

Automatische Methoden zur Reinigung von zeit-orientierten Daten

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Business Informatics

eingereicht von

Thomas Braunsberger BSc

Matrikelnummer 0955353

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Univ.Prof. Mag. Dr. Silvia Miksch
Mitwirkung: Mag. DI Dr. Theresia Gschwandtner Univ.Ass.

Wien, 19. April 2016

Thomas Braunsberger

Silvia Miksch

Automatic Cleansing Operations of Time-Oriented Data

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Thomas Braunsberger BSc

Registration Number 0955353

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Mag. Dr. Silvia Miksch
Assistance: Mag. DI Dr. Theresia Gschwandtner Univ.Ass.

Vienna, 19th April, 2016

Thomas Braunsberger

Silvia Miksch

Erklärung zur Verfassung der Arbeit

Thomas Braunsberger BSc
Alaudagasse 5, 1100 Wien

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

Wien, 19. April 2016

Thomas Braunsberger

Danksagung

Danke an meine Freundin für ihre Unterstützung, Geduld und die Mühe die vollständige Arbeit auf Fehler zu überprüfen. An meine Eltern und meine Familie, die mich seit meiner Kindheit an immer unterstützt und an mich geglaubt haben. An meine Freunde und Kollegen, die mich durch das Studium und durch schwierige, langweilige und spannende Übungen begleitet sowie mit mir gelernt haben. Ebenfalls ein großes Dankeschön an Theresia, die immer zur Stelle war, eine Menge konstruktives und positives Feedback gegeben und all meine Fragen schnell beantwortet hat.

Ein spezieller Dank geht an Novak Đoković, der mir stets eine Inspiration ist und bewiesen hat, dass man alles erreichen kann, wenn man nur hart genug dafür arbeitet.

Acknowledgements

Thanks to my girlfriend for her support, patience and the effort of reading through all pages and searching for misspellings and mistakes. To my parents and family, who always supported and believed in me. To my friends and colleagues, who accompanied me through my study and through tough, boring and exciting exercises and courses as well as studying with me. A huge 'thank you' to Theresia, who was always there for me, provided a ton of constructive and positive feedback and answered any questions in no time.

A special thanks to Novak Đoković, who is an inspiration to me and had proven that you can achieve anything if you just work hard enough.

Kurzfassung

Zeit-orientierte Daten sind von großer Bedeutung, da sie beinahe in jedem Datenbestand vorkommen. Sei es nun in Form von Stundenlisten über die Arbeitszeiten der Mitarbeiter oder in detaillierten Listen über die Verkaufsstatistiken eines Onlinehändlers. Wie alle anderen Datensätze neigt auch diese Art von Daten dazu fehlerhaft zu sein. Diese Fehler manuell zu korrigieren würde viel Zeit und Aufwand bedeuten und somit auch hohe Kosten verursachen. Manche Schätzungen besagen sogar, dass bis zu 40% der Daten in einer Datenbank mangelhaft sind [24].

Obwohl es bereits viele Methoden und Tools gibt, um ‘schmutzige’ Daten zu bereinigen, so werden die speziellen Charakteristiken von zeitbezogenen Daten nur selten berücksichtigt. Ansätze können ausgewählte Probleme, die bei zeit-orientierten Daten auftreten, beheben, aber kaum wird Zeit als potentielle Fehlerquelle berücksichtigt.

Daher haben wir einen wissenschaftlichen Prototypen entwickelt, der (halb-)automatische Operationen zur Verfügung stellt um möglichst viele Fehler in zeit-orientierten Daten beseitigen zu können. Die meisten Operationen setzen kein spezielles Wissen über die angewandten Methoden voraus und sind daher für eine breite Masse zugänglich und verwendbar.

In einer Evaluationsstudie haben wir die Nützlichkeit des entwickelten Prototypen untersucht und einige Verbesserungsmöglichkeiten abgeleitet.

Abstract

Time-oriented data are of great importance as they are found in almost any database. May it be in terms of a record of working hours or a detailed list of sales statistics in an online shop. However, as it is the case with any other data these records tend to contain errors and correcting them manually would require a lot of effort and time, and thus, high costs. Some estimations go so far as to say that up to 40% of data contains errors [24].

There are many methods and tools that focus on cleansing ‘dirty’ data, however, they rarely focus on time-oriented data. Some tools may help with a few time-oriented data problems, but time is hardly considered to be the main target. Those, who set a goal to deal with ‘dirty’ time-oriented data are mostly focused on a visual representation to make the task of error detection easier for the user.

This led us to implement a research prototype that provides (semi-)automatic operations in order to take care of many possible time-oriented quality problems. Most of them do not require any further knowledge of the methods applied and hence, are ready to use by a large audience.

We have evaluated the prototype in a usability study and derived suggestions for possible improvement.

Contents

Kurzfassung	xi
Abstract	xiii
List of Figures	xvii
List of Tables	xviii
1 Introduction	1
1.1 General Introduction	1
1.2 Research Questions	2
1.3 Methodological Approach	2
1.4 Structure of the Thesis	3
2 Related Work	5
2.1 General Approach	5
2.2 Cleansing Approaches	6
2.2.1 Conflict Resolution	6
2.2.2 Missing Values	6
2.2.3 Eliminating Duplicates	8
2.2.4 Smoothing Time Series Outliers	9
2.2.5 User Definable Cleansing Operations	11
2.3 Cleansing Tools	12
2.3.1 AJAX	12
2.3.2 Potter's Wheel	13
2.3.3 FraQL	14
2.3.4 Open Refine	14
2.3.5 Profiler and Wrangler	15
2.3.6 ERACER	18
2.3.7 Visual-Interactive Preprocessing (VIP) of Time Series Data	20
2.3.8 TimeCleanser	20
2.3.9 DataCleaner	22
2.3.10 Data Match 2013	23
2.4 Discussion	24

xv

2.4.1	Visual Interactivity	24
2.4.2	Support of Time-Oriented Data	26
2.4.3	Comparison	27
3	Design	29
3.1	Requirements	29
3.2	Design of Cleansing Operations	30
3.2.1	Change Values	30
3.2.2	Transform	31
3.2.3	Edit interval	31
3.2.4	Correct Implausible (Values)	34
3.2.5	Impute Missing Values	36
3.2.6	Deduplication	36
3.2.7	Remove	37
3.2.8	Non-Cleansing Features	38
4	QualityTime	41
4.1	Extension of TimeProfiler	41
4.2	Architecture	42
4.3	Design and Implementation of Cleansing Operations	44
4.3.1	QualityTime User Interface	44
4.3.2	Features	46
5	Evaluation	63
5.1	Operation Coverage	63
5.2	Usability Testing	63
5.2.1	Testing Method	64
5.2.2	Results	65
6	Future Work & Conclusion	67
6.1	Future Work	67
6.1.1	Enhancing Usability	67
6.1.2	Improving Algorithms	68
6.1.3	Data Sources	68
6.2	Conclusion	68
A	Appendix	71
A.1	Dialogs and Parameters	71
A.1.1	Impute Missing Values	71
A.1.2	Correct Implausible Values	77
A.1.3	Deduplication	82
A.1.4	Change Column	85
A.1.5	Add Column	94
A.1.6	Change Rows	95

A.1.7 Sort Rows	96
A.1.8 Edit Intervals	97
A.2 User Study Material	103

Bibliography	107
---------------------	------------

List of Figures

2.1 Relational Dependency Network	8
2.2 Lag in Smoothing	11
2.3 Potter’s Wheel Interface	13
2.4 Potter’s Wheel Interface	14
2.5 Open Refine Interface	15
2.6 Profiler’s interface	16
2.7 Profiler’s Duplicate Detection	16
2.8 Wrangler Interface	17
2.9 Wrangler Editable Natural Language Descriptions	18
2.10 Wrangler Visual Transformation Preview	19
2.11 Visual-Interactive Preprocessing	20
2.12 TimeCleanser’s Interface for Correcting Syntax Errors	21
2.13 DataCleaner’s Interface	23
2.14 Data Match 2013’s Interface	24
2.15 Interactive Selection of a Workflow in VIP	25
4.1 Fisheye Table in TimeProfiler	41
4.2 Structure of QualityTime	43
4.3 User Interface of QualityTime	44
4.4 The ‘Correct Implausible Values’ dialog	47
4.5 Impute Missing Intervals Dialog Example	51
4.6 Preview of Exponential Smoothing	52
4.7 Duplicate Correction Choice	54
4.8 Removing Duplicates Manually UI	54
4.9 Transformations UI	56
4.10 Change Time UI	57
4.11 History Dialog	60
4.12 Help Dialog	61
5.1 ‘Edit Interval’ - Available Operations Extend Across Two Rows	66

A.1	Imputation of Missing Values - Mean Dialog	72
A.2	Imputation of Missing Values - Median Dialog	73
A.3	Imputation of Missing Values - KNN Dialog	74
A.4	Imputation of Missing Intervals Dialog	76
A.5	Correct Implausible Values - Average of Last N Dialog	77
A.6	Correct Implausible Values - Exponential Smoothing Dialog	79
A.7	Correct Implausible Values - Set Null and Impute Dialog	80
A.8	Correct Implausible Values Cumulated Value Outliers Dialog	81
A.9	Deduplication Detection Dialog	82
A.10	Deduplication Method Dialog	83
A.11	Manual Duplicate Removal Dialog	84
A.12	Change Column - Delete Dialog	85
A.13	Change Column - Rename Dialog	86
A.14	Change Column - Split Dialog	87
A.15	Change Column - Merge Dialog	88
A.16	Change Column - Delete Content Dialog	89
A.17	Change Column - Interval Representation Dialog	90
A.18	Change Column - Apply Equation Dialog	91
A.19	Change Column - Change Time Dialog	92
A.20	Change Column - Conditional Equation Dialog	93
A.21	Add Column Dialog	94
A.22	Change Rows Dialog	95
A.23	Sort Rows Dialog	96
A.24	Edit Interval - Standardize Interval Length Dialog	97
A.25	Edit Interval - Group by Date Dialog	98
A.26	Edit Interval - Set Limits Dialog	99
A.27	Edit Interval - Minimum Off-Time Dialog	100
A.28	Edit Interval - Split Intervals for each Day Dialog	101
A.29	Edit Interval - Correct Overlapping Intervals Dialog	102

List of Tables

2.1	Comparison of Different Tools	27
3.1	Problems That Can Be Solved With Calculations	32
3.2	Problems That Can Be Solved With Transformation	33
3.3	Problems That Can Be Solved With Editing Intervals	34

3.4	Problems That Can Be Solved With Correcting Implausible (Values)	35
3.5	Problems That Can Be Solved By Imputing Missing Values	36
3.6	Problems That Can Be Solved With Deduplication	37
3.7	Problems That Can Be Solved With Remove	38
5.1	Errors That Require Manual Actions	64
A.1	Imputation of Missing Values - Mean Parameters	72
A.2	Imputation of Missing Values - Median Parameters	73
A.3	Imputation of Missing Values - KNN Parameters	75
A.4	Imputation of Missing Intervals Parameters	76
A.5	Correct Implausible Values - Average of Last N Parameters	78
A.6	Correct Implausible Values - Exponential Smoothing Parameters	78
A.7	Correct Implausible Values - Set Null and Impute Parameters	80
A.8	Correct Implausible Values - Cumulated Value Outliers Parameters	81
A.9	Deduplication Detection Parameters	83
A.10	Deduplication Detection Parameters	83
A.11	Manual Duplicate Removal Parameters	84
A.12	Change Column - Delete Parameters	85
A.13	Change Column - Rename Parameters	86
A.14	Change Column - Split Parameters	87
A.15	Change Column - Merge Parameters	88
A.16	Change Column - Delete Content Parameters	89
A.17	Change Column - Interval Representation Parameters	90
A.18	Change Column - Apply Equation Parameters	91
A.19	Change Column - Change Time Parameters	93
A.20	Change Column - Conditional Equation Parameters	93
A.21	Add Column Parameters	94
A.22	Change Rows Parameters	95
A.23	Sort Rows Parameters	96
A.24	Edit Interval - Standardize Interval Length Parameters	97
A.25	Edit Interval - Group by Date Parameters	98
A.26	Edit Interval - Set Limits Parameters	99
A.27	Edit Interval - Minimum Off-Time Parameters	100
A.28	Edit Interval - Split Intervals for each Day Parameters	101
A.29	Edit Interval - Correct Overlapping Intervals Parameters	102

Introduction

1.1 General Introduction

There are three common and quite different approaches when it comes to working with complex data and understanding it. Each of them has another purpose and thus, answers different questions. In [15] these methods are compared and their goals are highlighted.

- *Statistics* is mostly concerned with finding an appropriate model for a set of data.
- *Data Mining* deals with finding interesting facts and supports the user in testing hypotheses [24].
- *Information Visualization (InfoVis)* systems are especially helpful when performing exploratory tasks. That means it provides aid when we are simply looking at data without having a specific question in mind, but our goal is to learn more about data and gain new insights. *Visual Analytics (VA)* is tightly coupled with InfoVis systems. While InfoVis systems are focused on “producing views and creating valuable interaction techniques for a given class of data” [34, p. 158], VA methods are designed to combine the human’s visual and perceptual abilities with automatic data processing [34]. Keim et al. provide the definition: “Visual analytics combines automated analysis techniques with interactive visualizations for an effective understanding, reasoning, and decision making on the basis of very large and complex datasets.” [34, p. 157]

All three approaches have one commonality, they rely on the quality of the gathered data. Gupta [24] states that 40% of collected data in warehouses are erroneous, while in [51] it is suggested that one to five % of large datasets are corrupted if an enterprise is taking extraordinary measures to prevent errors. Otherwise error rates can be up to 30%. Odewahn [44] estimates for projects, which analyze data, that about 80% of the time and effort are spent on data cleansing activities.

There are different kinds of so-called ‘dirty’ data as mentioned in [4, 36, 41, 45]. For example, it is possible that certain values are missing or that entries are duplicated. A main source for errors in datasets is the usage of different types of data files [23], for instance from unstructured text files to Excel [40] files and databases.

This work will lay its focus on cleansing time-oriented data. Due to special characteristics it is quite different from other data types, i.e., string and number formats. Time-oriented data may consist of points of time or intervals. Allen [3] describes 13 possibilities of relationships between intervals. These intervals and their relationship can cause problems. For example, a dataset about working hours revealed one employee was working from 9pm to 5am. Thus, not only do we have to check and correct syntax errors, but also keep in mind that intervals must be reasonable. In our example we could say that the main office hours are from 9am to 3pm and it opens at 6am and closes at 9pm. Therefore, it should not be possible to record any working hours between 9pm and 6am.

Data cleansing is also known as data cleaning, scrubbing, reconciliation or wrangling [23, 41]. The work is focused on providing data analysts with a tool that allows user-definable cleansing operations (e.g., through regular expressions). There is existing work that deals with cleansing operations and time-oriented data, however they either lack the flexibility (e.g., no support of user-definable operations or limited interactivity) of different cleansing operations or are not dealing with the problems of time-oriented data.

1.2 Research Questions

The question, which is most important is:

- *How to support data analysts dealing with erroneous time-oriented data?*

Furthermore, we can derive additional questions that arise from that main question:

- *Which methods need to be applied in order to improve the data quality?*
- *Which types of errors can be handled by automatic operations and how to support this task?*
- *Which types of errors need to be handled manually and how to support this task?*

To answer these questions, we decided to integrate means for data cleansing into an existing research prototype that visualizes data and allows error recognition. Furthermore, the prototype will be evaluated to get feedback regarding the usability.

1.3 Methodological Approach

The methodological approach consists of the following steps.

1. *Literature Review.*

At first it is necessary to gather information of existing approaches and programs that will serve as theoretical basis.

2. *State of the Art Report.*

Once the information is available, we will analyze these approaches and programs in order to realize what has been done already. Afterwards, we will discuss this report to find out what still needs to be done, which problems exist and which gaps will have to be filled by this work.

3. *Conceptual Design of VA prototype.*

A theoretical design of what and how to implement it. The conceptual design will provide a clear vision for the technical realization and thus, helps us to save time during the implementation itself.

4. *Implementation.*

This step covers the technical realization of the prototype in Java and will result in a working tool that can be applied in order to automatically cleanse time-oriented data.

5. *Qualitative Evaluation.*

Subsequently, the prototype will be evaluated by performing qualitative evaluation, preferably by a data analyst. This will not only provide answers to research questions, but also necessary feedback in order to complete the following step.

6. *Discussion and Future Work.*

We will critically review the results of the work shown by the evaluation and state future research work within the area.

7. *Conclusion.*

At last a conclusion will be provided, which summarizes what has been done and learned and what the benefits and shortcomings of the work are.

1.4 Structure of the Thesis

The thesis is structured into five chapters in accordance with different steps of the methodological approach.

- **Chapter 2, Related Work, State of the Art:** this chapter will introduce current methods, concept and tools that can be used to correctify ‘dirty’ data. Towards the design and implementation it is necessary to be aware of existing concepts and find out what operations current tools are covering. In the end of the section a comparison between different tools will be made.
- **Chapter 3, Design:** within this chapter the design process will be described. In order to find out the necessary functions that have to be implemented, we need

to know the requirements and data problems we face. The taxonomy provided by Gschwandtner et al. [23] will be considered for operations that can take care of various problems.

- **Chapter 4, Implementation:** The implementation section will show the current version of the prototype as well as give an insight into the implementation of the cleansing operations.
- **Chapter 5, Evaluation:** The evaluation will point out how effective the prototype is in terms of functionality and provide results of a usability study.
- **Chapter 6, Conclusion & Future Work:** The conclusion will summarize the findings in the previous chapters. Furthermore, it will provide an outlook of future work that might follow the thesis.

Related Work

Since data quality problems have been identified and awareness has been raised that such problems easily cost millions of dollars [14], many approaches were made to improve dealing with ‘dirty’ data. We discuss theoretical approaches and tools that support data analysts.

