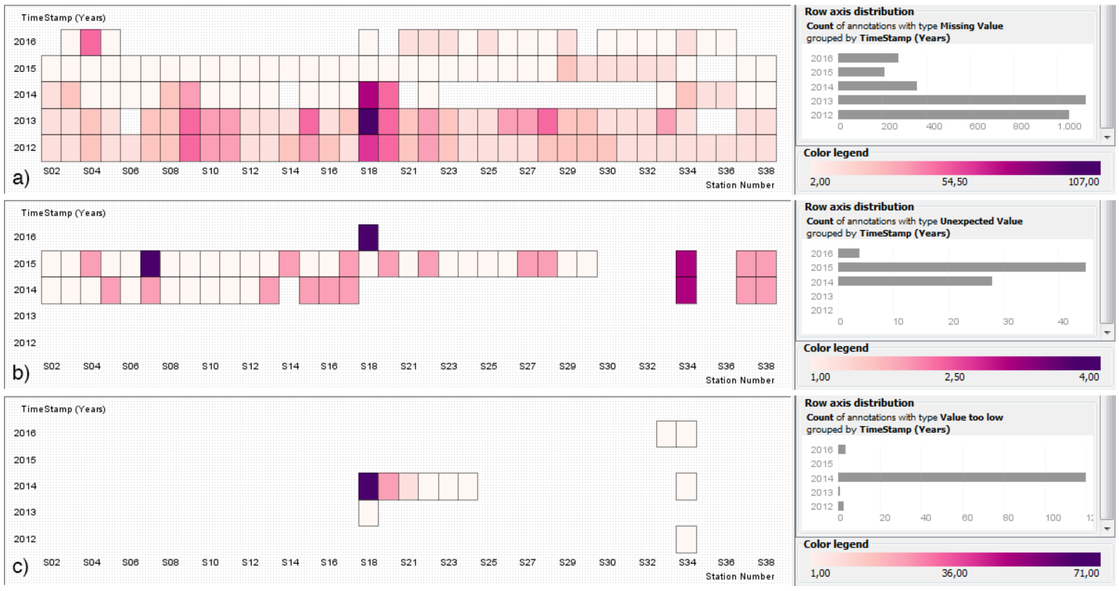

a)
b)
c)
TimeStamp (Years)
Station Number
Row axis distribution
Count of annotations with type Missing Value grouped by TimeStamp (Years)
Count of annotations with type Unexpected Value grouped by TimeStamp (Years)
Count of annotations with type Value too low grouped by TimeStamp (Years)
Color legend
2,00
54,50
107,00
1,00
2,50
4,00
1,00
36,00
71,00

Figure 6.1: Example of different heatmap visualizations with data quality annotations in *Grooviler*. The heatmap shows annotated data quality problems of water measurements from station numbers in relation to year for any annotation type (a), only of type “Unexpected Value” (b) and only of type “Value too low” (c).

Our test participants then inspected specific data quality problems further by clicking on a specific heatmap cell. With this, the respective data was highlighted inside the data table to get analyzed further.

One of our test users noted that it would be useful to display a textual statistic about annotation distribution. This way one can get a coarse overview about the data quality of the inspecting data and gets an idea where to start inspecting.

6.1.2 R2 - Data Profiling

Identifying further data quality problems in addition to annotated data was exemplary shown in Section 4.2.2. In this use case we identified missing tuples and domain violations in form of outliers.

Task 1 and task 2 of our evaluation aimed at Data Profiling with the help of *Grooviler's* heatmap and its navigation functionality (e.g., zoom, linking). With our heatmap visualization, test participants discovered unexpected values in form of repetitive measurements of 27° Celsius from a stuck sensor. They also identified unexpected high values of Oxygen. Domain violations of unexpected temperatures in December and April were also identified. Missing tuples and missing values were also discovered by all test participants. The spectrum of possible further identifiable data quality problems is covered with R3.

All of our test users stated in our post-interview, that the combination of heatmap visualization with statistics and linkage to raw data helps to identify further data quality problems. One user explains: “By experimenting with different heatmap visualization settings, sometimes a specific pattern catches your attention indicating data anomalies”.

6.1.3 R3 - Data Quality Problem Coverage

This requirement is about covering time-oriented single source data quality problems by Gschwandtner et al. [12]. We will now evaluate each type, whether it can be displayed as annotated data (R1) or can be found with the help of *Grooviler’s* InfoVis (R2).

In total, *Grooviler* supports profiling of 10 out of 16 single source data quality problems (62.5%). 4 data quality problems can not be handled due to technical limitations of *Prefuse* (25.0%). Each data quality problem will now be discussed in detail with the following structure:

Data Quality Problem

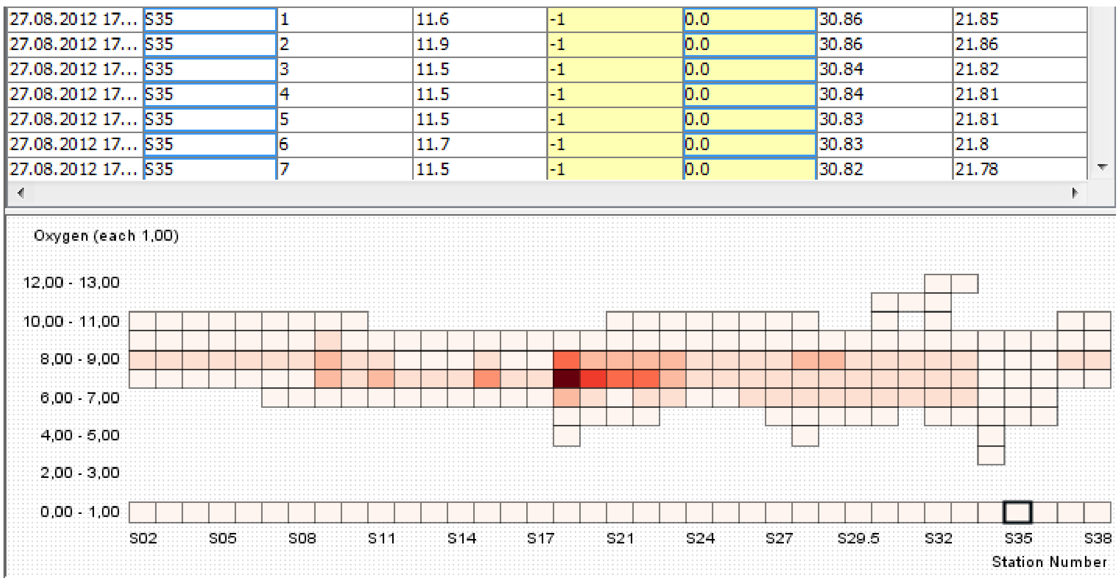
- **Annotations:** Does a specific data quality problem annotation type exist for this specific data quality problem (R1)? If yes, it will be described with a “Description” and (`JAVA_ENUMERATION_KEY`).
- **Grooviler Support:** Is our prototypical InfoVis able to find this specific data quality problem (R2)?
- **Example:** Optional example to display the possibility of our prototype.

Missing data: Missing value

- **Annotations:** Yes.
  Missing values get automatically annotated with “Missing Value” (`MISSING_VALUE`) at the start-up of *Grooviler* and *QualityTime*.
- **Grooviler Support:** Yes.
  Prefuse initializes missing integer values to -1, decimal values to 0.0 and missing date values to 01-01-1970. When the underlying values differ from 0 they can be easily spotted inside the heatmap visualization.
- **Example:**
  Figure 6.2 shows missing values of Oxygen in relation to measurement stations. Since Oxygen was normally recorded with values above 5, missing values concentrate at the bottom of the heatmap.