Many errors can be avoided with proper usage of a relational database management system (RDBMS) [4]. In SQL, for example, missing data can be avoided with the usage of the `NOT NULL` constraint or the `CHECK` constraint helps to avoid out of range values. Proper usage refers to the fact that a `NOT NULL` constraint does not necessarily increase data quality, because users may enter dummy values to avoid warnings and errors.

However, more often than not it is necessary to work with files that do not come from an RDBMS, for example Excel [40] files. Thus, we need possibilities to fix ‘dirty’ data since we cannot avoid them completely - even an RDBMS cannot help to avoid inconsistencies or name conflicts [4]. Time-oriented data also introduce further problems that cannot be handled easily without further examination of the data, for instance outdated temporal data or no proper dealing with summer timechange.

Therefore, several tools were introduced that can be used to repair one or more data quality issues (see Section 2.3). Barateiro and Galhardas [4] examine different tools on their generic functionalities, i.e., which type of interface do they use, do they provide versioning and many more. Whereas Müller and Freytag [41] compare five approaches regarding what types of anomalies they can cleanse.

2.1 General Approach

Maletic and Marcus [38] describe a basic approach before starting with data cleansing in general. They mention three phases before the actual process takes place:

1. *Define and determine error types.* For this task Gschwandtner et al. delivered many useful error types in [23] with regard to time-oriented data. Thus, it is known,

which possible errors have to be taken care of within the scope of this work. It does not matter whether a problem is only occurring to single or multiple sources, or to points of time, intervals or time-dependent values. The latter should be understood as events that take place at a certain point of time or within an interval.

2. *Search and identify error instances.* While this is a very important part that finally leads to cleansing itself, it will not be part of this work since this work will be integrated into a larger prototype which already tackles this part.
3. *Correct the uncovered errors.* The next section will provide an overview of existing approaches that aim on correcting these errors. This work aims at providing the user the possibility to specify his/her own cleansing operations and manipulate data within visualizations.

2.2 Cleansing Approaches

In this section we will go into detail of different theoretical concepts of data cleansing. The methods are diverse as some require mathematical understanding while others are rather easy to apply. The applicability of a certain approach depends on the structure of the dataset (e.g., percentage of missing values) or the trend of certain values (e.g., seasonality).

2.2.1 Conflict Resolution

In [47] it is suggested that certain transformation steps are needed in order to apply any further cleansing techniques. They address single-source problems to prepare data for integration with other sources. These steps are the following:

1. *Extracting values from free-form attributes.* Free-form attributes usually contain more than one individual value, for example an address could contain postal code, city, and street. Thus, it is helpful to achieve a more concrete representation (i.e., use three different columns instead of only one), so eliminating duplicates becomes easier.
2. *Validation and correction.* This basically deals with mistypings and tries to correct them automatically. For example, dependencies such as date and birthday can be used to replace missing values or correct wrong values (the age of a person).
3. *Standardization.* This means converting attribute values to a permanent and homogenous format, such as all dates to YYYY-MM-DD HH:mm.

2.2.2 Missing Values

Missing values is a common problem that occurs in many datasets. Possible reasons for them are errors that occur when inserting records due to sloppiness or when merging

different databases. These databases might have different structures and thus, if an entry is only recorded in one database it will miss certain values that are stored in the other database. Some values can be of statistical importance and therefore, the analyst must decide whether they want to replace them. Sometimes it is helpful to leave missing values empty and do not include them into any further research. However, if that is not the case the analyst has to decide which method is the best to deal with missing values. Further problems can emerge when smoothing outliers (see Section 2.2.4) is part of the cleansing process as some methods only work for time intervals of the same length. That means if one interval can not be used due to missing data these methods might not work properly. Acuña and Rodriguez state in [1] that rates of missing values below one % are considered marginal and one to five % manageable. However, when five to 15% of the entries contain missing values one needs to use reasoned methods to deal with them and over 15% influence any kind of interpretation. We will introduce some methods to impute missing values.

Open Refine [29] (see also Section 2.3.4) handles missing values in a very simplified way. Once the data is sorted they simply copy the value from existing entries and paste them into empty fields, the operation is called *fill down*. While this approach is not applicable to numerical fields, it is helpful when dealing with textual data. For example, Student A has written several exams, yet forgot to fill out his first name at one exam. If the entries are ordered by registration number one could simply ‘fill down’ the missing values.

The data mining program WEKA [25] allows the user to perform several cleaning operations, yet with another intention (e.g., classification and clustering of data). However, their approach is intuitive, easy to implement, and more effective than Google Refine when working with numbers. All they do is replacing missing values with the mean of the remaining observations.

Acuña and Rodriguez [1] introduces two further approaches. One of them is quite similar to what WEKA uses. Instead of the mean of an attribute the median is used. This is helpful when outliers should not bias the missing values.

The other one is *k-nearest neighbor (KNN)* imputation. The idea behind this strategy is to group the data into two sets. The first set does not contain any missing values, while the other set does. Afterwards for each item in the second set the distance¹ to all items of the first set is calculated. The missing value is then replaced with the mean of the ‘k-nearest’ attributes. On the one hand this method performs superior to the previous methods as it is very robust. On the other hand there are some disadvantages. Firstly, there are different ways to calculate the distance and the most popular method might not always outperform the others. Secondly, as the algorithm runs through a big part of the dataset it is time consuming, especially if we are dealing with big data. Lastly, the choice of *k* is of importance too. The choice of 10 seems accurate, although a smaller number can be chosen for smaller datasets [1].

¹There are several methods to calculate the distance, i.e., Euclidean, Manhattan, Pearson, etc. [1]. Euclidean appears to be the most popular method.

Acuña and Rodriguez [1] identified KNN to be the best method, especially when the percentage of missing values is high, i.e., up to 20%. The results were even satisfying when about 60% of the instances contain missing values. However, if the percentage of missing values is lower there is not a significant difference between mean, median or KNN imputation. Mean imputation performs worse than the other strategies if there are several outliers in the dataset.

ERACER [39] (see Section 2.3.6) uses *relational dependency networks (RDNs)* to derive missing values. In their article Mayfield et al. provide a relatively easy example of an RDN, which is about sensor networks. A sensor basically measures temperature and humidity, additionally each sensor has neighbors that are placed within a certain distance (meters). They observed errors in about 18% of the data, for instance temperatures of over 100 °C and relative humidity values of -4%. These values were set to NULL and treated like missing values. The resulting RDN of these example can be seen in Figure 2.1.

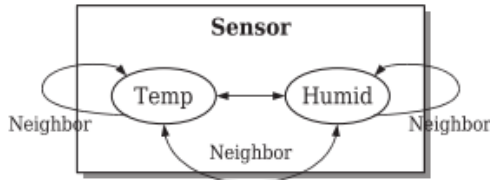


Figure 2.1: An example for an RDN given in [39]. Edges illustrate statistical dependencies. If they are within the plate they symbolize dependencies among random variables of the same tuple.

There are several functional dependencies involved with sensors, for example when the temperature rises it is very likely that also a decrease of humidity can be observed. Thus, they used *linear regression* in order to determine missing values. They modeled the regression in the following way [39]:

$$S.t \sim \beta_0^t + \beta_1^t * S.h + \beta_2^t * avg(N.t) + \beta_3^t * avg(N.h) \quad (2.1a)$$

$$S.h \sim \beta_0^h + \beta_1^h * S.t + \beta_2^h * avg(N.t) + \beta_3^h * avg(N.h) \quad (2.1b)$$

Whereas $avg(N.t)$ and $avg(N.h)$ stand for the average temperature of neighboring sensors respectively for the average humidity. ERACER estimates the coefficients *offline* in R, making use of linear model fitting and then record the resulting estimates in the database [39].

2.2.3 Eliminating Duplicates

At the very beginnings in the field of data cleansing the first approach was to *eliminate duplicates*. This issue is also known as the merge/purge problem [28]. Typically, we face this kind of error when dealing with data from multiple sources. Two other possible error sources are mistypings, such as *Max Musterman* and *Max Mustermann*, or inconsistencies,

i.e., the same name, but different birth dates [19]. Galhardas et al. further describe in [19] the necessary measures to eliminate duplicates:

1. *Mapping*, i.e., standardizing data formats (e.g., dates). If the methods of conflict resolution were not applied already.
2. *Matching*, i.e., finding two different entries by comparing fields that are matching certain criteria.
3. *Clustering*, i.e., generating groups of matching entries with a high likeness.
4. *Merging*, i.e., either removing duplicate entries or creating a new entry. This is applied for every cluster resulting from the previous activity.

With respect to time-oriented data, we will probably have to deal with duplicates of same intervals with the same data. Moreover, there can be inconsistent duplicates, for instance patient A has an appointment at 8am and at 8:30am, which is not plausible [23].

2.2.4 Smoothing Time Series Outliers

There are two categories for outliers described in [17, 27]:

1. *Additive Outliers*, i.e., so-called ‘hiccups’, where in a series one value is an outliers and the time series continues to normality right afterwards.
2. *Innovation Outliers*, which are caused by a single ‘innovation’ [17] and affect not only one value, but also values that follow. An example for this phenomenon is sensors, for example a thermometer that is exposed to a flame. The temperature will rise and slowly drop until it reaches the room temperature again. While the room temperature is constant all the time, the records will state the opposite.

Additive outliers are relatively easy to smooth, for example using the average of the last x Mondays. However, innovation outliers are a different topic. A widespread method to deal with them is the *exponentially weighted moving average (EWMA)* [27]. There are other approaches to calculate the moving average, but we will only have a look on EWMA.

2.2.4.1 Exponentially Weighted Moving Average

EWMA, also known as *exponential smoothing*, is mainly used for forecasting (e.g., in [20] or [31]). Since outliers have an influence on forecasting methods, smoothing them is necessary to get appropriate results. While we are not interested into forecasting, we can apply similar methods in order to smooth the curve shown in visualizations. Basically, there are three kinds of exponential smoothing [21, 31, 56], namely simple [7], double, and triple exponential smoothing [57].

The difference between these three approaches is that double exponential smoothing allows trends and triple exponential smoothing allows both trends and seasonal outliers [31].

Simple Exponential Smoothing [7, 31, 56]:

$$S_t = \alpha * X_t + (1 - \alpha) * S_{t-1} \quad 0 < \alpha < 1 \quad (2.2)$$

Simple exponential smoothing was first proposed by Brown in [7]. The equation is shown in 2.2. It takes both the current observation (X_t) and the previous statistic (S_{t-1}) into account and calculates a new statistic (S_t). The new value replaces X_t , thus, outliers are smoothed. α is the weight assigned to the current observation, i.e., if it is zero the current observation has no effect on S_t and is completely replaced with the smoothed value that results of previous statistics. The selection of α and S_t is a challenge and has a strong influence on the outcome of the smoothing [31]. While α has a range from 0 to 1 in best practice only values between 0.1 and 0.3 are used, as mentioned in [20].

Double Exponential Smoothing [31, 56]:

$$S_t = \alpha * X_t + (1 - \alpha) * (S_{t-1} + b_{t-1}) \quad 0 < \alpha < 1 \quad (2.3a)$$

$$b_t = \beta * (S_t - S_{t-1}) + (1 - \beta) * b_{t-1} \quad 0 < \beta < 1 \quad (2.3b)$$

2.3a does not look too different from the single exponential smoothing equation. However, b_t (calculated in 2.3b) is used to smooth the underlying trend of the data and thus, β is the smoothing factor of the trend.

Triple Exponential Smoothing

Triple exponential smoothing is needed because time series can exhibit unique characteristics and one of them is seasonality. An example for seasonality is that sales of toys increase every year in November and December. The reason for this increase is Christmas. After Christmas the sales go back to their normal level. We differentiate between *additive* and *multiplicative* seasonality [31].

We speak of *additive seasonality* when the sales increase by one million dollar each year. The term *multiplicative seasonality* is used when there is a factorial increase every year, i.e., the sales increase by 40% and therefore we use the factor 1.4 [31]. Multiplicative seasonality might vary a lot, because it depends on the previous sales of the year while additive seasonality is independent of such figures.

Formula [46, 56, 57]:

$$S_t = \alpha * \frac{X_t}{I_{t-L}} + (1 - \alpha) * (S_{t-1} + b_{t-1}) \quad 0 < \alpha < 1 \quad (2.4a)$$

$$b_t = \beta * (S_t - S_{t-1}) + (1 - \beta) * b_{t-1} \quad 0 < \beta < 1 \quad (2.4b)$$

$$I_t = \gamma * \frac{X_t}{S_t} + (1 - \gamma) * I_{t-L} \quad 0 < \gamma < 1 \quad (2.4c)$$

The important change here is the equation 2.4c and the addition of the seasonal index (I_t) in the general equation 2.4a. Each season is a cycle of the length L . In order to

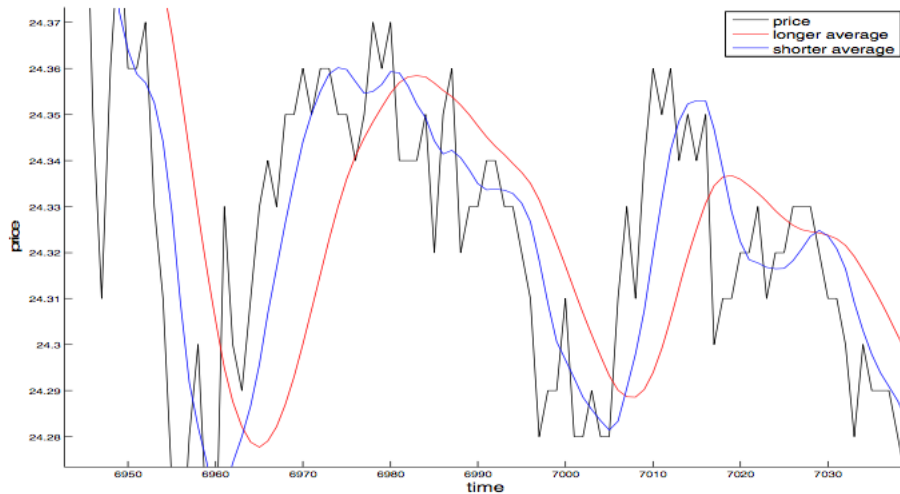


Figure 2.2: We can observe a lag between the real-world data (black) and the moving averages (red and blue). However, the (red) MA with the greater interval is smoother and more lagging, while the other (blue) with a shorter interval is less lagging and more bumpier [50].

consider seasonal influences at least one season of data is needed. Furthermore the trend factor between periods has to be calculated, so it is advisable to use two complete seasons ($2L$) before starting the smoothing process [46].

Raudys et al. in [50] stated that triple exponential smoothing is a very effective method to smooth data as its smoothness/lag ratio is one of the best. Lag is problematic as the observations at time t are displayed at a later point of time ($t + lag$) (see Figure 2.2). This leads to a wrong interpretation of data, for instance sales before Christmas could be displayed as sales afterwards and thus, may lead to the wrong assumption that a lot of people are spending money before New Year's Eve.

2.2.5 User Definable Cleansing Operations

Regular expressions are the most common way to allow users to define his/her own operations, especially for String manipulation and filtering. However, they are limited in their use and not easy to understand for non-programmers.

FraQL allows the user to write his/her own functions in Java as described in [52] (also see Section 2.3.3). Thus, the use of such functions can overcome limitations posed by query languages.

Google introduced their own expression language (GREL) in their software *Google Refine* (now known as *Open Refine*, see Section 2.3.4) [29]. Operations, which are definable by the user are mostly used for transformations. Nonetheless, they are also helpful in the process of data cleansing, in particular for eliminating duplicates. Different date

formats may convey the impression that two entries are not duplicates depending on the clustering rules. However, the language also has its limitations, for instance, it is not possible that operations include numbers of another column. This, for example, means one cannot calculate the sum of two different columns.

Wrangler [32] (see Section 2.3.5) allows the user to apply natural language descriptions. Each other language or tool requires knowledge of programming languages or expressions (although GREL provides a documentation and is quite intuitive to use). Additionally, Wrangler suggests which operations to use, for example when the user selects a row, such as `delete row 11` or `delete empty rows` (this suggestion is shown if row 11 is empty).

2.3 Cleansing Tools

This section will provide a closer look at different approaches, scientific ones as well as commercial tools, and explain their functionalities. They implement one or more cleansing techniques introduced in Section 2.2 to provide data analysts with tools to apply them.

2.3.1 AJAX

This system was mainly provided to support transformations in order to eliminate duplicates. “AJAX provides an expressive and declarative language for specifying data cleaning programs.” [19, p. 2] This language can be compared to SQL statements extended with certain macro-operations (primitives) for *mapping*, *matching*, *clustering*, and *merging* (see Section 2.2.3). Whenever the automatic transformation experiences difficulties, i.e., a dubious cluster during the merging operation, a human expert is required to take care of the conflict. Additionally, AJAX is an *extensible framework*. That means, it allows customization in various ways, for instance more complex data cleaning programs can be formulated by combining pre-defined macro-operations with pure SQL statements. Also functions written in Java can be included in the program, for example to extract certain Strings out of a database attribute [13]. A metadata repository allows the back-tracking of activities, i.e., which transformations were applied and what were the existing results [19].

An example given by Galhardas in [18] is the following match operation that matches authors with similar names with the goal to find possible duplicates:

Listing 2.1: Matching Operation in AJAX

```
CREATE MATCH MatchDirtyAuthors
FROM DirtyAuthors a1, DirtyAuthors a2
LET distance = editDistanceAuthors(a1.name, a2.name)
WHERE distance < maxDist
INTO MatchAuthors
```


Delay	Carrier	Source	Dest..	Date	Day	Dept_Sch	Arr_Sch
-12	TWA	JFK	STL	1997/10/17	F	17:30	19:18
0	TWA	ORD	STL	1997/07/28	M	12:25	13:36
-26	TWA	JFK to MIA		1998/12/04	F	09:40	12:40
-3	TWA	JFK to MIA		1997/12/30	Tu	07:30	10:36
-5	TWA	ORD	STL	1997/06/08	Su	15:05	16:17
2	TWA	JFK	MIA	1998/09/21	M	07:25	10:25
3	TWA	ORD	STL	1998/07/02	Th	11:20	12:30

Figure 2.3: Tabular layout used by Potter’s Wheel [49].

`editDistanceAuthors()` is a function that can be specified in Java. This allows different calculations of the distance. Dos Santos Dias defined two roles in [13]. The first role is the designer, who writes both SQL statements and necessary Java classes. The second role is the user, whose task is to execute the data cleaning process by defining which cleansing operations to use and when. AJAX provides a GUI for the user, which allows them to view a graphical representation of the data cleaning graph, i.e., which steps are performed at a certain time. Furthermore, the GUI supports browsing the data to analyze initial, intermediate, and final data and debugging of the cleaning process [13].

2.3.2 Potter’s Wheel

As described in [48, 49] Potter’s Wheel allows the resolution of conflicts. It provides several features in order to fix the problems that arise when working with free-form attributes. To support the user it shows a small exemplary fraction of the data collection (see Figure 2.3). Each transformation rule is visualized on this example and thus, gives the user the chance to see the effects of their change.

The supported transformation actions are format, add, drop, copy, merge, split, divide, fold, and select. In Figure 2.4 we see different actions performed on a dataset. On the left-hand side a divide action is performed, which simply divides columns that contain a comma from columns that do not. Afterwards a split action could be executed to convert *Such,Bob* into two separate columns *Such* and *Bob*. A merge operation could be used to merge *Such* in the column of *Ann* and *Bob* in the column of *Davis*. In the middle formatting, folding, and splitting is demonstrated on the example of fixing higher-order variations. On the right-hand side we can see how unfolding works. All operations can be exported either as C or Perl program or a Potter’s Wheel macro. This allows the execution on larger sets as transformations applied on the sample seen in the GUI are not invoked on the rest of the data. The macros can also be used on datasets that are different from the actual one.

Furthermore, Potter’s Wheel offers a discrepancy detection, i.e., if a column is specified as `[A-Za-z]* [0-9]*, [0-9]*` it only allows dates in the form of *April 20, 2015*. Other formats will be highlighted as wrong. Besides defining a customized format there are a number of formats available to use, such as *Names*, *Money*, or *ISpellWords* (English

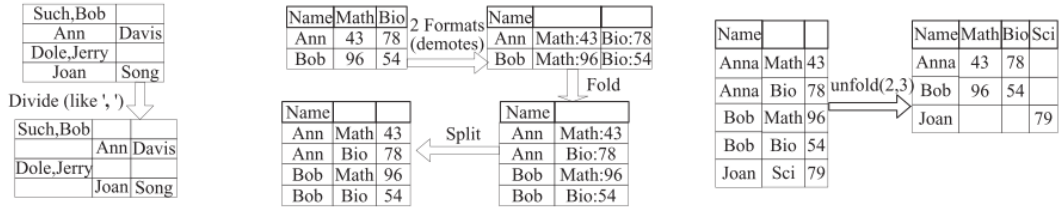


Figure 2.4: Examples for the supported transformation actions. [49].

words that are checked with iSpell). These formats are defined by using an interface provided in Java. Hence, some methods can be extended to do more than simply check for errors - for example `updateStats` could be used to eliminate duplicates by using hash tables [49].

2.3.3 FraQL

FraQL is a multidatabase query language and enables a user to use query operations for the integration and transformation of heterogeneous data [53]. Its main goal is the resolution of integration conflicts [52] and supporting the data mining process (as described by Han et al. in [26] and Chapman et al. in [10]) [53]. The data mining process does not only require data transformed into an appropriate format, but also cleansed data. Thus, FraQL provides operations for transformation and also a framework that supports the elimination of duplicates, dealing with missing values, detecting outliers, and handling noise in data. Nevertheless, the language still requires to write code (in Java) to make use of these operations. The language makes the task of data cleansing easier, but demands knowledge of SQL, programming, and methods to get rid of ‘dirty’ data. Another drawback is that the language runs on a special query processor.

2.3.4 Open Refine

Open Refine [29], formerly known as Google Refine, is an open source project for data transforming and cleansing. The interface is kept in the style of a spreadsheet (see Figure 2.5). Operations such as filtering, transforming, clustering, and merging are provided. They are either predefined or users themselves can specify the operations. Specifications can be entered as regular expressions or as *Google Refine Expression Language (GREL)*. GREL supports users by providing a syntax that reminds of a programming language. For example, users can convert different date formats into one consistent format by using the expression:

```
value.toDate('MM/yy', 'MMM-yy').toString('yyyy-MM').
```

Data functions are one part of the expression language, while other prominent functions are Boolean, String, or Math. Google Refine also partly allows the usage of regular expressions within GREL (replace, match, partition, rpartition, and split functions).

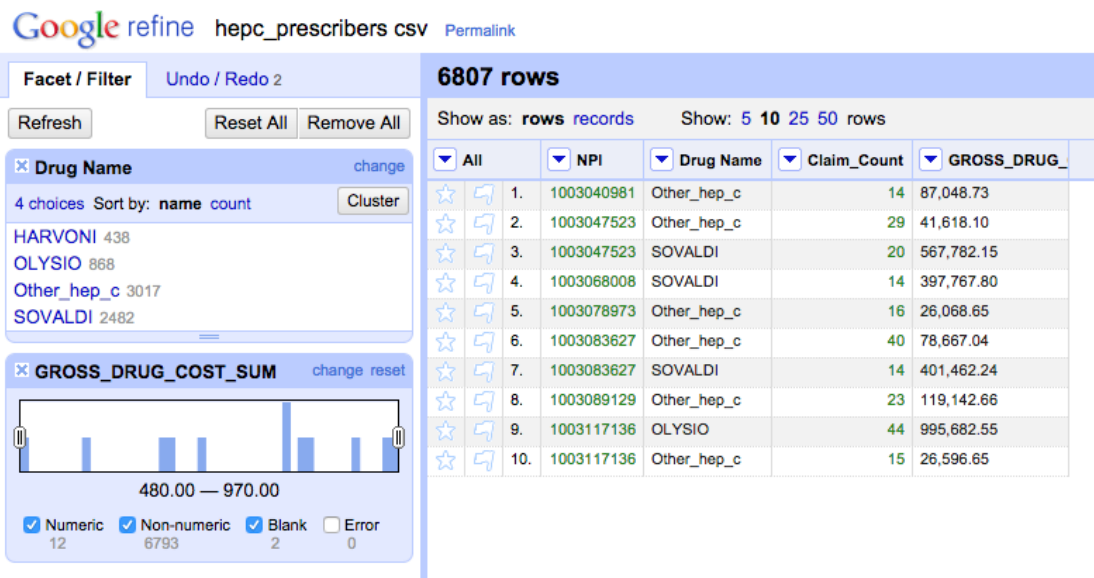


Figure 2.5: Open Refine Interface [29]. On the right side the data is displayed and on the left side different values of a column are shown. For numeric values a histogram is provided.

However, Google Refine is only able to handle file based data, i.e., data cannot be imported from databases directly.

2.3.5 Profiler and Wrangler

Profiler was introduced in [33] and offers a variety of visualizations in order to detect data quality problems. Besides, it offers an anomaly detector to support the user. While some issues are clear errors, such as missing values, extreme values are not and thus, visualizations are needed. Moreover, visualizations foster to recognize the reasons for erroneous data, for instance see Figure 2.6. The user can decide which fields they want to examine closer. This can be done by double clicking in the *Schema Browser* or dragging them from there into the visualization form. Clicking on the Motion Picture Association of America (MPAA) Rating in the *Anomaly Browser* causes the highlighting seen in the example.

Eliminating duplicates is supported by a short overview of duplicates that can also be filtered (see Figure 2.7). This avoids removing duplicates, which are none and the filter operation allows the user to reduce the number of recognized duplicates significantly. If any of the duplicates are real ones the user can remove them - instead of forcing the merge operation.

Wrangler was described in [32] by Kandel et al. The tool is used for interactive data transformation and uses Profiler to detect quality problems. The operations are expressed

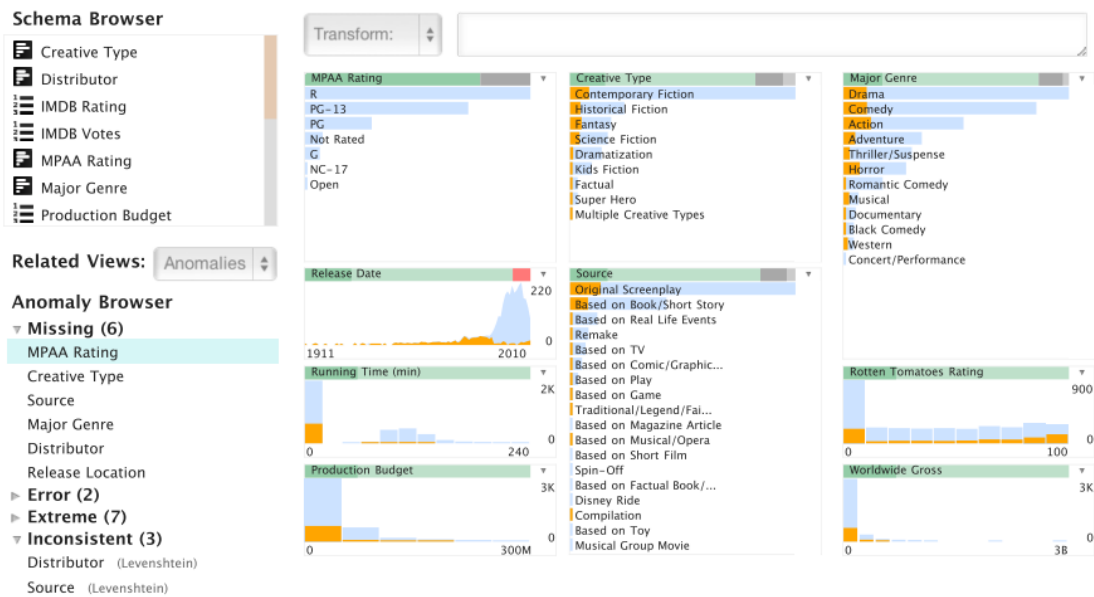


Figure 2.6: Profiler's interface. Here it is shown that missing values in the MPAA Rating fields (marked orange within other attributes) highly correspond to an older release date [33].

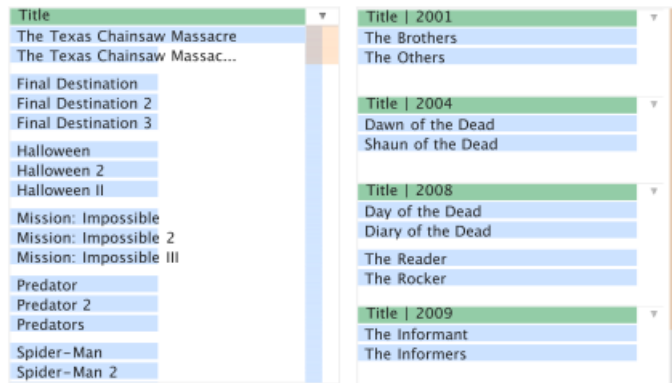


Figure 2.7: Profiler's Duplicate Detection. On the left-hand side all movie titles clustered by Levenshtein distance can be seen (over 200). On the right-hand side we see all duplicates where the year is added to the detection algorithm (only 10 findings) [33].

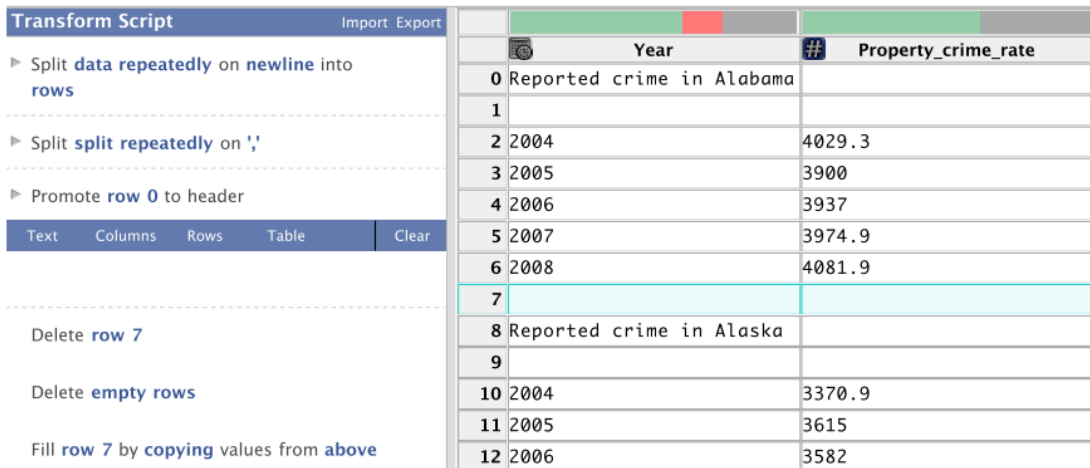


Figure 2.8: Wrangler Interface [32]. The left panel contains three parts. The history of executed transformations, a transform selection menu, and automatically suggested transforms based on the current selection. The right panel visualizes an interactive data table and above each column is a quality meter.

in *natural language descriptions* (see also Section 2.2.5) and it is possible to preview them visually. Both of these features should support the user, novice or expert. The user either types or selects (automatically generated suggestions based on which line is selected) the needed transformation (see Figure 2.8) or if necessary they can undo or modify already executed transformations within the transform history. The transform history can be exported and “run in a web browser using JavaScript or translated into MapReduce or Python code” [32, p. 2].

The functionality of Wrangler can be compared to Potter’s Wheel (Section 2.3.2). However, Kandel et al. in [32] did only use Potter’s Wheel as a starting point and extended the language afterwards. They added the following new features [32]:

- *Map*: maps one row of input data to zero, one or multiple output data rows.
- *Lookups and joins*: with Wrangler one can integrate data from external sources. For example, if only ZIP codes are known one can join them with other databases to obtain the state or city names.
- *Reshape*: allows to manipulate both table structure and schema. The two operators for reshaping are fold and unfold.
- *Positional*: fill and lag are the operators for positional transforms. Fill can be used to write values for neighboring cells in a row or column (e.g., fill empty cells with preceding non-empty values). Lag shifts values of a column up or down by a certain number of rows.

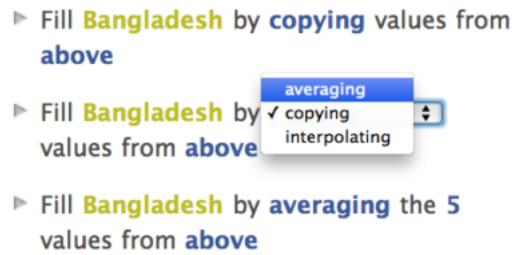


Figure 2.9: Editable Natural Language Descriptions within Wrangler [32]. The fill operator can be altered in different ways. The example shows how the change of a parameter is done. If the user selects moving average instead of copying then an additional parameter has to be specified, i.e., number of values.

- *Other operators:* Besides the operators mentioned above Wrangler also supports *sorting, aggregation, and key generation* as well as *schema transforms*.

The automatic suggestions are administrable in three ways. Firstly, by providing examples, for instance on how to split columns, and the automatic suggestions will adapt to these samples. Secondly, filtering is possible by selecting an operator, i.e., all suggestions will contain only the selected function. Lastly, suggested transformations are editable by clicking on bold written text (see Figure 2.9).

The visual transformation previews are shown in-place, that means within the same panel as the source data. The advantage is that also slight changes are recognizable and that the user does not need to change focus. In Figure 2.10 we can see the preview of an unfold operation on a given dataset. The input data is highlighted in the same way, i.e., color, as the output data.

2.3.6 ERACER

ERACER, presented by Mayfield et al. in [39], is an iterative statistical framework for deriving missing values and correcting them automatically in the environment of RDBMs. Additionally, it provides means to identify and clean potentially corrupt values. The name is an acronym for the major phrases of their framework [39]:

1. *Extract:* with the use of domain knowledge (i.e., model templates) the graphical model structure is built.
2. *RDNs:* learn the parameters for each applicable (RDN) (see 2.2.2 or [42] for further information) component model.
3. *Apply:* every time a variable is modified, the relevant models are applied to update the output distribution ‘message’, which is sent to all related variables (i.e., neighbors) in the next step.
4. *Combine:* aggregate the resulting predictions to deal with heterogeneity.

	Year	State	Property_crime_rate
0	2004	Alabama	4029.3
1	2005	Alabama	3900
2	2006	Alabama	3937
3	2007	Alabama	3974.9
4	2008	Alabama	4081.9
5	2004	Alaska	3370.9
6	2005	Alaska	3615
7	2006	Alaska	3582
8	2007	Alaska	3373.9
9	2008	Alaska	2928.3
10	2004	Arizona	5073.3
11	2005	Arizona	4827
12	2006	Arizona	4741.6
13	2007	Arizona	4502.6
14	2008	Arizona	4087.3

State	2004	2005	2006	2007	2008
0 Alabama	4029.3	3900	3937	3974.9	4081.9
1 Alaska	3370.9	3615	3582	3373.9	2928.3
2 Arizona	5073.3	4827	4741.6	4502.6	4087.3
3 Arkansas	4033.1	4068	4021.6	3945.5	384
4 California	3423.9	3321	3175.2	3032.6	294
5 Colorado	3918.5	4041	3441.8	2991.3	285
6 Connecticut	2684.9	2579	2575	2470.6	249
7 Delaware	3283.6	3118	3474.5	3427.1	359

Figure 2.10: Wrangler’s Visual Transformation Preview [32]. The user selects two columns (Year and Property_crime_rate) and wants to perform an unfold operation. The result is presented to simplify the user’s decision whether they want to execute the action or not.

5. *Evaluate*: for each variable, the previous and current inferred distribution are compared. Depending on the result the changes are accepted or rejected.
6. *Repeat*: until the update count is close to zero.

The user has to define three relations as input for the whole cleaning process [39].