Missing data: Missing tuple

- **Annotations:** No.



27.08.2012 17...	S35	1	11.6	-1	0.0	30.86	21.85
27.08.2012 17...	S35	2	11.9	-1	0.0	30.86	21.86
27.08.2012 17...	S35	3	11.5	-1	0.0	30.84	21.82
27.08.2012 17...	S35	4	11.5	-1	0.0	30.84	21.81
27.08.2012 17...	S35	5	11.5	-1	0.0	30.83	21.81
27.08.2012 17...	S35	6	11.7	-1	0.0	30.83	21.8
27.08.2012 17...	S35	7	11.5	-1	0.0	30.82	21.78

Figure 6.2: Example of missing values in *Grooviler*. Due to *Prefuse's* initialization of missing values with -1 or 0.0 Oxygen can be identified as missing due to its occurrence at the bottom of the heatmap.

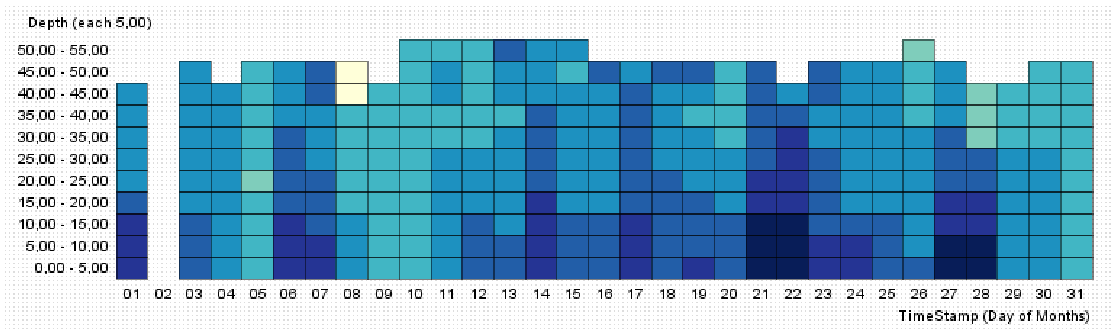
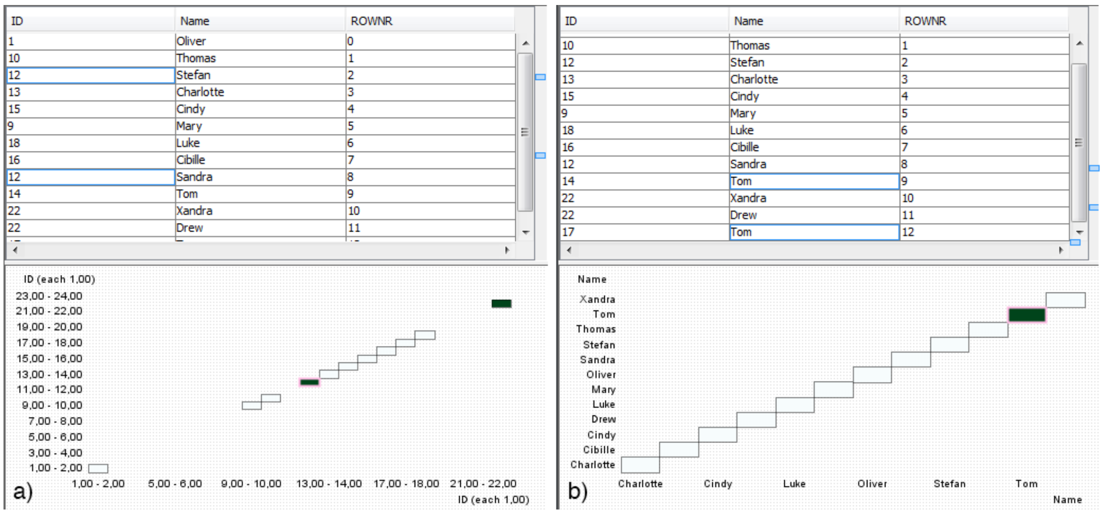


Figure 6.3: Example of missing tuples in *Grooviler*. The heatmap renders water measurements at a specific day of month in relation to depth. At the 2nd day of month no measurement took place, indicating missing tuples.

- **Grooviler Support:** Yes.
  By configuring the heatmap with coloring based on tuple count, users can identify missing values based on missing rendered cells.
- **Example:**
  Figure 6.3 renders measurements of water temperature in relation to depth. At the 2nd day of month no cells get rendered vertically, indicating missing measurements.



ID	Name	ROWNR
1	Oliver	0
10	Thomas	1
12	Stefan	2
13	Charlotte	3
15	Cindy	4
9	Mary	5
18	Luke	6
16	Cibille	7
12	Sandra	8
14	Tom	9
22	Xandra	10
22	Drew	11

ID	Name	ROWNR
10	Thomas	1
12	Stefan	2
13	Charlotte	3
15	Cindy	4
9	Mary	5
18	Luke	6
16	Cibille	7
12	Sandra	8
14	Tom	9
22	Xandra	10
22	Drew	11
17	Tom	12

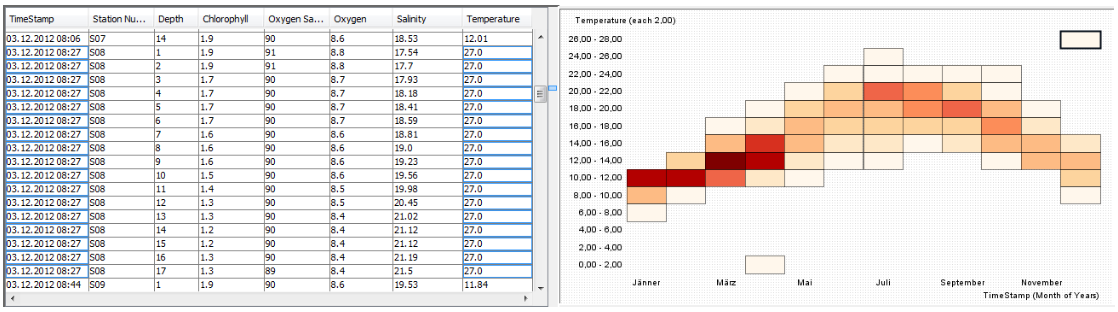
Figure 6.4: Example of Duplicate Detection in *Grooviler*. Duplicates can be identified by divergent colors of numerical values (a) or ordinal values (b).

Duplicates: Unique value violation, Exact & Inconsistent duplicates

- **Annotations:** No.
  QualityTime provides possibilities to cleanse duplicates instead of annotating them. Therefore the existing annotation type "Duplicate" (`DUPLICATE`) will not end up in *Grooviler*.
- **Grooviler Support:** Yes (although not necessary).
  Configuring the heatmap with coloring based on tuple count and setting both axes with a unique attribute should reveal a linear, uniform colored heatmap. Duplicates can then be identified by a divergent coloring.
- **Example:**
  Figure 6.4 shows an example with a small data set containing IDs and names. Setting the unique attributes on each axis shows divergent colors when duplicates occur.