- (a) `model`, which contains the learned parameters for the desired inference functions (e.g., coefficients for regression).
- (b) `node`, which corresponds to vertices and
- (c) `link`, which represents the edges of the graphical model.

`node` and `link` can be both defined using SQL statements. The last step after defining them is to execute the `erace` aggregation query, which delivers the updated version of each node (i.e., “inferred values filled in and/or corrected values identified” [39, p. 80]). The result can be stored in a new table or used to update existing ones. The `erace` query does not automatically repeat the process (as the missing ‘r’ suggests), therefore the user has to execute the query again (multiple times) manually.

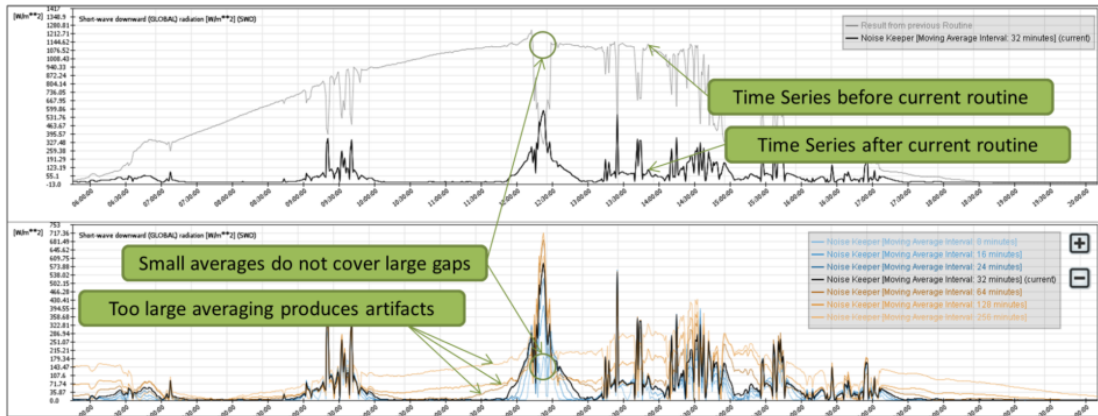


Figure 2.11: The visual interactive preprocessing approach mentioned in [6]. In the bottom part the current time series is compared to the results of the previous preprocessing steps, in the upper part it is compared to the original data. The user can adapt the moving average interval by clicking the + respectively the - Button or change it manually.

2.3.7 Visual-Interactive Preprocessing (VIP) of Time Series Data

Bernard et al. describe an approach in [6] that aims to improve preprocessing of time-oriented data. The user can select several modules, which are used in the workflow of data cleansing. Such a module would be *moving average* or *removing outliers*. However, in contrast to black box approaches, where a user can try certain operations and see what the outcome looks like, it provides a preview of the result after applying operations (see Figure 2.11). This, for example, is helpful when the user applies a moving average operation and is unsure, which interval is best used to represent the data.

2.3.8 TimeCleanser

In [22] a tool is introduced that primarily deals with problems of time-oriented data (see Figure 2.12). The main focus lies on detecting quality problems in time-oriented data through visualization, however it also provides some cleansing operations.

TimeCleanser provides operations to tackle different time-oriented data quality problems. These functionalities are [22]:

- *Correcting Syntax*: operations that allow String manipulation of dates, for example replacing ‘-’ with ‘/’ or switching month and day.
- *Smoothing Outliers and Getting Rid of Noise*: by applying a moving average to the dataset.
- *Removal of Missing Values*
- *Eliminating Duplicates*: i.e., merging identical timestamps.

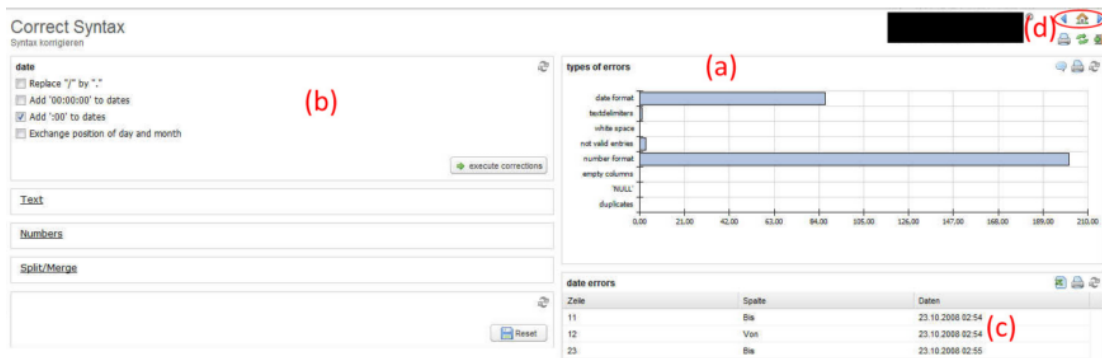


Figure 2.12: TimeCleanser’s Interface [22] for correcting syntax errors. It consists of four parts, (a) shows the error types, (b) the possibilities to correct these errors. (c) Single errors that can be corrected manually. (d) The navigation through the cleansing process is wizard-like.

All of the mentioned cleansing operations are applied to the whole dataset at once, however the tool provides means to change single entries manually as well. The underlying design principle regarding data cleansing is that this process is a sequential task with loops. This means that certain quality issues must be solved in order to discover and resolve new ones. Gschwandtner et al. defined five steps in [22], which extend Shneiderman’s Visual Information Seeking Mantra (“overview first, zoom, and filter, then details-on-demand” [55, p. 337]) and Keim’s Visual Analytics [35] Mantra:

1. *Correct Syntax First*: Erroneous syntax prevent data from being processed, visualized or analyzed. Hence, correcting the syntax is a prerequisite for any further cleansing step.
2. *Assign Semantic Roles*: For each column a semantic role should be defined. In the context of time-oriented data this, for example, means that the user has to declare, which column corresponds to ‘from’ values and which column contains the ‘to’ values.
3. *Overview*: A selected column is plotted along a time axis. Thereby, suspicious entries can already be spotted.
4. *Zoom and Filter*: As the user might be only interested in some values (e.g., sales data of last month), the tool provides methods for filtering and zooming.
5. *Data Analysis and Details-on-demand*: Furthermore, the program provides automatic as well as user-defined checks for data quality analysis. Again, a certain sequence is necessary to get meaningful results.

Firstly, time values are checked against predefined constraints, such as all intervals must be of the same length or some gaps are required (no working hours in the middle of the night).

Secondly, the validity of data values gets checked - it is important that time values are already corrected, as data values could be wrong beforehand (e.g., if the data of a shop is recorded within different intervals the sales figures are biased, i.e., between 9am and 12pm more toys were sold than between 12pm and 1pm).

Lastly, the user can resolve value sequences (e.g., correct timing of values generated by sensors) and multiple datasets (e.g., heterogeneity of scales, i.e., different time intervals in two documents).

Another design principle is that visualizations and raw data tables are not mutually exclusive. While visualizations are very helpful when it comes down to seeing patterns, outliers, and strange behavior in the observed dataset, users tend to switch back to data tables to see the raw values or for double-checking [22].

2.3.9 DataCleaner

DataCleaner [12] is (commercial) open source software, i.e., some functions are only available when purchasing commercial editions. This includes duplicate detection and merging. However, transformations are available regardless of the user's license. Disposable transformations are:

- *Conversions* (e.g., change the format from string to date)
- *String manipulations* (e.g., concatenate, removal of unwanted characters or regex search and replace)
- *Data and time operations* (e.g., convert a date into age, generate a timestamp (from date to a more grained format) or format date)
- *Matching and standardization* (e.g., country (into different ISO formats or the full name), e-mail or name standardizer)
- *Manipulations of numbers* (e.g., generating an ID, increment a number or writing a math formula)
- *Scripting*, i.e., using an expression language or JavaScript to perform transformations that are not predefined.

Figure 2.13 illustrates how a 'job' is displayed and built in DataCleaner. It requires a data set, which can be imported from different sources, i.e., databases such as, Microsoft SQL ², PostgreSQL ³, Oracle ⁴ or MongoDB ⁵, or files (e.g., Excel [40] or plain text). This set is then used to apply different analysis or transformations, however, sometimes transformations require some cleansing beforehand. For example, *Convert to date* (named

²<http://www.microsoft.com/en-us/server-cloud/products/sql-server/>

³<http://www.postgresql.org/>

⁴<https://www.oracle.com/database/index.html>

⁵<https://www.mongodb.org/>

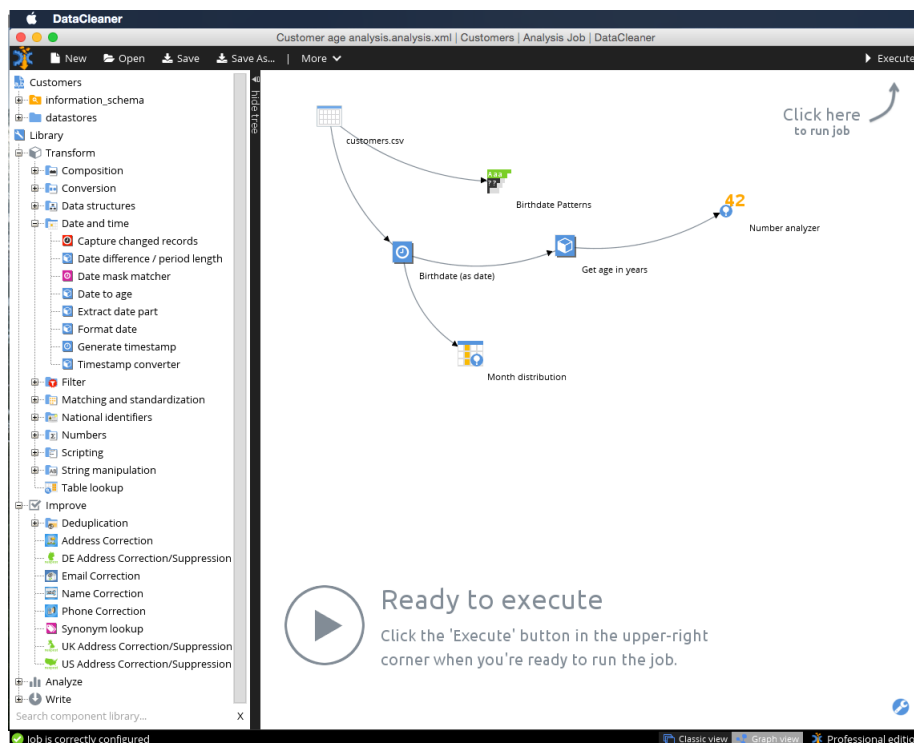


Figure 2.13: DataCleaner’s Interface [12]. In the panel on the left side the data can be chosen (data stores contains all imported tables) as well as operations (transform and improve) and tools for data analysis (analyze). They can be dragged into the panel on the right and linked to each other (visualized by arrows). Additionally, a large text field informs the user whether the job is ready for execution or if there are any components left to configure.

‘Birthdate (as date)’ has to be applied before *Date to age* (renamed to ‘Get age in years’) if the birthdate field in the source is not recognized as date. Usually, transformations are only applied on the current job. The user has to apply a *Write* operation to store changes permanently. If the source is a database it is possible to perform update operations, otherwise the user needs to save it as a new file or create a new database table.

2.3.10 Data Match 2013

Data Match 2013 [37] is another commercial data cleansing product. As DataCleaner (Section 2.3.9) it supports many different types of file formats, for instance, Excel [40], Text (.txt and .csv) and databases such as Oracle, MySQL ⁶ and Microsoft SQL. Its main cleansing operations are eliminating duplicates and syntax error checks (e.g., checking the syntax of e-mail addresses, see Figure 2.14). The tool suggests to go through the

⁶<https://www.mysql.com/>

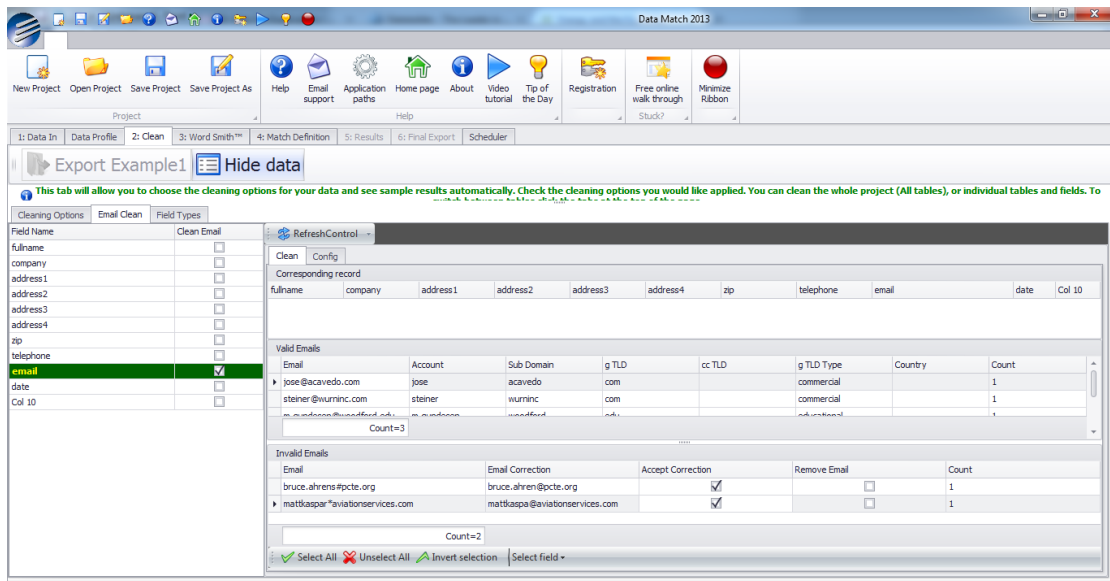


Figure 2.14: Data Match 2013’s Interface [?]. The interface supports a straight forward process, i.e., it starts with *Data In* and ends with *Final Export*. In the screenshot the automatic syntax check on e-mail addresses is performed. There are suggestions on how to correct them, but one can also decide to alter them manually.

cleaning process step by step, i.e., starting with removing different syntax errors and conducting the deduplication afterwards.

The interface consists of a table that shows the current data and checkboxes to select different options. These options can be, which field should be checked or which field is important to recognize duplicates. After a matching process the user can select the *Master* data (the entry that should exist after the merging process) and state whether an entry is a duplicate or not.

2.4 Discussion

2.4.1 Visual Interactivity

From the review of related work we observe different approaches that aim to include (visual) interactivity. Some data cleansing approaches do not aim to provide visual interactivity or support at all (e.g., [19, 39, 52, 53]), i.e., the data cleansing operations have to be defined without a graphical user interface. Their method of resolution is to use a language that enhances SQL. Thus, a specialized user benefits insofar as they can use a language they know, but with several possibilities to improve the data quality.

However, the lack of visualization makes it harder to check if an executed command succeeded, i.e., after a cleansing operation the user needs to perform a `SELECT` query to scan through the data. Potter’s Wheel [48, 49] or Open Refine [29] provide a table view

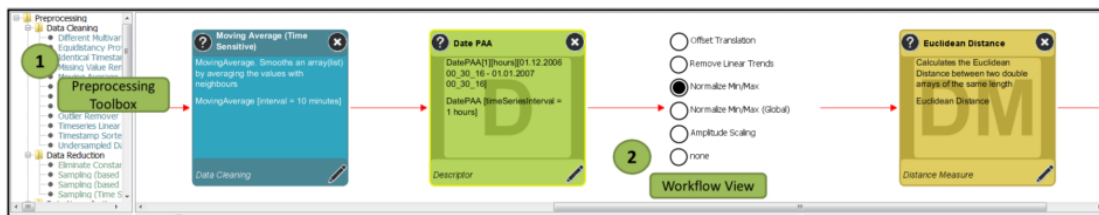


Figure 2.15: The visual interactive preprocessing approach provides a visually interactive selection of the workflow [6]. Different data cleansing operations are shown in (1) and can be dragged into (2). All operations in (2) are consecutively executed.

and refresh it each time a transformation was executed. Potter’s Wheel only updates the dataset for domains the user is currently viewing. That means the shown table is a small sample of the whole dataset and only if the user scrolls other areas are updated. Open Refine additionally provides small visualizations, i.e., distributions of numeric values. These visualizations are interactive to such a degree as one can limit the area of interest. This area is shown in the data table and thus, helps to identify possible outliers and the reason for being one. For example, the numeric attribute of a profit organizations generated the last year - however, some values are very high and others very low. A reason for this observation could be that some values are stated in billions (e.g., 8.532 billion dollars) and others are stated in thousand dollars (e.g., 17,321.45 thousand dollars) while the majority of the data is displayed as millions.

Wrangler [32] (Section 2.3.5) provides a visual preview of the table before changes are applied. Therefore, it is easy for the user to grasp the impact of the chosen transformation. Despite these previews and automatically generated suggestions, it is very hard to work with Wrangler for inexperienced users. While having some operations in mind that need to be executed, it is oftentimes hard to find the corresponding transformations. Hence, one has to be familiar with the consequences and results of each transformation, because trial and error is not a promising approach in this case. In [32] an evaluation was conducted and all participants were all professional data analysts or graduate students, who work with data on a regular basis. After a short tutorial they had to complete three tasks with both Wrangler and Excel [40]. The results were that the median performance of Wrangler was more than twice as fast as Excel. Thus, given an adequate introduction one can benefit a lot from this tool.

While table views are certainly very helpful, they are best used in combination with visualization, as Gschwandtner et al. stated in [22]. Visualization itself is a very powerful way to support users that are working with data. With respect to data cleansing it is helpful insofar as it eases the search for information and improves the recognition of patterns [8]. When it comes down to working with time-oriented data it is unthinkable to work without any visualizations. In [6] and [22] two tools focusing on time-oriented data are presented. Both approaches heavily rely on visual representations of the data.

An important concept of interactive visualizations is brushing and linking. Brushing is first described by Becker and Cleveland in [5]. The basic idea of brushing is to

highlight data that is selected in another diagram or in the table. This enables the user to understand reasons for erroneous data (as it is used in Profiler [33]), but is also a promising approach for data cleansing. For example, selecting data and then executing certain cleansing operations, such as merging or standardizing. The name linking is derived from the fact that brushing links different views on the data [2].

Yet visual interactivity can be understood in many ways. Not only working with data in diagrams is visually interactive, but also using a drag and drop menu for data cleansing operations, as it is done in [6] (see Figure 2.15).

2.4.2 Support of Time-Oriented Data

AJAX (Section 2.3.1) is not designed to solve problems we face when working with time-oriented data. However, as it is a highly flexible and customizable tool (e.g., using Java to tackle different problems) one can eliminate duplicates of the same time or interval. The missing support for these issues requires a lot of work and customization, thus it only partly supports time-oriented data problems.

Like AJAX FraQL (Section 2.3.3) is also a highly flexible language and supports many cleansing operations as well as the possibility to apply them to time-oriented data. However, these operations do not only require a lot of knowledge regarding the FraQL, but also a lot of trust in one's skills. This is because there is no preview provided and thus, for example, a merging operation could go terribly wrong. The focus on time-oriented data is only partially given as there are some aspects that are not considered, i.e., only exact duplicates are recognized. Thus, it is not possible to deal with inconsistencies.

Potter's Wheel (Section 2.3.2) is rather focused on transformations. Besides a discrepancy detection that allows to find incorrect date formats there is not much it does to support the cleansing of 'dirty' time-oriented data. The same can be said about Wrangler (Section 2.3.5), however, with the addition of Profiler (Section 2.3.5) it at least allows to detect and further eliminate duplicates. Nevertheless, the key aspect of both Wrangler and Profiler is not on time-oriented data.

Google Refine (Section 2.3.4) allows syntax checks and transformation, which are prerequisites of further cleansing operations, however, these operations do not support time-oriented data. The elimination of duplicates with Google Refine geared to merge entries with typos or different spellings (e.g., Microsoft Corp., Microsoft Corporation and MS Corporation), thus, only one column is regarded, which does not work with time-oriented data (two entries with the same timestamp are not always duplicates - it depends on additional values).

While not providing too many cleansing operations, ERACER (Section 2.3.6) offers support for time-oriented data problems. TimeCleanser (Section 2.3.8) and VIP (Section 2.3.7) are the only two tools with special focus on time-oriented data.

DataCleaner (Section 2.3.9) provides conversions as well as examinations of dates (e.g., distribution of birth dates for each month). Compared to Google Refine it is possible to eliminate duplicates by considering further values. Thus, it is rather suited to cleanse 'dirty' time-oriented data, but overall the aim to cleanse 'regular' data, for instance, customer data.

<i>Tool</i>	<i>Conflict Resolution</i>	<i>Missing Values</i>	<i>Eliminating Duplicates</i>	<i>Smoothing Outliers</i>	<i>User Definable Operations</i>	<i>Visual Interactivity</i>	<i>Time-Oriented Data</i>
AJAX	—	○	●	—	●	—	○
Potter's Wheel	●	○	●	—	—	○	—
FraQL	●	●	●	○	●	—	○
Google Refine	●	○	●	●	●	○	—
Profiler Wrangler	●	●	●	○	○	●	—
ERACER	—	●	—	○	—	—	●
VIP	—	●	●	●	—	○	●
TimeCleanser	○	●	●	●	—	○	●
DataCleaner	●	—	●	—	●	●	—
Data Match	●	—	●	—	—	●	—

Table 2.1: Comparison of different tools.

● means something is fully supported,

○ refers to a partial support or the possibility to define an operation yourself and

— stands for not supported.

Data Match 2013 (Section 2.3.10) is not applicable on time-oriented data.

2.4.3 Comparison

As the literature review shows there are a lot of different approaches for the task of cleansing data. However, they differ in purpose and the methods they support. The following table will provide a short overview of the different cleaning tools presented in Section 2.3 with respect to the methods they support, visual and interactive features and special regards to time-oriented data.

3.1 Requirements

In order to start the implementation we had to find out what is expected of the prototype. We based our requirements on a taxonomy of time-oriented data quality problems by Gschwandtner et al. [23]. The next step was to classify and categorize each possible data quality problem that might occur when working with time-oriented data as well as finding a possible solution.

Within the scope of this work, we focused on cleansing time-oriented data quality problems of data from a single source only. However, some of our cleansing operations allow coping with quality issues stemming from multiple sources as well. After analyzing the taxonomy we figured out that at least six distinct operations were necessary to deal with the mentioned problems¹.

These categories of cleansing operations are the following:

- **Change values:** modifying values, for example, through calculation.
- **Transform:** changing the column's content by changing its representation.
- **Edit intervals:** editing intervals, for instance, standardizing their length to a certain time span.
- **Correct implausible values:** correcting outliers (e.g., applying a method mentioned in Section 2.2.4).
- **Impute missing values:** filling missing fields (e.g., using an approach stated in Section 2.2.2).
- **Deduplication:** detecting and removing duplicates (see Section 2.2.3).

¹Except ambiguous and outdated data. For those no solutions are presented in this work. For further information see Section 6.1.

- **Remove:** removing rows that are incorrect.

Not all of the methods are not only suited to correct time-oriented data problems, but they can also be applied to regular data without any time fields. However, all of them were extended to work properly for fields that contain time or date values, whereas some methods will be disabled if the data set does not contain any time or date columns. The next sections will introduce the theory behind cleansing operations in detail and explain the time-oriented data problems that can be solved by applying them.

3.2 Design of Cleansing Operations

The following sections describe which methods are necessary to deal with most of the identified problems by Gschwandtner et al. [23].

3.2.1 Change Values

These methods can be used to modify certain data, i.e., all data within one column, by applying semi-automatical changes. Some operations are rather static and do not require the user to specify many parameters (e.g., change time), while others are highly flexible and completely user-definable (e.g., calculations). Additionally, the user can set restrictions, i.e., only the data within a certain month and a certain year should be changed.

3.2.1.1 Calculations

The *Calculations* function allows correctifying wrongly recorded data, which have a reference to one or more other values. In terms of time-oriented data this could be a rather simple calculation, for example, the field `Working Hours` is defined as difference between `Work Begin` and `Work End`. In order to make sure that this is correct for all entries the calculation `Work End - Work Begin` has to be executed. Moreover, calculations allow the usage of constants to correct time shifts, for instance, if the clock that recorded the data was one hour ahead (e.g., did not switch to wintertime accordingly). Thus, a simple solution would be to subtract one hour of all in wintertime. Thus, it is possible to apply calculations only to rows that fulfill a certain condition (e.g., `month > 10 || month < 6`) and subtract one hour of them.

As Table 3.1 shows, calculations can be an effective way to deal with time-oriented data problems. Furthermore, they allow to compute other parameters such as profit (e.g., `volume of sales * 0.82`) or turnover per employee (e.g., `volume of sales / employees at store`). These calculations will not be essentially important in order to correctify ‘dirty’ data, but allow the user to use the prototype throughout the whole process of data processing without having to switch tools in order to execute any equations.

²Regular value added tax (VAT) rate in Austria.

3.2.1.2 Change Time

This is a more accurate method to deal with dates and an extension of *Calculations*. Since dates have many components, i.e., day, month, year, hour, and minute at least, it is desirable to alter only one of them. Furthermore, dates are somewhat difficult to modify by entering an equation (e.g., `date - 10-1-1` to subtract ten years), hence, it should be possible to specifically change dates by picking a parameter to change and enter the desired value (e.g., `year - 10`). Also, calculations have some disadvantages regarding time-oriented data. For example, to adjust a time to the correct time zone someone might simply use a calculation (e.g., `time + 07:00`). This might work in many cases, however, it will cause problems if the original time is at or after 19:00. For example, if the time is 19:30 the calculation will display 24:30 instead of 0:30 and a shift in date. Therefore, changing time takes care of such issues. It will not only increment the time accordingly (starts with 0:00 if 23:59 gets increased by one minute), but also increment the date, i.e., adding one minute to 2015-01-01 23:59 results in 2015-01-02 00:00.

3.2.2 Transform

Transformation is a rather imprecise term as it could mean anything. In this case, we are referring to any changes within the data structure of the table. These changes support the user in displaying the data as needed (e.g., changing the date format or split the date (DD-MM-YYYY HH:mm) into a date (DD-MM-YYYY) and a time field (HH:mm)), adding new columns, or removing existing columns. A rather special and time-oriented case is transforming a time column to an interval column. This transformation is only possible if there is a reference column, for example, the data contains two time or date columns (`start` and `end`) and `end` should be shown as interval. If the time in `start` was 08:30 and the time in `end` 09:45, `end` will be displayed as 75'.

Table 3.2 shows common issues that can be dealt with by using *Transformations*. However, there are some cases which the prototype cannot handle (see Section 6.1), because certain formats will not be recognized as time fields (e.g., 7h42' or 03-22) and thus, cannot be treated accordingly.

3.2.3 Edit interval

Editing Intervals is a kind of transformation, because some operations alter the table structure with respect to some changes in the interval structure (e.g., aggregating intervals which results in merging some rows of the table). Intervals require two columns, which contain dates. It does not matter whether these columns are date, time, or interval columns.

3.2.3.1 Standardization

For instance, one feature is the *Standardization* of interval lengths within the whole table. Let us consider a case in which some entries cover a time span of 30 minutes, while others

Category	Title	Description (<i>Example</i>)	Possible Solution
Implausible values	Implausible Range	Very early date / time in the future (<i>Date: 1899-03-22</i>); (<i>date: 2099-03-22</i>); (<i>date: 1999-03-22, duration: 100y</i>)	year - 100 where year == 2099
Wrong data	Wrong data type	No time / interval <i>Date: AAA; duration: *</i>	Calculate duration or Previous time + x hours
	Coded wrongly or not conform to real entity	Wrong time zone <i>UTC data in stead of local time</i>	Change time: +/- x hours
		Valid time/interval, but not conform to the real entity (<i>Admission: 2012-03-04</i>) vs. (<i>real admission: 2012-03-05</i>)	date + 1 day
	Domain violation (outside domain range)	Minimum / maximum violation for given time/interval/type of day <i>Sales at night even though no employees were present</i>	Set Sales to 0 where time > 20:00 && time < 08:00
		Sum of sub-intervals impossible <i>Seeing the doctor + working hours longer than regular working hours</i>	Set working hours to working hours - hours at doctor's office
		Start, end, or duration do not form a valid interval (<i>End ≤ start</i>); (<i>duration ≤ 0</i>)	Set end time to start time + duration
Incorrect derived values	Error in computing duration <i>Computing the number of work hours per day without deducting the breaks</i> <i>No proper dealing with summer time changes</i>	Set work hours to works hours - breaks Add/subtract one hour within the period where the time change was not properly dealt with.	

Table 3.1: Problems defined by Gschwandtner et al. [23] that can be solved with *Calculations*.

Category	Title	Description (<i>Example</i>)	Possible Solution
Wrong data	Wrong data format	Wrong date/time/datetime/duration format (<i>Date: YYYY-MM-DD</i>) vs. (<i>date: YY-MM-DD</i>); (<i>duration: 7.7h</i>) vs. (<i>duration: 7h42'</i>)	Change format accordingly
	Misfielded values	Time in datefield, date in time/duration field (<i>Time in datefield: 14-03</i> , <i>date in timefield: 12:03:08</i>)	Switch fields (and change format)
	Embedded values	Date+time in date field, timezone in time field/duration field (<i>Time: 22:30</i>) vs. (<i>time: 22:30 CET</i>)	Delete 'CET' from the cell
Heterog. syntaxes	Different table structure	Time separated from date vs. date+time or start+duration in one column (<i>Table A: start-date, start-time</i>) vs. (<i>table B: start-timestamp</i>)	Split/merge date and time fields, so both tables have the same structure

Table 3.2: Problems defined by Gschwandtner et al. [23] that can be solved with *Transformations*.

cover an hour and again other entries cover two or more hours. By standardizing these entries they all can be set to a user-specified time span (whether it's on a fine level, e.g., 10 minutes, or a rather coarse one, e.g., four hours).

3.2.3.2 Aggregation

Furthermore, intervals cannot only be aggregated within a day, but also on a coarser level (e.g., a number of days or even months). This might be of interest to the user if he/she is interested in seeing the sales of a month rather than the sales of each hour. In order to make this possible the user can define which columns are time-relevant, i.e., which columns must be adjusted to the new time interval. For example, workers at a shop might not be relevant when changing the interval length (the same two salesmen are in the office all day), however, the number of sales will obviously depend on the length of the interval. Thus, the prototype needs to provide means for the user to interactively select the columns that should be summed up with respect to the aggregated time interval.

3.2.3.3 Restrict Intervals

Another operation would be to restrict intervals to a certain time range, i.e., only allow intervals from 08:00 to 20:00 (e.g., because the shop is closed the other hours). Intervals outside the specified range can be deleted or marked as erroneous. Additionally, it is possible to define a minimum off-time (e.g., between shifts). Again, the prototype

Category	Title	Description (<i>Example</i>)	Possible Solution
Implausible values	Unexpected low/high values	Too long/short intervals between start-start/end-end <i>Below one second at the cash desk</i>	Standardize the interval duration
		Too long/short intervals between start-end/end-start <i>Off-time between two shifts less than 8 hours</i>	Define a minimum off-time duration
		Too long/short overall timespan (first to last entry) <i>Continuous working for more than 12 hours</i>	Limit the maximum interval length
Wrong data	Wrong data format	Times outside raster (e.g., for denoting end of day) <i>1-hour-raster but time is 23:59:00 for the end of the last interval</i>	Standardize interval length to 1-hour
	Domain violation (outside domain range)	Uneven or overlapping intervals <i>Turnover data for 8:00-9:00, 9:00-11:00, 11:00-12:00</i>	Standardize interval length
Heterog. semantics	Heterogeneity of scales (measure units / aggregation)	Different granularities; different interval length <i>(Table A: whole hours only) vs. (table B: minutes)</i>	Standardize the interval length to a common format

Table 3.3: Problems defined by Gschwandtner et al. [23] that can be solved with *Edit Interval* operations.

needs to provide simple means for the user to delete, annotate, or shift the hours so the minimum off-times are not violated anymore.

3.2.4 Correct Implausible (Values)

Implausible values are similar to outliers (see Section 2.2.4). This section introduces two ways on how to deal with this type of data quality problem.

3.2.4.1 Outlier Correction by Applying Moving Average

The prototype offers smoothing methods such as *exponentially weighted moving average*, which have been discussed in the referenced section. However, it further allows to apply a *weighted moving average* with focus on time series. That means, that the user can select whether he/she wants to apply a moving average on the last n entries or the last n entries that share the same time/weekday/day within a month/day within a year. Thus, it is possible to correctly deal with outliers where a common pattern can be identified (e.g., increased number of sales before Christmas and Easter compared to other times of the year), instead of just using the last couple of entries to calculate a plausible value. Exponential smoothing in general requires some knowledge, so the user can decide which

Category	Title	Description (<i>Example</i>)	Possible Solution
Implausible values	Unexpected low/high values	Deviations from daily/weekly ... profile or implausible values <i>(Average sales on Monday: 50) vs. (this Monday: 500)</i>	Moving average of last n Mondays
		Changes of subsequent values implausible <i>(Last month: 4000 income) vs. (this month: 80000 income)</i>	Exponential smoothing
		Same value for too many succeeding records <i>17 customers in every interval of the day</i>	Exponential smoothing
Wrong data	Misfielded values	Values attached to the wrong/adjacent time/interval <i>GPS data shows sprints followed by slow runs although the velocity was constant</i>	Denote outliers as missing values and replace them with the mean value
	Domain violation (outside domain range)	Outliers in % of concurrent values (attention with small values) for a given point in time/interval <i>On average (median) 30 customers in a shop in a given hour - in a 10'interval within that hour, a value of 200 is present</i>	Denote outliers as missing values and replace them with the median value of that day

Table 3.4: Problems defined by Gschwandtner et al. [23] that can be solved with *Correct Implausible*.

operation, i.e., single, double, or triple, fits his/her requirements best. An alternative method is using this implementation of a weighted moving average which is a simpler concept to comprehend and can be just as powerful as exponential smoothing (depending on the data structure).

3.2.4.2 Outlier Correction by Imputation

Another method is to define all outliers as *missing values* and impute an average. This average can be either the mean, median, or computation of the *k-nearest neighbors*. Mean and median may not be as precise as moving average methods, yet they allow correctifying outliers where no pattern can be identified. KNN is something in between as it searches for the most similar entries and these can be defined over time (even though this is not always the case as it depends on the number of different fields), but it does not necessarily take the last n entries into consideration but those, who have a good general fit. For example, instead of using the sales figures of the 23rd December of 2008 to 2013 ($n = 5$) to correct the outlier that was found on 23rd December 2014 it may take the 23rd December of 2004, 2005 and 2009 as well as the 21st December of 2011 and the

22nd December of 2014.

3.2.5 Impute Missing Values

Category	Title	Description (<i>Example</i>)	Possible Solution
Missing data	Missing value	Missing time/interval and/or missing value <i>(Date: NULL, items-sold: 20)</i>	Impute missing intervals (if there is a missing gap within dates, e.g., 20-01-2015 is missing, NULL will be replaced with that date)
		Dummy entry <i>Date: 1970-01-01; (duration: -999)</i>	Impute the mean or find the real date of the entry
	Missing tuple	Missing time/interval + values <i>The whole tuple is missing</i>	Impute missing intervals as well as missing values for each field that is neither interval start/end nor date
Wrong data	Wrong data type	No time/interval <i>Date: AAA; duration: *</i>	Impute missing values (e.g., duration mean)

Table 3.5: Problems defined by Gschwandtner et al. [23] that can be solved with *Imputation of Missing Values*.