Implausible Data: Implausible range

- **Annotations:** Yes.
  QualityTime provides a check for Intervals resulting in annotated data with type of "Negative Interval" (`INTERVAL_NEGATIVE`).
- **Grooviler Support:** Yes (with help of a workaround).
  Grooviler does not directly support intervals as data type and can therefore not be specifically rendered inside the heatmap. However, by calculating interval lengths as seconds or milliseconds in a separate column, intervals can then be treated as



TimeStamp	Station Nu...	Depth	Chlorophyll	Oxygen Sa...	Oxygen	Salinity	Temperature
03.12.2012 08:06	S07	14	1.9	90	8.6	18.53	12.01
03.12.2012 08:27	S08	1	1.9	91	8.8	17.54	27.0
03.12.2012 08:27	S08	2	1.9	91	8.8	17.7	27.0
03.12.2012 08:27	S08	3	1.7	90	8.7	17.93	27.0
03.12.2012 08:27	S08	4	1.7	90	8.7	18.18	27.0
03.12.2012 08:27	S08	5	1.7	90	8.7	18.41	27.0
03.12.2012 08:27	S08	6	1.7	90	8.7	18.59	27.0
03.12.2012 08:27	S08	7	1.6	90	8.6	18.81	27.0
03.12.2012 08:27	S08	8	1.6	90	8.6	19.0	27.0
03.12.2012 08:27	S08	9	1.6	90	8.6	19.23	27.0
03.12.2012 08:27	S08	10	1.5	90	8.6	19.56	27.0
03.12.2012 08:27	S08	11	1.4	90	8.5	19.98	27.0
03.12.2012 08:27	S08	12	1.3	90	8.5	20.45	27.0
03.12.2012 08:27	S08	13	1.3	90	8.4	21.02	27.0
03.12.2012 08:27	S08	14	1.2	90	8.4	21.12	27.0
03.12.2012 08:27	S08	15	1.2	90	8.4	21.12	27.0
03.12.2012 08:27	S08	16	1.3	90	8.4	21.19	27.0
03.12.2012 08:27	S08	17	1.3	89	8.4	21.5	27.0
03.12.2012 08:44	S09	1	1.9	90	8.6	19.53	11.84

Figure 6.5: Example of Outlier Detection in *Grooviler*. This example renders water temperatures in relation to measurement timestamp aggregated by Month. We can identify a possible outliers in December around 26-28° Celsius (selected heatmap cell) and in April around 0° Celsius.

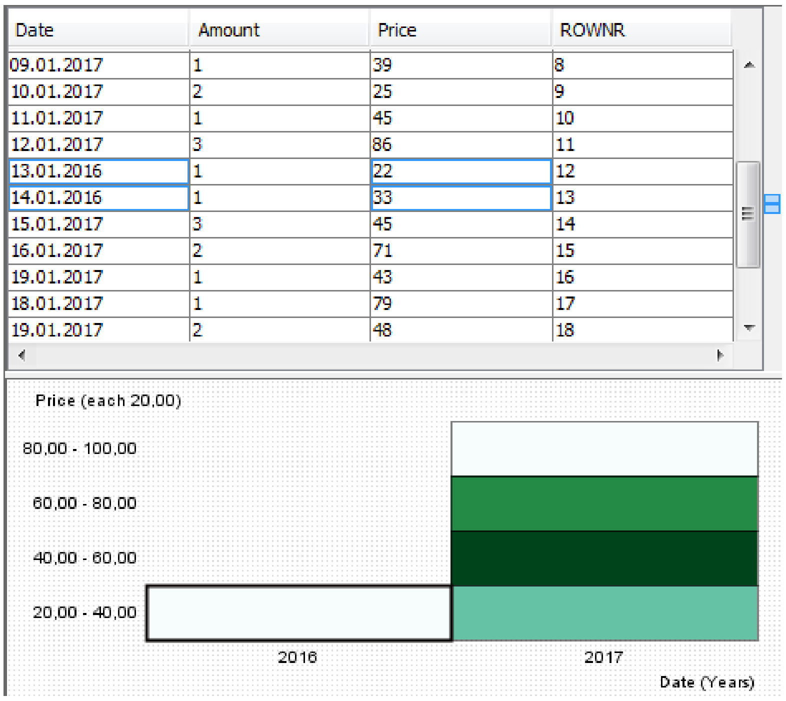
numeric values and addressed properly. Negative intervals can then be identified due to their value less than 0.

Implausible Data: Unexpected low/high values

- **Annotations:** Yes.
  QualityTime supports outlier detection for values and intervals. Based on the results, corresponding data entries get annotated with one of the following annotation types:
  - "Interval too long/short" (`INTERVAL_TOO_LONG/SHORT`)
  - "Interval out of Boundaries" (`INTERVAL_OUT_OF_BOUNDARIES`)
  - "Value too low/high" (`VALUE_TOO_LOW/HIGH`)
  - "Time value out of Boundaries" (`VALUE_INVALID`)
  - "Unexpected Value" (`VALUE_UNEXPECTED`)
- **Grooviler Support:** Yes.
  Our prototype can go beyond the algorithmic functions of *QualityTime*. Additional outliers can be identified in our heatmap visualization by setting values in relation with another value dimension. We can therefore discover outliers which depend on another value (e.g., date).
- **Example:**
  Let us take a look at our evaluation test data set with water temperatures with a mean of 20° Celsius. There are some measurements about 27° Celsius in December. This might not be identified as an outlier due to its adjacency to the mean of 20°C Celsius. With the help of *Grooviler's* heatmap visualization such time-oriented outliers can be identified (see Figure 6.5).

Outdated: Outdated temporal data

- **Annotations:** No.



Date	Amount	Price	ROWNR
09.01.2017	1	39	8
10.01.2017	2	25	9
11.01.2017	1	45	10
12.01.2017	3	86	11
13.01.2016	1	22	12
14.01.2016	1	33	13
15.01.2017	3	45	14
16.01.2017	2	71	15
19.01.2017	1	43	16
18.01.2017	1	79	17
19.01.2017	2	48	18

Figure 6.6: Example of *Grooviler* detecting outdated data of sales.

- **Grooviler Support:** Yes.
  With our prototype outdated data can be found when its identification is based on a date column inside the data set.
- **Example:**
  Figure 6.6 shows a small data set of sales, where outdated sales occur at year 2016.

Wrong data: Wrong data type

- **Annotations:** No.
- **Grooviler Support:** No.

Prefuse initializes data based on its textual representation, and does not allow a manual configuration of data types. Therefore no adequate annotation exists and *Grooviler* is not able to identify wrong data based on a specific type. This kind of data quality problem, however, is handled with “Wrong data format” (see below).

Wrong data: Wrong data format

- **Annotations:** Yes.
  QualityTime Data entries with a wrong data format are automatically getting annotated at start-up with the annotation type “Data Format” (`DATA_FORMAT`). Users

can additionally perform a regular expression check to annotate data with the annotation type "Mismatching Regular Expression" (`REGULAR_EXPRESSION_MISMATCH`). For time-oriented data quality problems, users can let *QualityTime* evaluate and annotate the time raster. This can result in annotations "Raster Missing" (`RASTER_MISSING`) and "Out of Raster" (`OUT_OF_RASTER`).

- **Grooviler Support:** Partially.
  Prefuse initializes numeric integer values to -1 and decimal values to 0.0 when they contain a wrong format. There can then be found inside the heatmap visualization, the same as with missing values (see Figure 6.2). Date values with a wrong format result in a Java Exception breaking the whole *Prefuse* framework.
  Data quality problems based on the amount of occurrences inside a specific raster can be found by inspecting values aggregated by specific date- or time-domain. The structure of a raster (so that values are "out of raster") can not be inspected with *Grooviler*.

Wrong data: Misfielded values

- **Annotations:** No.
- **Grooviler Support:** No.

Misfielded date-values (e.g., day is set in month field) would fail at start-up due to invalid date values.

Wrong data: Embedded values

- **Annotations:** No.
- **Grooviler Support:** No.