As already mentioned in the previous section, the methods of imputation are mean, median, and k-nearest neighbors. Additionally, it should provide the user with some tailored operations regarding time-oriented data. The prototype needs to provide means for imputing missing intervals, for example, if there is an interval gap between 12:00-14:00, or imputing missing times, i.e., the interval end of an entry is missing. However, we know that an average interval is two hours long, so with regard to the start time we can basically impute `start time + 2h`.

All imputation means allow to replace both time/interval fields as well as numerical (e.g., sales or profit) and non-numerical (e.g., name or gender) values.

3.2.6 Deduplication

The definition of duplicates in the context of time-oriented data can differ from its usual definition. Each entry that occurs at the same time may be a duplicate with respect to different key entities (e.g., two different workers can work at the same time but two entries of the same worker at the same time would be a duplicate). The goal of *Deduplication* is both to remove overlapping or identical intervals, but also to find and eliminate highly similar entries. For example, if two entries are almost the same, but the date is different, there is a high chance these two are duplicates and there is a typo in the date column. Thus, it is not enough to simply check if two entries share the same time or not. The prototype needs to provide means to identify possible duplicates. After identifying them

Category	Title	Description (<i>Example</i>)	Possible Solution
Duplicates	Unique value violation	Same time/interval (exact same time/interval though time/interval is defined as unique value) (<i>Holidays: 2012-04-09; 2012-04-09</i>)	Detect duplicates of the same time and remove one of them
	Exact duplicates	Same time/interval and same values (<i>Date: 2012-03-29, items-sold: 20</i>) <i>is in table twice</i>	
	Inconsistent duplicates	Same real entity with different time/interval or values (<i>Patient: A, admission: 2012-03-29 8:00</i>) vs. (<i>patient: A, admission: 2012-03-29 8:30</i>)	Detect duplicates that are very similar with focus on patient name and admission date and remove one of them, if these are duplicates
		Same real entity of time/interval (values) with different granularities (rounding) (<i>Time: 11:00 vs. 11:03</i>); (<i>Weight: 34,67 vs 35</i>)	Search for similar entries and remove the unwanted duplicate
Wrong data	Domain violation (outside domain range)	Circularity in self-relationship $Interval A \subset interval B, interval B \subset interval A, A \neq B$	Detect them as duplicates and remove one

Table 3.6: Problems defined by Gschwandtner et al. [23] that can be solved with *Deduplication*.

the user must be able to select which entries are actual duplicates and how they should be dealt with.

3.2.7 Remove

The last cleansing operation introduced in this chapter is also the operation that should only be applied as a last consequence, i.e., if no other cleansing operation provides the necessary means to cope with a time-oriented data quality problem.

3.2.7.1 Removal as a Last Resort

While removing is certainly not a well-established technique for dealing with problems of data quality it might be the only choice to solve some issues, especially if no further context is available. For example, in a quarterly record of sales figures without any gap in time, a record with a date 2099-01-01 is found. As there are no gaps or missing dates in the remaining data table, this entry is not just a valid entry with a wrong date. Thus, the choice is either to leave the entry the way it is and ignore it or simply removing it. Removing whole entries should be the last choice and only executed if they are completely meaningless (because there is no way to interpret them in the right way).

Category	Title	Description (<i>Example</i>)	Possible Solution
Missing data	Missing values	Dummy entry (<i>Date: 1970-01-01</i>); (<i>duration: -999</i>)	Remove the entry if the missing values made it meaningless
Implausible values	Implausible range	Very early date / time in the future (<i>Date: 1899-03-22</i>); (<i>date: 2099-03-22</i>); (<i>date: 1999-03-22, duration: 100y</i>)	Remove the entry
Wrong data	Domain violation (outside domain range)	Minimum / maximum violation for given time/interval/type of day <i>Sales at night even though no employees were present</i>	
		Start, end, or duration do not form a valid interval (<i>End \leq start</i>); (<i>duration \leq 0</i>)	
		Circularity in self-relationship <i>Interval A \subset interval B, interval B \subset interval A, A \neq B</i>	Delete either A or B

Table 3.7: Problems defined by Gschwandtner et al. [23] that can be solved by *Removing Entries*, if the other means are not applicable.

3.2.7.2 Removal & Deduplication

Additionally, removing is also needed to deal with duplicates and thus, the function has its right to exist. Furthermore, the operation gives users the possibility to delete entries, which are annotated (e.g., missing values or outliers). This way the user does not have to select each entry manually if he/she only wants to target dirty entries.

3.2.8 Non-Cleansing Features

Besides the mentioned cleansing methods the prototype should also provide the user with two further features, namely a history and the execution of commands via a command line. Additionally, the prototype should provide the user with a help function as some operations require many parameters and are hard to comprehend (e.g., triple exponential smoothing requires parameters called α , β , γ and l).

3.2.8.1 History

The *History* should allow the user to undo his/her last n operations and also to redo them. However, a redo is only possible if the user has not executed any further operations after the undo operation. That is because some operations would have a different result if they are applied on a different data set (e.g., the *Imputation of a Mean* is different if all data is taken into consideration or if 50% of all entries are deleted).

3.2.8.2 Command Line

This feature is intended for the use of user definable cleansing operations (see Section 2.2.5). The *Command Line* supports a faster input of operations that do not require many parameters. *Calculations* can be seen as user definable as there are no limitations (except not all operators are supported, depending on the type of data). The idea is to support the user in doing routine tasks in a faster way (typing the command is usually faster than opening dialogs and selecting parameters).

QualityTime

4.1 Extension of TimeProfiler

The cleansing operations are an extension to an existing prototype, namely *TimeProfiler*, written by Theresia Gschwandtner and Benjamin Klaus, at the *Vienna University of Technology*. The code of the prototype is written in Java and TimeProfiler reads input from .csv (comma separated values) files. The separators between fields depend on the used software and user settings, and thus, TimeProfiler can process files with different separators: comma, semicolon, tabulator, or space.

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@b.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.at
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	Johar				PREPARE	a@b.at
19	Mar				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	330	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: The data is shown in a fisheye table, i.e., the hovered row and neighbor rows are bigger than the rest. In this screenshot different checks were performed and detected data quality issues are highlighted by background color. A short details-on-demand description of the hovered problem is also provided.

The functionality of TimeProfiler allows the representation of data in the form of a fisheye table (see Figure 4.1) as well as the recognition of erroneous data. Partly the dirty data is identified during the process of reading in and converting the file (e.g., missing values); other types of faulty data can be identified by running checks (e.g., finding outliers regarding interval length). Running checks is helpful and sometimes mandatory before applying cleansing operations as some of these cleansing operations only focus on annotated entries (e.g., correcting values which have been previously identified as outliers). It also helps to get a better overview of the data set and to guide the user to what corrections have to be applied (e.g., setting the interval length to a minimum of one hour).

The visualization, i.e., the storing of the entries are supported by using a slightly altered version¹ of the *prefuse visualization toolkit*².

4.2 Architecture

The general architecture of the prototype is a model-view-controller design pattern [11], which is kept during the addition of cleansing operations.

- **Model:** The raw data table, namely the `DirtyTable` class, that contains the columns as well as tuples, i.e., each entry/row is stored as a tuple.
- **Controller:** The checks and cleansing operations represent the controller. The controller performs as intermediary between view and model. Thus, the view has no direct access to alter data in the model via operations (however, single cells can be edited in the table).
- **View:** The view consists of the fisheye table and dialogs that are used to enter parameters for different checks and operations. The user input is forwarded to the controller and afterwards the fisheye table gets updated accordingly (e.g., visualizes annotations or the changed data (structure)).

Nevertheless, it is necessary to make some further architectural changes in order to implement a history, i.e., undo and redo operations (see Section 4.3.2.11). Most promising option for the realization is the command pattern [11], which allows objects to call apply or revert of other objects. The structure of this pattern is also used in Open Refine [30] and due to its efficiency and simplicity it is the basis of our implementation.

The most important classes of the structural additions (see Figure 4.2) are `History`, `HistoryEntry`, and `Change`.

- **Change:** each instance of `Change` implements the actual cleansing operation that is executed on the model. For example, the cleansing operation to add a column can be found under `ColumnAddChange` and the operation to impute the mean is located at `ImputeMeanChange`.

¹<https://github.com/ieg-vienna/ieg-prefuse>

²<http://prefuse.org>

- **HistoryEntry**: each Change is stored in a HistoryEntry, which accesses the necessary methods to *apply* or *revert* (undo) an operation. *Redo* is simply to be understood as a repetition of the *apply* method.
- **History**: each HistoryEntry is stored in the History either as *past* or *future* entry, i.e., past entries can be undone and future entries can be redone.

In order to simplify the creation of HistoryEntry and Change objects a further type of class is introduced. AbstractCleansingOperation is an abstract class, which extensions are accessed by the view, for instance, AddColumnTransformation. The functionality of this class is to check whether all parameters are correct, i.e., correct naming of the new column (e.g., an existing name cannot be used twice), as well as transforming user input if necessary. An example for such a transformation would be the input of a calculation, i.e., checking the semantics and checking if the used column names exist or if the constants fit the type of the target column (e.g., while 03:00 would work for time columns it would not work for any number columns).

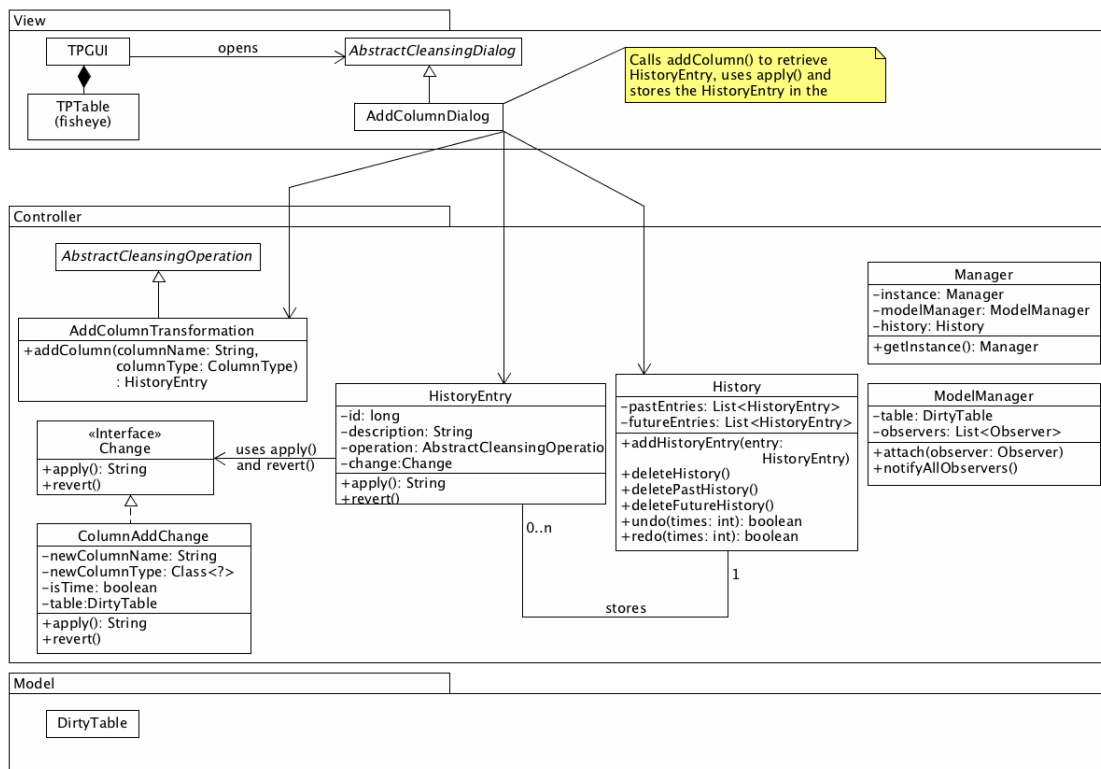


Figure 4.2: Structure of QualityTime visualized as UML diagram using the example of adding a new column to the table.

Lastly, the singleton design pattern [11] is used in order to ensure all cleansing operations access the same table and that only one History exists. We called this class Manager,

which grants access to the *History* and to the *ModelManager*. The *ModelManager* class enables access on the model and notifies the view in case of changes (if so, the view will be updated accordingly).

4.3 Design and Implementation of Cleansing Operations

4.3.1 QualityTime User Interface

TimeProfiler’s user interface was extended by adding four tabs in the menu (*Table/File*, *Cleansing Operations*, *History*, and *Help*) and a command line (Figure 4.3). A status bar informs the user about what happened after the execution of a cleansing operation, i.e., a message that contains the number of rows affected, or a general update (e.g., ‘New column *name* of type *type* has been added’), or an error message in case the entered parameters were not correct. In Figure 4.3 two tabs of the menubar were highlighted, because they are essential for cleansing.

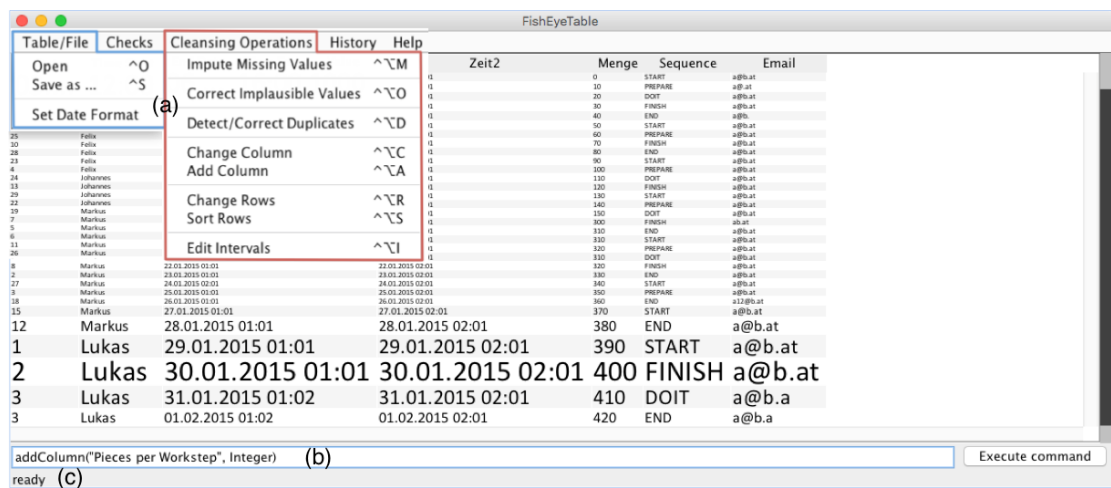


Figure 4.3: QualityTime’s UI consists of 4 parts. (a) shows the menu bar. Highlighted are *Table/File* and *Cleansing Operations*. The *Table/File* tab contains the transformation to change the date format, whereas the *Cleansing Operations* tab holds all other operations. (b) visualizes the command line and (c) a status bar that shows feedback for each executed check or operation. The fourth part is the fisheye table already introduced in TimeProfiler.

4.3.1.1 The Menu Bar

The menu bar is the key element of QualityTime as it provides access to all available features, whereas the command line (see Section 4.3.1.2) only covers operations that do not need a lot of parameters. It includes the following menus:

1. ‘Table/File’:

- ‘Open’ (see Section 4.3.2.2)
 - ‘Save as ...’ (see Section 4.3.2.2)
 - ‘Set Date Format’ (see Section 4.3.2.3)
2. ‘Checks’: In this work we do not focus on the available checks as they were implemented in a previous work.
 3. ‘Cleansing Operations’:
 - ‘Impute Missing Values’ (see Section 4.3.2.4)
 - ‘Correct Implausible Values’ (see Section 4.3.2.5)
 - ‘Detect/Correct Duplicates’ (see Section 4.3.2.6)
 - ‘Change Column’ (see Section 4.3.2.7)
 - ‘Add Column’ (see Section 4.3.2.8)
 - ‘Change Rows’ (see Section 4.3.2.9)
 - ‘Sort Rows’: sort entries with respect to different criteria
 - ‘Edit Intervals’ (see Section 4.3.2.10)
 4. ‘History’:
 - ‘Show History’ (see Section 4.3.2.11)
 5. ‘Help’
 - ‘Show Help’ (see Section 4.3.2.12)

4.3.1.2 The Command Line

Purpose of the command line is to grant easy access to basic features. Basic features are those which do not require many parameters. For example, adding a new column only requires two parameters (see Figure 4.3), the name of the column to insert and its type. On the contrary, imputing missing values with the method of k-nearest neighbors requires at least three parameters, however, up to seven parameters can be specified for an advanced usage. The difficulty thereby would be on how to process the input if the user wishes to set parameters 1, 2, 3, and 6 (without needing parameters 4, 5, and 7).

The commands can be entered in two possible ways:

- `<columnToChange>.<operationName>(<parameters>)`
- `<operationName>(<columnToChange>, <parameters>)`.

The following eleven operations can be executed by using the command line (see Section 4.3.2):

- **Add column**

- **Remove column**
- **Rename column**
- **Split column**
- **Merge column**
- **Delete column part**
- **Set value/calculate**
- **Switch columns**
- **Remove rows**
- **Change interval representation**
- **Change date format**

These are mostly transformations, because transformations work on a highly generalized level, e.g., the table structure does not matter on whether you can add or remove a column. Additionally, most of them only need the specification of a few column names in order to perform a cleansing operation.

4.3.1.3 Dialogs

If the user wants to apply advanced operations or is not interested in using the command line the input is entered into dialogs (Figure 4.4). For each menu entry (in Section 4.3.1.1) there is a dialog that supports using the correct input for each cleansing operation.

4.3.2 Features

In this section the implemented features are explained in detail. Furthermore, several examples are provided to better grasp the concept of each feature.