Embedded date-values result in Java Exceptions.

Wrong data: Coded wrongly or not conform to real entity

- **Annotations:** No.
- **Grooviler Support:** No.

Neither *QualityTime* nor *Grooviler* provide a functionality to identify whether a given data entry is conform to the real world.

Wrong data: Domain violation (outside of domain range)

- **Annotations:** Yes.
  QualityTime can annotate data entries during domain violation checks:
  - "Value too low/high" (`VALUE_TOO_LOW/HIGH`): When a value is too low or high based on a given check (e.g., calculation based on mean value).

  - "Value unexpected" (`VALUE_UNEXPECTED`): When a value is unexpected due to a given check (e.g., distance-based).
  - "Only Key in Time Unit" (`KEY_ONLY`): When this is the only entry identified with a given key with the only time-instant (e.g., station S18 is the only one containing Month April).
  - "Only Key Missing in Time Unit" (`KEY_ONLY_MISSING`): When this is an entry identified with a given key missing a given time-instant (e.g., station S17 is missing entries in year 1999).
  - "Not all Keys have an entry in Time Unit" (`KEY_NOT_ALL_PRESENT`): Annotates a whole column when keys are missing a given time-instant.
  - "Sequence not monotone" (`SEQ_NOT_MONOTONE`): When an entry is disturbing the monotony of a sequence (e.g., 1, 2, 4, *3*).
  - "Value not a part of defined Sequence" (`SEQ_INVALID`): When an entry is not part of a manually defined sequence pattern (e.g., 0, 2, 4, *7*).
  - "Time between intervals too low (off time)" (`OFF_TIME_VIOLATION`): When pauses between specified intervals are too low (e.g., between Interval A and B must be a pause of 30ms).
- **Grooviler Support:** Yes (depends).
  - Unexpected, low and high values can be identified by setting them in relation with other values (see Figure 6.5).
  - Identifying entries with a given key (e.g., station number) where given time-instants are missing (e.g., Day of Month) is also possible to a certain extent (see Figure 6.2).
  - Sequence checking is not possible with *Grooviler's* heatmap visualization.
  - Since *Grooviler* does not support intervals it is also not possible to visualize pauses between intervals.

Wrong data: Incorrect derived values

- **Annotations:** No.
- **Grooviler Support:** No.

Both tools do not support detection of errors in computing duration.

Ambiguous data: Abbreviations or imprecise/unusual coding

- **Annotations:** No.
- **Grooviler Support:** No.

None of both prototypes can handle ambiguous data due to a short format (e.g., 03-05-06 or 5 minute interval represented as '09:00') due to technical restrictions of *Prefuse.*

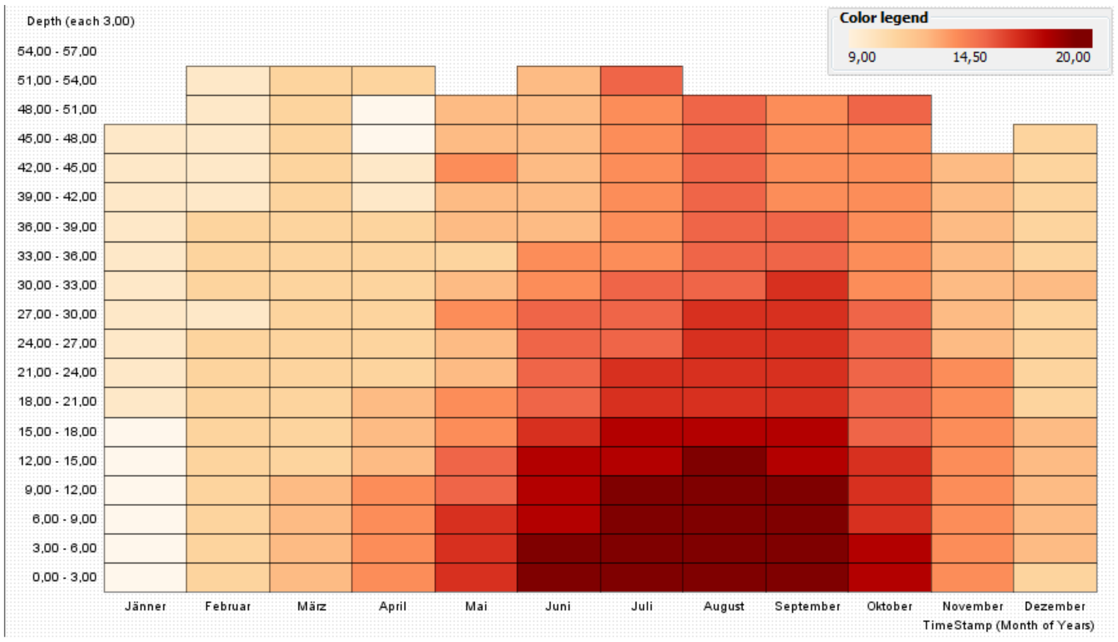


Figure 6.7: Example of a heatmap visualization showing mean temperatures per depth and Month. The color legend gives an idea about the temperature distribution ranging from 9° to 20° Celsius.

6.1.4 R4 - Statistics

This requirement is about providing statistical information about the data set or parts of it. We designed task 4 and 5 of our evaluation to cover statistical tools of *Grooviler*. Section 4.1.3 gives an overview about the GUI containing 3 box plots and 2 histograms. The box plots provide information about the distribution of the users current selected heatmap cell. The histograms give an overview about the value distribution along the respective heatmap axis.

Besides this visualizations *Grooviler* provides aggregation functions for value calculation of heatmap cells. Users can relate values along the axes, while calculating values based on the following aggregation functions: Count, Distinct Count, Median, Sum, Mean, Standard Deviation, Minimum and Maximum. Test participants admired the ability to evaluate mean temperatures of different depths at over the year (see Figure 6.7).

Grooviler does not provide a coarse, textual statistic about the whole data set (e.g., mean temperature over the whole data set), while most of the evaluated profiling tools offer this out of the box. However, there is a workaround by configuring axes so that the heatmap contains only a single cell (and therefore all values) with a given aggregate function.

6.2 Answers to Research Questions

Main Research Question

Grooviler was developed to answer RQ1: *'How can VA methods help to increase the data quality of complex time-oriented data?'*. It covers the ability to display already annotated data quality problems in combination with profiling the data set further. With this, data analysts can examine already annotated data and better reason about the causes of data quality problems, resulting in a better knowledge about the data set and its quality. They can also identify further data quality problems with the help of our interactive heatmap visualization. Our evaluation with a focus on revealing insights showed the usefulness of our approach.

Sub Research Questions

We already covered our research questions RQ1.1: *'How can already detected and annotated quality problems be visualized in an effective and adequate way?'* and RQ1.2: *'What visualizations can help to find additional quality problems of time-oriented data?'* They were covered in our iterative design from the first brainstorming iteration (see Section 3.3.1) to the final evaluation of our prototype Grooviler. We first demonstrated five existing visualization types which can be used to display already detected and annotated data quality problems. Further we discussed eight visualization approaches to support users to identify further data quality problems.

We decided for a combination of visualizations and interaction means that best fit our requirements and implemented these in a prototypical VA solution. To support both use-cases, we decided to use a two-dimensional visualization. Thereby our heatmap visualization can be used to display already annotated data and it provides techniques to identify further data quality problems. Our evaluation shows that users could effectively use our prototype to understand existing quality problems in the data and also to identify additional quality problems.

6.3 Possible Improvements

Besides gaining a lot of feedback during our evaluation and associated post-interviews (see Section 5.7), we also asked test participants about possible improvements of our prototype.