4.3.2.1 The Key Column Field

Almost every cleansing operation has an optional parameter, called *key column*. This field is needed if there are several different entries within the same data set. For example, a mother company (e.g., *REWE*³) keeps track of all sales and other activities of their daughter companies (for *REWE* that would be *Merkur*, *Billa*, *Bipa*, *ADEG*, and *Pennymarkt*). Obviously, sales figures and corresponding profits are deviating, because of different strategies (e.g., *Pennymarkt* is positioned in the lower price segment whereas *Merkur* is concentrated on the higher price segment), shop size, or different goods (*Bipa* sells mainly cosmetic products but the other shops are focused on selling food). For

³https://www.rewe-group.at/Startseiten/Startseite/rg_Homepage.aspx

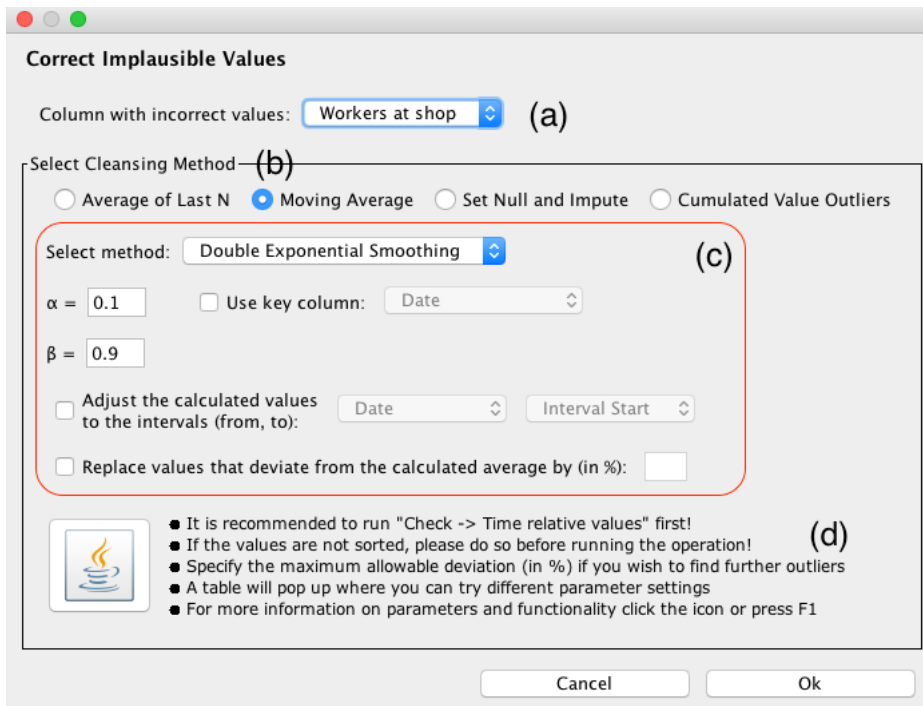


Figure 4.4: The *Correct Implausible Values* dialog. It consists of four parts: (a) mandatory parameters for all methods to be found in the dialog, (b) the different methods available for this cleansing operation, (c) operation dependent parameters (some of them mandatory, others optional - which is described in (d) a short help overview).

instance, a database contains sales values of different stores. If there are some missing values for one store, it does not make sense to compute a mean sales value for this store from all available sales values (including all stores and missing values). Thus, we only use non-missing sales values for this specific store to compute the mean sales value for this store. Additionally, the user can select which entity of the *key column* he/she wants to update, i.e., he/she can update all supermarkets or just *Merkur* for example.

4.3.2.2 Open / Save as ...

Open is a very basic feature that allows the user to load a new .csv file into the table. The changes on the current table will be lost if the user does not save his/her previous work. As mentioned earlier there are several types of .csv files, because possible delimiters are ‘;’ (German), ‘,’ (English) and ‘**tab stop**’. Choosing the wrong format (e.g., ‘,’ instead of ‘**tab stop**’) will cause the table to be displayed incorrectly, i.e., one column contains all the data that should be displayed in various columns.

Save as ... is simply used to write a new .csv file and the user can select the delimiter that should be used.

4.3.2.3 Set Date Format

This feature allows the user to change the date format of one or of all date columns. The format specification is similar to what Java uses⁴, for instance, displaying a date in the American date format (mm/dd/yyyy) would require the input of MM/dd/yyyy. These changes only affect the visual appearance, but do not change the data in particular. When loading a new data set the format is automatically converted to a European format, i.e. dd.mm.yyyy, thus changing the position of month and day will result in problems when reloading the data.

4.3.2.4 Impute Missing Values

The *Imputation of Missing Values* is only available to interval, numerical, and alphabetical (including Boolean) columns. We do not support the imputation of date values as it would not be helpful to replace a dummy or missing date value with the average. Most likely, it would lead to duplicates because of two or more entries at the same time. However, we introduce a method that can help to find the right date for an erroneous entry (see *Impute Missing Intervals*).

Impute Mean/Median

These two features are almost identical as they only differ in the statistical measure that is used. We can distinguish between imputing missing values into time columns, i.e., columns that solely contain times, but not dates, and numerical or alphabetical columns.

For the *Imputation of Missing Values* in time columns it is possible to define a so-called reference column, which is used to determine the average distance between two time columns. For example, we have three entries with two columns (*start* and *end*) that contain the following value pairs: (08:00, 16:00), (09:00, 17:00), (12:00, -). If we simply use the mean/median value for imputation the result would be 16:30, however the average duration is *eight hours*, thus, making use of a reference column would result in imputing 20:00.

In case of imputing missing values into a numerical column there is the possibility to impute values considering the length of the respective time interval. For example, we have a table with three columns, two of them represent the interval on a certain day (like in the previous example) the third one is a time-related field, sales during that interval. To simplify the table representation I will use the difference between the two interval columns and only write the interval length. These value pairs are in our table: (2 hours, 24), (1.5 hours, 18), (2.5 hours, -). If we imputed the mean without considering the interval length of each entry, we would get an average of 21.5 (or 22 if only whole numbers are accepted) sales, otherwise the figure would be 30.

Imputing values into alphabetical fields is done by searching for the most frequent string and inserting it.

⁴<http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html>

The mean/median is always calculated once for the whole data set and afterwards imputed in each field that is annotated as missing value. In case of data from different sources within the same table (e.g., different stores), the imputation can be adapted to a selected column, i.e., only the matching entries for each missing value (same store name) are taken into consideration.

Impute KNN [54]

K-nearest neighbors also supports the usage of reference columns for time columns and for interval columns when it comes to imputing numerical values, like mean and median do.

Further options are distance calculation (Euclidean or Manhattan), whether Levenshtein distance [9] should be applied to nominal values (i.e., non-numerical fields), if the distance should be standardized, and whether special attention should be paid to date fields. The latter means that not only the time difference (in hours) is taken into account, but also whether the dates are on the same day, month, year, and so forth. That way two entries on the same day, but in a different year (e.g., *2015-08-20* and *2014-08-20*) are very close, whereas under the aspect of a normal distance calculation they would be very far apart.

In order to get the value for imputation the mean of the k -nearest neighbors is determined.

Impute Missing Intervals

This feature covers the *Imputation of Missing Intervals*, i.e., gaps in time that should not exist. The user can specify which interval spans should be covered on each day of the week (on weekdays, weekends, or the whole week) and the tool will close any gaps during that time.

Figure 4.5 visualizes how the parameterization of this method looks like. Besides the regular specification (interval start and end, and date in case interval start is only a time field), it is possible to set intervals for each day, instantly impute missing values that occur when a new entry is created (a new entry will only consist of a date (if interval start is not a date already), interval start and end) and set a fixed length for newly inserted intervals. The fixed length is working insofar as the desired interval length fits the gap. For example, if the gap is two hours long, the interval length of 60 minutes can be inserted twice. If the gap is only 1.5 hours, the first interval will have a length of 60 minutes whereas the second interval is only 30 minutes long.

As we have mentioned at the beginning of this section, it is not directly possible to impute missing date values, because if there is an entry that consists only of a start and end point in time, but the date is NULL or a dummy value, it will be held back until a certain gap fits exactly the start and end time of that entry. If the entry does not fit any gap the date will not be changed from its faulty value. Additionally, it is possible to avoid overlapping intervals, i.e., the imputed interval ends before the next one starts (e.g., *end: 13:59, start:14:00*).

4.3.2.5 Correct Implausible Values

Implausible values are time-relative values that do not follow a given trend or are certainly recorded incorrectly. Thus, only numeric fields can be affected in this category.

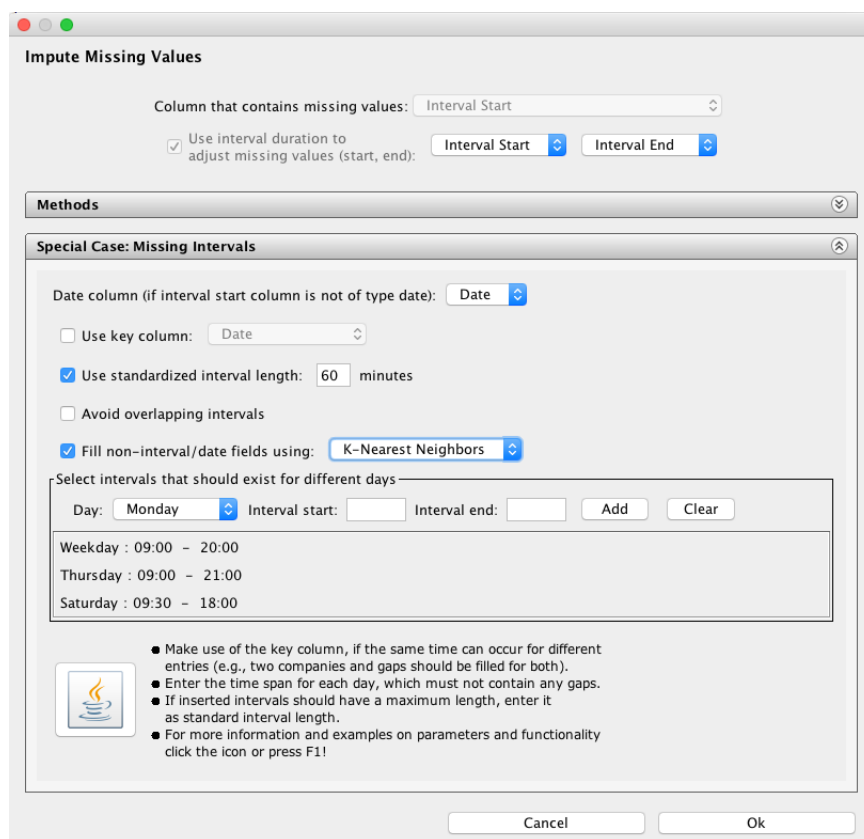


Figure 4.5: Dialog to impute missing values. In this example, we are trying to impute missing intervals into a shop's data. The shop's usual opening hours during the week are from 9:00 to 20:00, except on Thursday the shop is open an hour longer. On Saturday the shop is open from 9:30 to 18:30. Moreover, the new inserted intervals should be 60 minutes long and using KNN imputation fills all other fields.

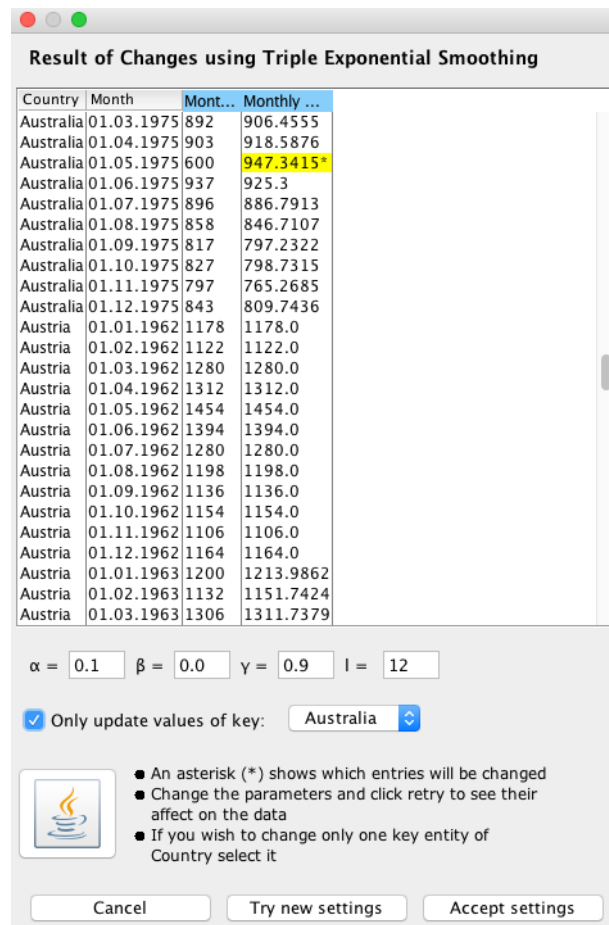
Average of Last N

This operation is based on simple [46] or weighted moving average⁵, however, we do not simply use the last n entries, but the users can specify which last entries they want to use. These last entries can be of the same time, for instance, if we detect an outlier on Thursday in the interval of 13:00 to 15:00, we will use the last n entries at the same time, i.e., Wednesday 13:00 to 15:00, Tuesday 13:00 to 15:00 and so on. Additionally, the user may choose if the last entries should occur on the same weekday (only Thursdays in our example), same day of the month (e.g., only the second Thursday of the month), or same day of the year (e.g., the 20th of December).

⁵<http://fxtrade.oanda.com/learn/forex-indicators/weighted-moving-average>

Exponential Smoothing


As discussed in Section 2.2.4, this is the implementation of various kinds of exponential smoothing. Depending on the data structure the user can decide whether single, double, or triple exponential smoothing fits his/her needs best. Besides previously detected outliers it is possible to define a certain tolerance for deviations between calculated and actual data. Figure 4.6 shows the preview that pops up after submitting a parameter setting. This is because of the relative complex operation especially for inexperienced users.



Country	Month	Mont...	Monthly ...
Australia	01.03.1975	892	906.4555
Australia	01.04.1975	903	918.5876
Australia	01.05.1975	600	947.3415*
Australia	01.06.1975	937	925.3
Australia	01.07.1975	896	886.7913
Australia	01.08.1975	858	846.7107
Australia	01.09.1975	817	797.2322
Australia	01.10.1975	827	798.7315
Australia	01.11.1975	797	765.2685
Australia	01.12.1975	843	809.7436
Austria	01.01.1962	1178	1178.0
Austria	01.02.1962	1122	1122.0
Austria	01.03.1962	1280	1280.0
Austria	01.04.1962	1312	1312.0
Austria	01.05.1962	1454	1454.0
Austria	01.06.1962	1394	1394.0
Austria	01.07.1962	1280	1280.0
Austria	01.08.1962	1198	1198.0
Austria	01.09.1962	1136	1136.0
Austria	01.10.1962	1154	1154.0
Austria	01.11.1962	1106	1106.0
Austria	01.12.1962	1164	1164.0
Austria	01.01.1963	1200	1213.9862
Austria	01.02.1963	1132	1151.7424
Austria	01.03.1963	1306	1311.7379

$\alpha = 0.1$ $\beta = 0.0$ $\gamma = 0.9$ $I = 12$

Only update values of key: Australia



- An asterisk (*) shows which entries will be changed
- Change the parameters and click retry to see their affect on the data
- If you wish to change only one key entity of Country select it

Cancel Try new settings Accept settings

Figure 4.6: Preview dialog of *Exponential Smoothing*. Yellow entries marked with an asterisk are detected outliers and will be replaced. A blue column header shows which columns are compared.

The users can try different settings and see the effect of a change in parameters before they update any values. Furthermore, they may choose to update one key set (in the example shown Figure 4.6 only outliers for Australia would be corrected), by entering a

key column beforehand.

Set Null and Impute

All identified outliers (by applying a check on the data) will be treated like NULL or dummy values and replaced with the selected technique (see Section 4.3.2.4 for available methods).

Cumulated Value Outliers

A special case of time-related values are cumulative values. For instance, if a company keeps track of the stock of inventory by adding up the numbers for each day. The sequence of the resulting values is cumulative, i.e., on the first day 20 items are in stock, on the second day 38 items are in stock (e.g., 30 which are newly stored and 12 of them left the stock) and so on.

All the methods regarding outliers mentioned so far will not work for that structure of data, thus, it is necessary to provide an operation to correct dirty data in this case. The values are determined by calculating the mean for the first n values. There are two options to compute the mean for the difference between two start points or for the difference between start and end point. Afterwards, the mean is used to correctify annotated entries, i.e., *previous value + mean* replaces the old value, whereas the mean is adjusted to the interval length depending on the chosen option. It is also possible to correct these outliers by using calculations (see Section 4.3.2.7).

4.3.2.6 Detect/Correct Duplicates

This part consists of two steps, at first possible duplicates need to be recognized and then they can be handled by three different approaches.

Detect Duplicates

There are two kinds of duplicates. The first type only concerns time-oriented data. Intervals have to be unique within each data set, unless there are more instances (see Section 4.3.2.1), for instance, shop A and B can be open at the same time.

The second sort of duplicates is what people usually understand as duplicates - two very similar or even equal entries within the same data set. In order to improve the search for duplicates it is possible to a) specify the search accuracy and b) assign a weight to each column, i.e., if certain columns are not relevant for a duplicate search (e.g., shop staff present at a certain interval might not be relevant, because there is always a very similar number of them in the shop) or if a column is more important than others. The method to calculate the similarity of one entry to another is only a slight modification of the distance calculation KNN uses.

Correct Duplicates

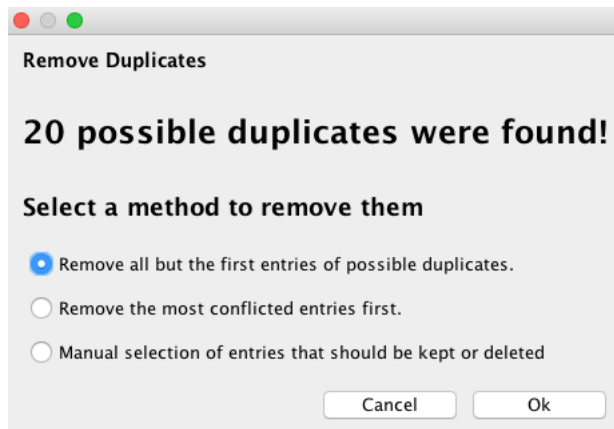


Figure 4.7: Possibilities to deal with detected duplicates. To make a decision easier the number of possible duplicates is shown. In case of a high number of conflicts it might prove tedious to look at each single group of possible duplicates.