Save Heatmap Configuration

One of the most criticized points was a missing navigation functionality to zoom back out of a specific zoom level. This is however not easily implemented, due to our flexible setting to handle zooming for axes independently. This suggested feature arose from the main problem that users are not able to preserve a desired heatmap visualization configuration state.

As an example, a user configured the heatmap with Station Numbers in relation to *Year* to inspect separate stations. He/She therefore zooms into a specific station and in a certain year to inspect the cause of concrete problems. In the next step he/she wants to inspect a different station. For this, the user needs to reset filter levels and manually change the time granularity from *Month* to *Year*.

To address this issue, users suggested different solutions:

1. *Popout-Zoom*
   One user suggested a zoom that opens a separate window running another instance of *Grooviler* with the settings for the zoomed values. This would certainly impair the current positive feedback about the tidy UI, and that everything fits on a single screen. Another problem is the high performance it takes to process two *Grooviler* instances.

2. *Configuration Presets/Favorites*
   Two users provided the idea to save a configuration state as a kind of preset or favorite setting. This allows users to switch back to specific configurations without thinking of a certain zoom event history.

General Filter

66.7% of our test users seek a possibility to set a general filter on the whole data set. Given our evaluated test data set, users want to filter by a specific station. Then the table and corresponding heat map settings only apply to this specific station which helps to narrow down specific data quality problems.

Users only mentioned use cases for nominal data specific measurement stations. However, filtering could also be done with temporal values (e.g., only year 2015, only Fridays, ...), but also with quantitative value ranges (e.g., only with temperatures between 20.5° and 23.5° Celsius).

Improve Heatmap Selection

Our evaluation showed that users sometimes had difficulties to identify what the user has clicked on an axis. Figure 6.8 shows an example of a selected heatmap cell of Station Numbers in relation to depth. Due to missing space, axis label descriptions get hidden. Hence, it is not visible that the user selected a range between 46.0 - 49.0 and Station Number S18.

There are several approaches to address this issue (see Figure 6.8). The first is to render floating labels of the current selection when the user clicked on it. The second idea is to display a grid line to trace back corresponding axis labels. Finally, a user suggested to format the associated selected axis label with a bold text like in Microsoft Excel.

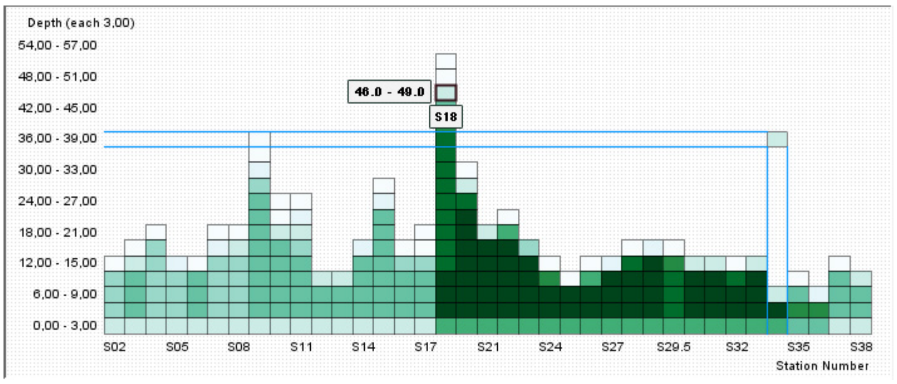


Figure 6.8: Possible improvements for selected heatmap cells in *Grooviler*. The first approach is providing floating labels showing the corresponding axis (middle). Another idea is to provide grid lines to trace back the axis labels (right).

Statistics Panel

It seems that the statistics panel is not as helpful as other visualizations. It only had an effect on 7.1% of all insights (see Section 5.6). One test participant considered the statistics panel so irrelevant, he suggested a functionality to minimize it.

Based on this feedback, we could think of removing the statistics panel. However, these arguments are not sufficient to discard that panel and its statistical visualizations. Task 4 and Task 5 of our evaluation demonstrated possible use cases for this panel. Furthermore, our test users did not possess enough knowledge about the test data set and its domain. Therefore they concentrated more on exploration instead of quantitative statistics. Long explanations were needed to give an idea about what the box plot of a given selection rendered. From our test participants point of view, our box plot visualizations seemed cryptic.

Minor improvements

- **Axis configuration improvement:**
  To improve the usability of the heatmap configuration, we could add a small horizontal arrow to the x-axis configuration. For y-axis configuration we would add a vertical arrow respectively.
- **Nominal data support:**
  Grooviler should be able to display each data type as nominal data. This would result in a pixel-based visualization since each row acts as a single heatmap cell. This feature is implemented but disabled due to performance issues.
- **Remove columns:**
  Users seek a functionality to completely remove a specific column. This would retain a good overview about the data table and improve performance.

- **Improve tooltips:**
  The timeout on which Tooltips automatically disappear should be removed (3 Seconds). Another option would be to display the detailed values inside a status bar at the bottom of our prototype.

- **Export functionality:**
  Test participants suggested an export functionality for still images, but also as SVG images. Another possibility is exporting specific heatmap visualization as HTML containing tooltips.

- **Reset all filters:**
  Our evaluation showed the need of a functionality to reset all filter levels at once. The current prototype only supports selective filter resets.

Experimental Improvements

The following improvements are major design changes of the heatmap visualization. They could result in losing overview about the ordinal scale of underlying data. Therefore we recommend a separate evaluation of these features.

- **Flexible hiding of specific bins:**
  Assuming a heatmap configured with an axis of Days of Week. Users might want to only display working days or weekend days. They could also want to only render Monday, Wednesday and Friday.

- **Display only non-empty heatmap cells:**
  Another possibility is to hide any cells inside the heatmap that contain no values. Imagine a data set ranging from year 2000 to 2005 containing a single entry with year 1970. Our current implementation renders a heatmap with an axis from Year 1970 to 2005 containing 36 cells ([1970, 1971, ..., 2004, 2005], see Figure 6.9 a). In this case we render 30 unnecessary cells because of a single unexpected value ([1971 - 1999]). With only displaying non-empty cells, we would only render 6 non-empty cells (see Figure 6.9 b). This avoids unnecessary stretching of the heatmap visualization, and might increase the overview about data in some cases.

6.4 Conclusion

In this work we developed *Grooviler*, a scientific VA prototype to communicate and identify quality problems of time-oriented data. We conducted a literature research which showed the demand of Data Profiling tools supporting time-oriented data. We concentrated on profiling data quality problems defined in a taxonomy of dirty time-oriented data from Gschwandtner et al. [12]

We built apon a prototypical application named *QualityTime*, which already annotates dirty data entries, which can then be visualized by Grooviler for detailed investigation

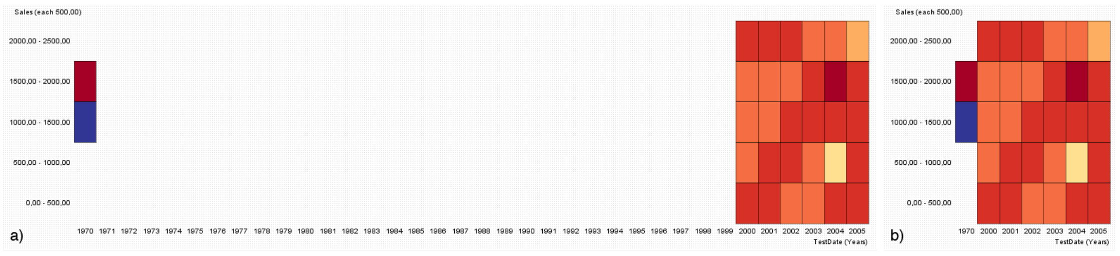


Figure 6.9: Possible experimental improvement of *Grooviler's* heatmap visualization. The current implementation renders cells, even if they are empty (a). This could be improved by only displaying cells holding values (b).

and reasoning. Yet, *Grooviler's* main requirement is to act as a profiling tool to discover further data quality problems. Thus, we designed Grooviler including a two-dimensional heatmap visualization which meets both requirements.

After iterative refinement and implementation, we evaluated the prototype with respect to its potential to identify data quality problems and reveal insights into the data. This evaluation showed very promising results; *Grooviler* was especially appreciated for its flexible heatmap visualization, dynamic coloring, and connection to corresponding data entries. Moreover the evaluation pointed us to interesting possibilities for improvement and further work. We thus demonstrated how to effectively use VA methods to identify and understand complex quality problems of time-oriented data.

APPENDIX A

Appendix

A.1 User Study Material

User Study: Grooviler

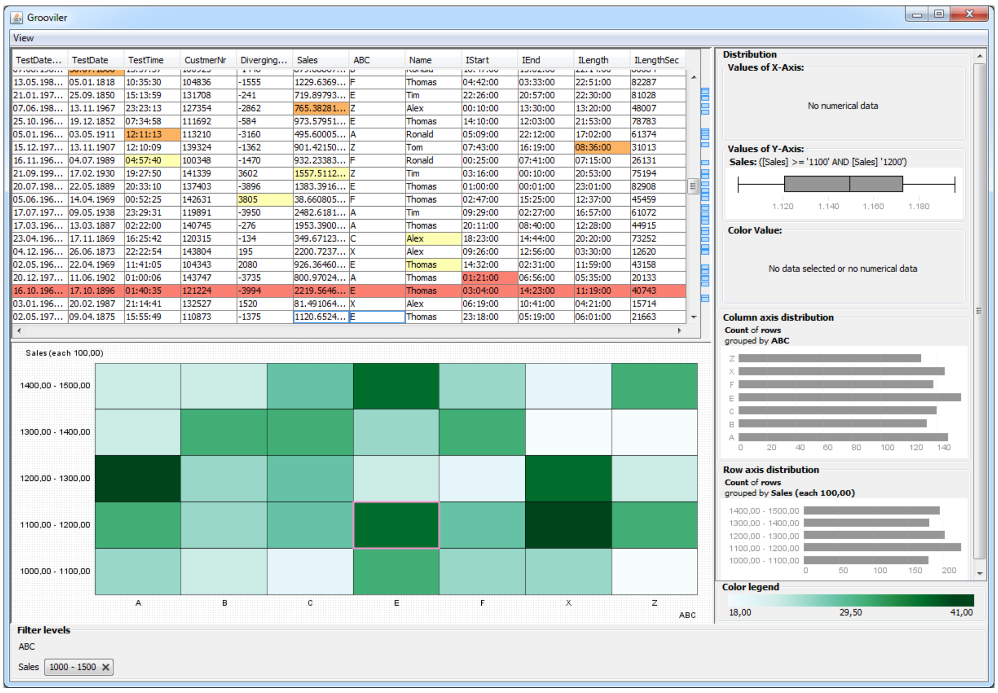
Introduction

Grooviler is an interactive information visualization prototype supporting users by identifying time-oriented data quality problems in two-dimensional data sets (tables).

Grooviler is planned to be a part of a workflow-based framework to cleanse and profile time-oriented data quality problems. In a previous step a prototype named *QualityTime* is used to detect, partially correct and annotate data quality problems with pre-defined techniques. Grooviler should then be used to visualize already annotated data quality problems, as well as providing techniques to identify further data quality problems.

Graphical User Interface

Overview



TestDate...	TestDate	TestTime	CustmerNr	Diverging...	Sales	ABC	Name	IStart	IEnd	ILength	ILengthSec
13.05.198...	05.01.1818	10:35:30	104836	-1555	1229.6369...	F	Thomas	04:42:00	03:33:00	22:51:00	82287
21.01.197...	25.09.1850	15:13:59	131708	-241	719.89793...	E	Tim	22:26:00	20:57:00	22:30:00	81028
07.06.198...	13.11.1967	23:23:13	127354	-2862	765.38281...	Z	Alex	00:10:00	13:30:00	13:20:00	48007
25.10.196...	19.12.1852	07:34:58	111692	-584	973.57951...	E	Thomas	14:10:00	12:03:00	21:53:00	78783
05.01.196...	03.05.1911	12:11:13	113210	-3160	495.60005...	A	Ronald	05:09:00	22:12:00	17:02:00	61374
15.12.197...	13.11.1907	12:10:09	139324	-1362	901.42150...	Z	Tom	07:43:00	16:19:00	08:36:00	31013
16.11.196...	04.07.1989	04:57:40	100348	-1470	932.23383...	F	Ronald	00:25:00	07:41:00	07:15:00	26131
21.09.199...	17.02.1930	19:27:50	141339	3602	1557.5112...	Z	Tim	03:16:00	00:10:00	20:53:00	75194
20.07.198...	22.05.1889	20:33:10	137403	-3896	1383.3916...	E	Thomas	01:00:00	00:01:00	23:01:00	82908
05.06.196...	14.04.1969	00:52:25	142631	3805	38.660805...	F	Thomas	02:47:00	15:25:00	12:37:00	45459
17.07.197...	09.05.1938	23:29:31	119891	-3950	2482.6181...	A	Tim	09:29:00	02:27:00	16:57:00	61072
17.03.196...	13.03.1887	02:22:00	140745	-276	1953.3900...	A	Thomas	20:11:00	08:40:00	12:28:00	44915
23.04.196...	17.11.1869	16:25:42	120315	-134	349.67123...	C	Alex	18:23:00	14:44:00	20:20:00	73252
04.12.196...	26.06.1873	22:22:54	143804	195	2200.7237...	X	Alex	09:26:00	12:56:00	03:30:00	12620
02.05.196...	22.04.1969	11:41:05	104343	2080	926.36460...	E	Thomas	14:32:00	02:31:00	11:59:00	43158
20.12.197...	11.06.1902	01:00:06	143747	-3735	800.97024...	A	Thomas	01:21:00	06:56:00	05:35:00	20133
16.10.196...	17.10.1896	01:40:35	121224	-3994	2219.5646...	E	Thomas	03:04:00	14:23:00	11:19:00	40743
03.01.196...	20.02.1987	21:14:41	132527	1520	81.491064...	X	Alex	06:19:00	10:41:00	04:21:00	15714
02.05.197...	09.04.1875	15:55:49	110873	-1375	1120.6524...	E	Thomas	23:18:00	05:19:00	06:01:00	21663

Distribution

Values of X-Axis:

No numerical data

Values of Y-Axis:

Sales: ([Sales] >= '1100' AND [Sales] '1200')

Color Value:

No data selected or no numerical data

Filter levels

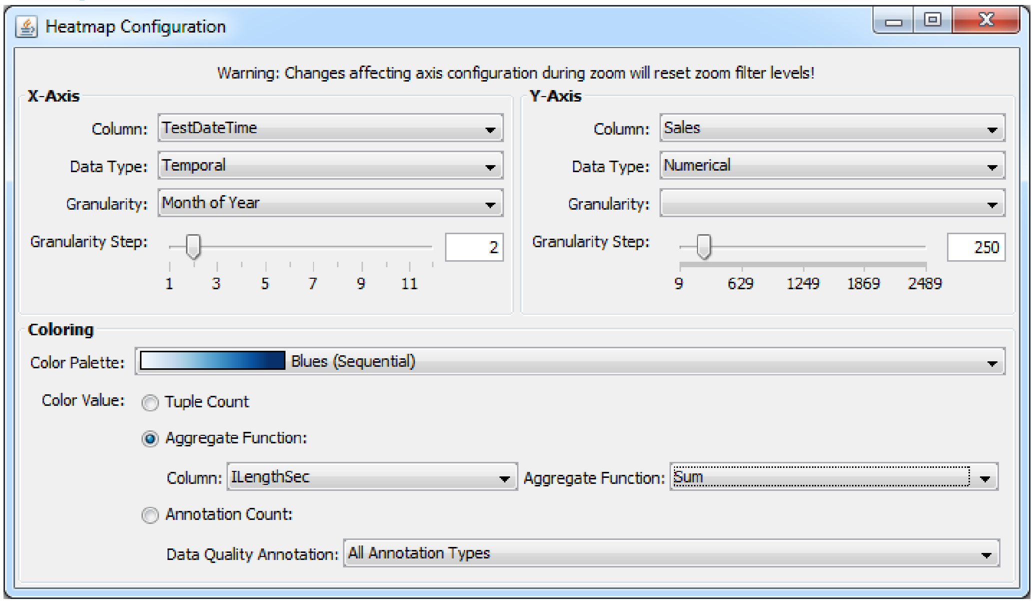
ABC

Sales 1000 - 1500

Figure 1: Grooviler GUI. Table showing the underlying data (top) aggregated in a heatmap (center). Statistics panel (right) including a color legend (lower right). Panel showing the zoom history/filter levels (bottom).

The table at the top displays the data set. With the configuration panel (which can be opened with “View > Heatmap Configuration”) the data will then be aggregated and displayed as a heatmap (Figure 1, center).

Configuration


Heatmap Configuration

Warning: Changes affecting axis configuration during zoom will reset zoom filter levels!

X-Axis

Column: TestDateTime

Data Type: Temporal

Granularity: Month of Year

Granularity Step: 2

Y-Axis

Column: Sales

Data Type: Numerical

Granularity:

Granularity Step: 250

Coloring

Color Palette: Blues (Sequential)

Color Value: ○ Tuple Count

[✓ Aggregate Function:]

Column: ILengthSec Aggregate Function: Sum

○ Annotation Count:

Data Quality Annotation: All Annotation Types

Figure 2: Heatmap Configuration of Grooviler. Configuration of each axis with column, data type, possible granularity and its step (top). Coloring an its calculation function can be set as tuple count, aggregate function, or annotation count (bottom).

The main visualization of Grooviler is its heatmap. With the configuration frame you have the ability to freely configure axis and colored values.

Each axis can be configured with a column that should be used with a specific data type (e.g. temporal, numerical), granularity (if applicable, e.g. year, month) and a specific “step” which can be used to group data.

Configuration examples can be seen in Figure 2. The x-axis is configured with the column “TestDateTime” as a temporal data type, aggregated with the granularity “Month of Year” and a step of 2. That means that the x-axis shows each 2 months of each year. The y-axis is configured with the column “Sales” as numerical data type and a step of 250, which means that the y-axis is grouped in bins containing ranges of 250 (e.g., 1-250, 251-500, …)

The calculation (and therefore coloration) of a heatmap cell can also be freely configured. By default the coloration is based on the tuple count falling into each axis group. You may also define an aggregation function over a specific column falling into each heatmap bin. Another possibility is to colorize the heatmap based on the amount of occurring data quality annotations.

Interactions

Each cell of the heatmap provides two interaction operations:

(1) Left-clicking a cell results in selecting and highlighting a cell.
All containing table rows get highlighted inside the table with an interactive blue marker beside the table to navigate to the corresponding items. Also a box plot is rendered inside the statistics panel on the right side when the particular values are numerical.

(2) Right-clicking a cell displays a context menu which provides zoom operations.
Users can then decide to zoom inside the x- or y-axis or into both dimensions. During zooming an adequate filter is set and displayed at the filter panel (Figure 1, bottom)

Statistics

Depending on configuration, the right panel shows different statistics about what is rendered as a heatmap. If the selected bins x-, y- or coloring-values are numerical a box plot of their value distribution is rendered. A value distribution in form of histograms is always provided for each axis (Figure 1, lower right).

Data Set

In this task you will use an adapted data set from U.S. Geological Survey containing measurements about water quality of San Francisco Bay. For the sake of simplicity we adapted the data so that we only include the following columns instead of the original 26 columns:

Column	Description
Timestamp	Timestamp of measurement.
Station Number	Station Numbers ranging from S1 to S38.
Depth	The depth at which a measurement was carried out.
Chlorophyll	The green pigment of plants, including phytoplankton, which absorbs sunlight energy and converts it to chemical energy used in the process of photosynthesis. Measured in units of milligram per liter.
Oxygen Saturation [%]	Measured oxygen concentration divided by the concentration that would occur if dissolved oxygen was determined only by exchanges with the atmosphere.
Oxygen	Amount of oxygen measured in units of milligram per liter.
Salinity	Salt content of water measured in practical salinity units (psu). River water often contains about 0.1 psu while Pacific Ocean water about 34 psu.
Temperature	Water temperature.

The original data set contains measurements since year 1969. For this user study we only provide five years (2012 – 2016) to avoid performance issues. The adapted data set is already annotated with different data quality problem annotations like: “Missing value”, “Value too high” or “Unexpected Value”.

User Study

Groovilers flexible configuration may help to spot anomalies in the data set which where undiscovered. Different axes and coloration configurations display data in various ways. There is no right or wrong, each task can be solved differently.

If you stumble across data quality problems which may not part of the specific task, please feel free to investigate them.

Task 1: Changes over Time

Identify measurement stations which are most error-prone. Which stations improved or decreased their measurement quality over the last years?

Task 2: Identifying specific causes

There is a significant amount of data quality problems within some of the stations. Identify the exact time when these problems occur (Hint: Zoom). Can you find correlations between data quality problems and their cause (across different stations)?

Task 3: Annotated data quality problems

Identify already annotated data quality problems. Can you identify patterns where or why specific problems occur?

Task 4: Statistics Panel

Configure the heatmap with the following settings and take a look at the histograms:

	Column	Data Type	Granularity	Granularity Step
X-Axis:	TimeStamp	Temporal	Day of Month	1
Y-Axis:	Temperature	Numerical	-	5

What can you tell about the measurement distribution?

Task 5: Statistics

Task 5a: Detailed Statistics

Identify the coldest and hottest month.

Task 5b: General Statistics

Also identify the following statistics of temperature and salinity: minimum, maximum, mean, median, standard deviation *(Hint: Try to configure a heatmap with a single cell containing all values)*

Thank you for participating in the study!

Bibliography

[1] S. Miksch and W. Aigner, “A matter of time: Applying a data–users–tasks design triangle to visual analytics of time-oriented data,” *Computers & Graphics*, vol. 38, pp. 286–290, Feb. 2014.

[2] E. Rahm and H. Do, “Data Cleaning: Problems and Current Approaches,” *IEEE Data Eng. Bull.*, vol. 23, pp. 3–13, 2000.

[3] J. Bernard, T. Ruppert, O. Goroll, and e. al, “Visual-Interactive Preprocessing of Time Series Data,” in *SIGRAD 2012, Interactive Visual Analysis of Data. Proceedings*, 2012, pp. 39–48.

[4] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, “From data mining to knowledge discovery in databases,” *AI magazine*, vol. 17, no. 3, p. 37, 1996.

[5] J. E. Olson, *Data Quality: The Accuracy Dimension*, new. ed. San Francisco: Morgan Kaufmann, Dec. 2002.

[6] H. Galhardas and J. Barateiro, “A Survey of Data Quality Tools,” *Datenbank-Spektrum*, vol. 1, pp. 14:15–21, 2005.

[7] S. Kandel, J. Heer, C. Plaisant, J. Kennedy, F. V. Ham, N. H. Riche, C. Weaver, B. Lee, D. Brodbeck, and P. Buono, “Research Directions in Data Wrangling: Visualizations and Transformations for Usable and Credible Data,” 2011.

[8] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, “Wrangler: Interactive Visual Specification of Data Transformation Scripts,” *Human factors in computing systems. ACM*, pp. 3363–3372, 2011.

[9] R. Verborgh and M. D. Wilde, *Using OpenRefine*, 1st ed. Packt Publishing, 2013.

[10] W. Kim, B. J. Choi, E. K. Hong, S. K. Kim, and D. Lee, “A Taxonomy of Dirty Data,” *Data Mining and Knowledge Discovery*, vol. 7, pp. 81–99, 2003.

[11] P. Oliveira, F. Henriques, and P. Henriques, “A Formal Definition of Data Quality Problems,” *2005 International Conference on Information Quality*, pp. 1–14, 2005.

[12] T. Gschwandtner, J. Gärtner, W. Aigner, and S. Miksch, "A Taxonomy of Dirty Time-Oriented Data," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7465 LNCS, pp. 58–72, 2012.

[13] S. K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vision to Think.* Morgan Kaufmann, 1999.

[14] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon, "Visual Analytics: Definition, Process and Challenges," no. 4950, pp. 154–175, Aug. 2008.

[15] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer, "Profiler: Integrated Statistical Analysis and Visualization for Data Quality Assessment," *Proceedings of Advanced Visual Interfaces, AVI*, pp. 547–554, 2012.

[16] T. Munzner, "A Nested Model for Visualization Design and Validation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 921–928, Nov. 2009.

[17] "Prefuse | Information Visualization Toolkit," http://prefuse.org/, retrieved at May 30, 2015.

[18] "ACM Digital Library," http://dl.acm.org/, retrieved at May 14, 2015.

[19] "IEEE Xplore Digital Library," http://ieeexplore.ieee.org/Xplore/home.jsp, retrieved at May 14, 2015.

[20] "Springer Link," http://link.springer.com/, retrieved at May 14, 2015.

[21] C. Tominski and W. Aigner, "The TimeViz Browser: A Visual Survey of Visualization Techniques for Time-Oriented Data," http://browser.timeviz.net/, Oct. 2016, retrieved at October 7, 2016.

[22] T. Gschwandtner, W. Aigner, S. Miksch, J. Gärtner, S. Kriglstein, M. Pohl, and N. Suchy, "TimeCleanser: A Visual Analytics Approach for Data Cleansing of Time-Oriented Data," in *Proceedings of the 14th International Conference on Knowledge Technologies and Data-Driven Business*, ser. i-KNOW '14. New York, NY, USA: ACM, 2014, pp. 18:1–18:8.

[23] "Data Manager - Data Transformation, Data Cleaning, Data Cleansing," http://datamanager.com.au/, retrieved at May 14, 2015.

[24] "The Premier Open Source Data Quality Solution | DataCleaner," http://datacleaner.org/, retrieved at May 8, 2015.

[25] D. Hoang, "DataLadder - DataMatch 2013 - Best Data Matching software for Deduping," https://dataladder.com/data-matching-software/, retrieved at May 14, 2015.

[26] "Open Studio Integration Software Platform | Talend," https://www.talend.com/products/talend-open-studio, retrieved at May 14, 2015.

[27] "Data Profiling Tool | Datamartist," http://www.datamartist.com/, retrieved at May 14, 2015.

[28] "DataWrangler," http://vis.stanford.edu/wrangler/app/, retrieved at May 15, 2015.

[29] R. A. Becker and W. S. Cleveland, "Brushing Scatterplots," *Technometrisc*, vol. 29, no. 2, pp. 127–142, 1987.

[30] "GitHub: StanfordHCI/profiler," https://github.com/StanfordHCI/profiler/, retrieved at May 16, 2015.

[31] "TalendForge Tutorials," https://www.talendforge.org/tutorials/menu.php, retrieved at May 14, 2015.

[32] "YADC2S: Yet Another DataCleaner 2.0 Screenshot," http://kasper.eobjects.org/2010/12/yadc2s.html, retrieved at May 14, 2015.

[33] W. Aigner, S. Miksch, H. Schumann, and C. Tominski, *Visualization of Time-Oriented Data*, ser. Human-Computer Interaction Series. London: Springer London, 2011.

[34] B. H. S. Havre and L. Nowell, "ThemeRiver: Visualizing Theme Changes over Time," in *Information Visualization, 2000. InfoVis 2000. IEEE Symposium On*, 2000, pp. 115–123.

[35] D. Mintz, T. Fitz-Simons, and M. Wayland, "Tracking Air Quality Trends with SAS/GRAPH," *Proceedings of the 22nd Annual SAS User Group International Conference (SUGI)*, 1997.

[36] W. S. Cleveland, *Visualizing Data.* Murray Hill, N.J. : Summit, N.J: Hobart Press, Jan. 1993.

[37] M. Weber, M. Alexa, and W. Muller, "Visualizing Time-series on Spirals," in *IEEE Symposium on Information Visualization, 2001. INFOVIS 2001*, 2001, pp. 7–13.

[38] T. Lammarsch, W. Aigner, A. Bertone, J. Gärtner, E. Mayr, S. Miksch, and M. Smuc, "Hierarchical Temporal Patterns and Interactive Aggregated Views for Pixel-Based Visualizations," in *2009 13th International Conference Information Visualisation*, Jul. 2009, pp. 44–50.

[39] T. Braunsberger, "Automatic Cleansing Operations of Time-Oriented Data," Master's thesis, Vienna University of Technology, 2016.

[40] "Microsoft Excel 2016 Spreadsheet Software, Excel Free Trial," https://products.office.com/en/excel, retrieved at July 14, 2017.

[41] "MiG Layout Java Layout Manager for Swing and SWT," http://www.miglayout.com/, retrieved at October 24, 2016.

[42] E. Foundation, "Eclipse," https://www.eclipse.org/, retrieved at October 18, 2016.

[43] B. Shneiderman, "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations," in *, IEEE Symposium on Visual Languages, 1996. Proceedings*, Sep. 1996, pp. 336–343.

[44] G. E. Krasner and S. T. Pope, "A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80," *J. Object Oriented Program.*, vol. 1, no. 3, pp. 26–49, Aug. 1988.

[45] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software.* Pearson Education, Oct. 1994, google-Books-ID: 6oHuKQe3TjQC.

[46] M. Harrower and C. A. Brewer, "ColorBrewer.org: An Online Tool for Selecting Colour Schemes for Maps," *The Cartographic Journal*, vol. 40, no. 1, pp. 27–37, Jun. 2003.

[47] R. Mcgill, J. W. Tukey, and W. A. Larsen, "Variations of Box Plots," *The American Statistician*, vol. 32, no. 1, pp. 12–16, Feb. 1978.

[48] J. E. Cloern and T. Schraga, "USGS measurements of water quality in San Francisco Bay (CA), 1969-2015," 2016.

[49] G. Klein, *Seeing What Others Don't: The Remarkable Ways We Gain Insights.* PublicAffairs, 2013.