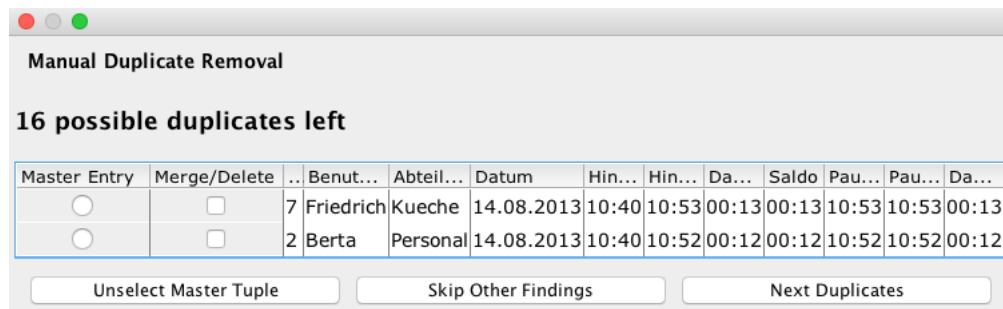


Figure 4.8: Manual removal of duplicates. These two entries show a high similarity, however, since the name and ID of the employees are different they are hardly real duplicates. Ultimately, the user can decide which entry should be kept and which should be deleted (if he/she found real duplicates). The entry that is deleted is also used to fill potential missing values from the master entry.

We implemented three ways to deal with possible duplicates (see Figure 4.7).

- Keeping the first entry of each duplicate. If the user is sure that all found duplicates are actually duplicates he/she can solve all conflicts by choosing this option.
- Removing those entries that cause the largest number of conflicts. This operation removes as many entries as necessary to clear all conflicts.
- Manual removal of duplicates (see Figure 4.8). The user has to look at each set of duplicates and decide whether two or more entries are duplicates or not.

4.3.2.7 Change Column

This section mainly consists of two parts. The first part is about transformations and changing the table structure and the second part offers a user interface for calculations.

Transformation

Figure 4.9 provides an overview which operations are to be understood as transformation. These transformations are:

- *Delete Column*: Delete a column and its values.
- *Rename*: Change the name of a column.
- *Split*: Split a column at a certain character (which gets lost within the operation, e.g., @-symbol for mail addresses into name and domain or space character to split a datetime column into a date and a separate time column) and adds a new column with a given name and type.
- *Merge*: The counterpart of the split operation. Merge two columns into an existing one by removing the second column. For example, a date and a time column can be merged into a single datetime column under the name of the date column. The time column is deleted within the process of merging.
- *Delete Content*: Delete everything within a column including the appearance of a certain sign. For example, removing the part of an e-mail address after (and including) the @-sign.
- *Interval Representation*: Changes the representation of a date or time column into an interval column. For example, if you have two time columns namely *Start* and *Stop*, you can display *Stop* as an interval with respect to the value at *Start*. For the value pair $(12:00; 12:30)$, we will get $(12:00; 30')$.

Calculation

Calculations make use of columns, certain rows of columns, and constants. They can be applied to all types of columns, but date and Boolean ones. The semantics for entering an equation is

```
'ColumnName' + constant + 'ColumnName[rowNumber]'
```

There is no limitation of constants or column names as long as they are of the right type. The type is defined by the target column of the calculation, i.e., targeting a time column allows the usage of date columns, time columns and time constants (e.g., 03:00). A number column allows whole number and decimal columns, but the constants must be of the correct type (e.g., a double value for an integer column will result in an error) and text columns allow any input.

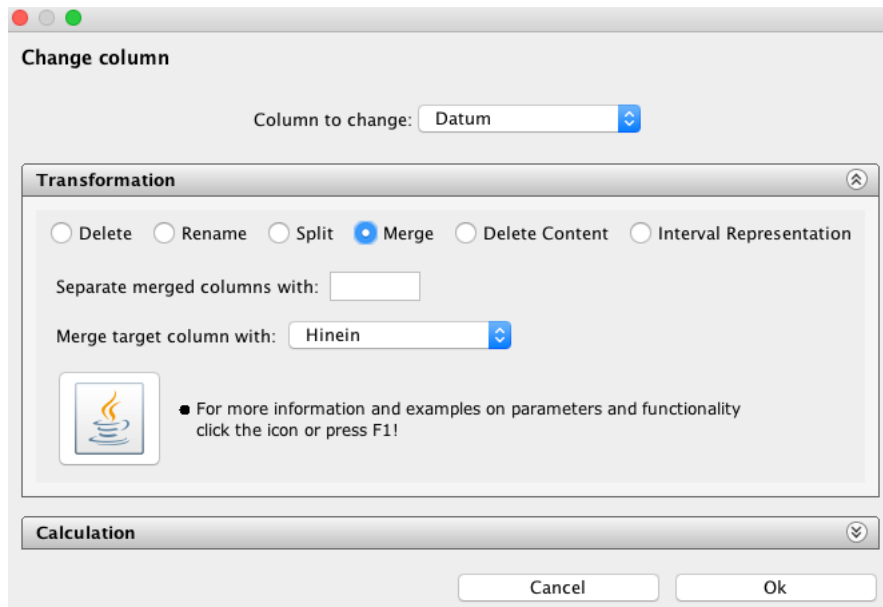


Figure 4.9: Transformations available for change column. Depending on the type of the selected column more or less operations are available. Here we can see a merge transformation for a date column. It can be merged with a time column (e.g., `Hinein` in the screenshot) and furthermore a separation character can be added (default would be a space sign).

There are two ways to reference rows. The user can either make a static reference to a field (e.g., `[1]` always uses the value of the entry in the first row) or a dynamic reference (e.g., `[-1]` always uses the value of the previous row of the current one). Using dynamic references enables the user to generate cumulated values, for instance, executing the equation `'Sales[-1]' + 'Sales'` on the `Sales` column results in adding up the sales figures. There is a second opportunity to create a sequence of values with respect to key columns. Either by using the command `sumUp(Equation, Key Column)` or by ticking the checkbox in the dialog. It will create a sequence for the values of the target column for each key value (e.g., summing up the total working hours for each employee).

The user can also decide whether he/she wants to update all rows with the entered equation or only annotated, i.e., erroneous, rows. This is especially important when dealing with very large datasets and in case the majority of values was computed correctly. An example would be to impute missing values via calculations (e.g., total working hours = end of work - start of work, should be correct for most entries but some entries are missing these values even though start and end values exist).

Date columns can only be used to determine results for time columns, i.e., if start and end intervals are given as dates and it is desired to calculate the difference between these two it is possible. Moreover, it is possible to include more than two date columns but with special rules. The first rule is that only the difference of two dates can be calculated

and the second one follows from the first rule that only an even number of date columns can be used. A short example would be that the total working hours equals the difference between the end and the start date of work minus the difference between the end and the start of the lunch break. The equation must look like this:

$$('Work\ End' - 'Work\ Start') - ('Break\ End' - 'Break\ Start')$$

Another type of calculation is especially tailored to time and date columns. That means that certain time units can be added to existing fields. For example, adding 5 hours to a time field in order to regulate a wrongly recorded time zone. Instead of using calculations it is recommended to make use of *Change Time* (see Figure 4.10). This is because adding hours to a time field might result in a change of date as well and this would not be possible with equations.

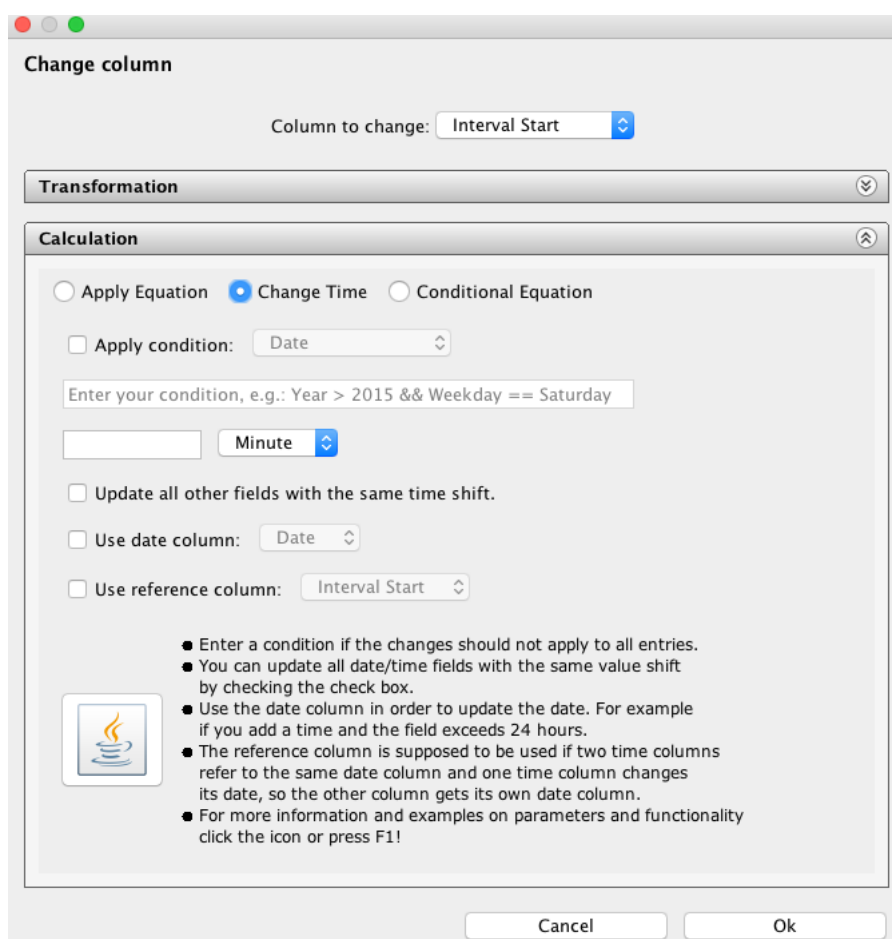


Figure 4.10: *Change Time* dialog. Options are to specify a condition to update only designated rows, to update all fields of the same type, as well as using a date and a reference column in case a change in time results in a change in date.

Using a reference column makes sense if you are expecting that a change of time results in having two times on two different days. Let's have a look at our work example again. If you have working hours that range from 09:00 to 18:00 (or later) and you make a time shift of seven hours. Then you will see working hours from 16:00 to 01:00 and only one date (depending on which column was selected it will be either the same day or the next day), which does not make much sense. Thus, by specifying a reference column a second date column will be added in case that any row will contain two times at two different dates. So, if both times were on the 1st December 2015, after the calculation 16:00 would be on the 1st December and 01:00 on the 2nd.

The third part of calculations are *conditional* calculations. The functionality is equal to the equations we described earlier, the only difference is that an equation can be executed on a specific set of rows. For example, a condition might be `Weekday == Monday && Month == 8`, so only entries that are on a Monday in August will be changed.

4.3.2.8 Add Column

This is a simple operation that adds a column with a given name and type. All values of the column will be missing values and can be imputed via calculation, as there are no values in the newly added column that can be used for an imputation method.

4.3.2.9 Change Rows

This feature contains the application of *Conditional Calculations*, but also the possibility to *Remove* certain rows. For example, we have some erroneous date values (e.g., date in the future) and to get rid of them we can use `year > 2015` as condition and choose to remove all rows that fulfill this condition. Another option is to remove all erroneous rows that fulfill a condition, i.e., removing all entries for a certain condition that contain missing values, outliers, or any other data problem.

4.3.2.10 Edit Intervals

Editing Interval operations are tailored to meet the specific requirements when cleansing time-oriented data. The operations available under this menu point are:

- *Standardize Interval Length*: using this operation the user can change all intervals to a desired length. However, since time-related values depend on the length of the interval the user has to specify which columns should be adjusted to the new size as well. For example, while there are three salesmen in the shop the whole day (and therefore, no adjustment is necessary), the number of sales depends on the interval duration. The only problem is if the interval size gets smaller the sales are only estimated values, i.e., if one reduces the size from one hour intervals to 20 minutes and within the one hour interval 6 items were sold, each 20 minute interval will show 2 sales. Even though it is quite possible that there were a different number of

sales for each interval within this hour. However, choosing to not adjust the values will result in 6 items sold every 20 minutes, which is obviously wrong.

- *Group By Date*: by applying this transformation the user will get an overview of all intervals within a day/month/year. Thus, if a day contains 24 one hour intervals all will be shown as one interval within a day and as it is possible with standardizing the interval length, column values can be adjusted to this new interval length, i.e., summing up the values of all 24 one hour intervals.
- *Set Limits*: setting limits allows the user to avoid intervals at given times. For example, if the opening hours of a shop are from 08:00 - 18:00, any intervals outside this time span are most likely erroneous. Thus, the user can set possible intervals for each day (or weekday, weekend, whole week) and decide whether he/she wants to delete them (if the interval is completely outside the given boundaries), adjust them (if the interval is partly within the boundaries, i.e., 17:00 - 18:30 becomes 17:00 to 18:00), or annotate them as outliers. These boundaries are not saved for further entries or dates, but have to be applied again in case further entries are added.
- *Minimum Off-Time*: such off-times are common in almost every business, but easily overseen in factories where people work in shifts. That is so because, with many different end and start times and possibly with a large number of workers it is hard to check each entry. Setting a minimum off-time for an employee (the key column in this example), for example, eight hours, will find all entries that violate this condition. The user has three options to deal with that, namely
 - deleting the entries because they are erroneous,
 - annotating them to further investigate the situation (e.g., if there were real violations or simply wrong recordings),
 - shift them in time so they do not violate the condition any longer.
- *Split Intervals for each Day*: is appropriate if the user does not want to have any entries that overlap days, i.e., an interval starts at 20:00 on Monday and ends on 6:00 on Wednesday. Hence, it is possible to split this interval into three parts: 20:00 - 23:59 on Monday, 0:00 - 23:59 on Tuesday and 0:00 - 6:00 on Wednesday. This is even more important when dealing with time columns, because if an interval starts at 16:00 and ends at 12:00 this is either an error or the interval ends on the next day. So splitting this interval will give us an additional date column that records the day after the interval start (e.g., 2015-12-02 if the interval start is at 16:00 on 2015-02-01).
- *Correct Overlapping Intervals*: If someone wishes to have no overlapping intervals (i.e., one interval ends at 12:00 and another starts at the exact same time). The corrective method is to let the previous interval end one minute early (11:59 instead of 12:00).

4.3.2.11 History

The *History* enables the users to view all previous operations they executed. Furthermore, they can reset the data to any point within these steps, for instance, undoing the last three operations (see Figure 4.11). In case the users changed their mind and only wanted to undo the last two operations they can redo one operation (or all three they undid earlier). However, once a new operation was applied on the data set the future history will vanish and the new operation will be added on top of the previous operations. Technically, it would be possible to redo operations even after executing a new operation, but it might lead to different results as earlier (e.g., imputing missing values might not be the same after applying a calculation on a column).

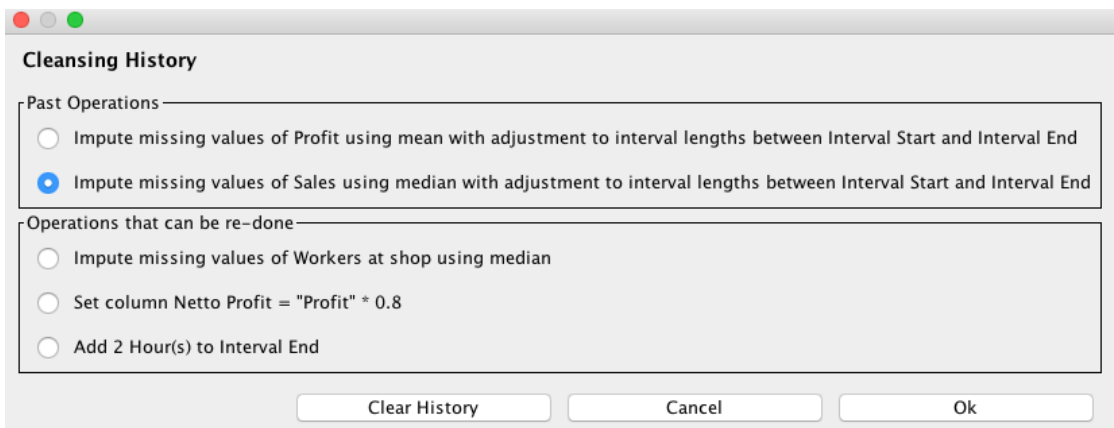


Figure 4.11: *History* dialog. Opening the history after undoing the last three options: they are available for redoing while the previous operations still can be undone.

4.3.2.12 Help

The *Help* is available at any point during the prototype. It is either accessible by pressing F1 or clicking the icon in the dialog. The tab of *Help* will adjust to the current position, i.e., if you press F1 in the table view it will open a guide containing a list of the available commands and if the user clicks the button in the, *Correct Implausible*, dialog the help will show all information regarding removing outliers. The dialog (see Figure 4.12) usually shows information about each parameter for every operation and gives some further explanation should the concept of a parameter be harder to grasp. We figured out providing a help function is important, because sometimes labeling a parameter, i.e., text field, so everyone understands it immediately, is not easy. Especially the terms *key column* and *reference column* might be unclear to the user. While the key column (see Section 4.3.2.1) is often equal to the key column as understood in SQL⁶, this is not always the case, i.e., a foreign key could be the key column for a certain operation. For

⁶http://www.w3schools.com/sql/sql_primarykey.asp

example, if a table contains the columns ‘shop’ (primary key) and ‘employee’ (foreign key) and the user wants to set the minimum off-time for an employee the key column would not be equal to the primary key. Instead the key column would refer to the foreign key, because ‘employee’ is the main focus of the operation whereas it does not matter which ‘shop’ he/she is employed at.

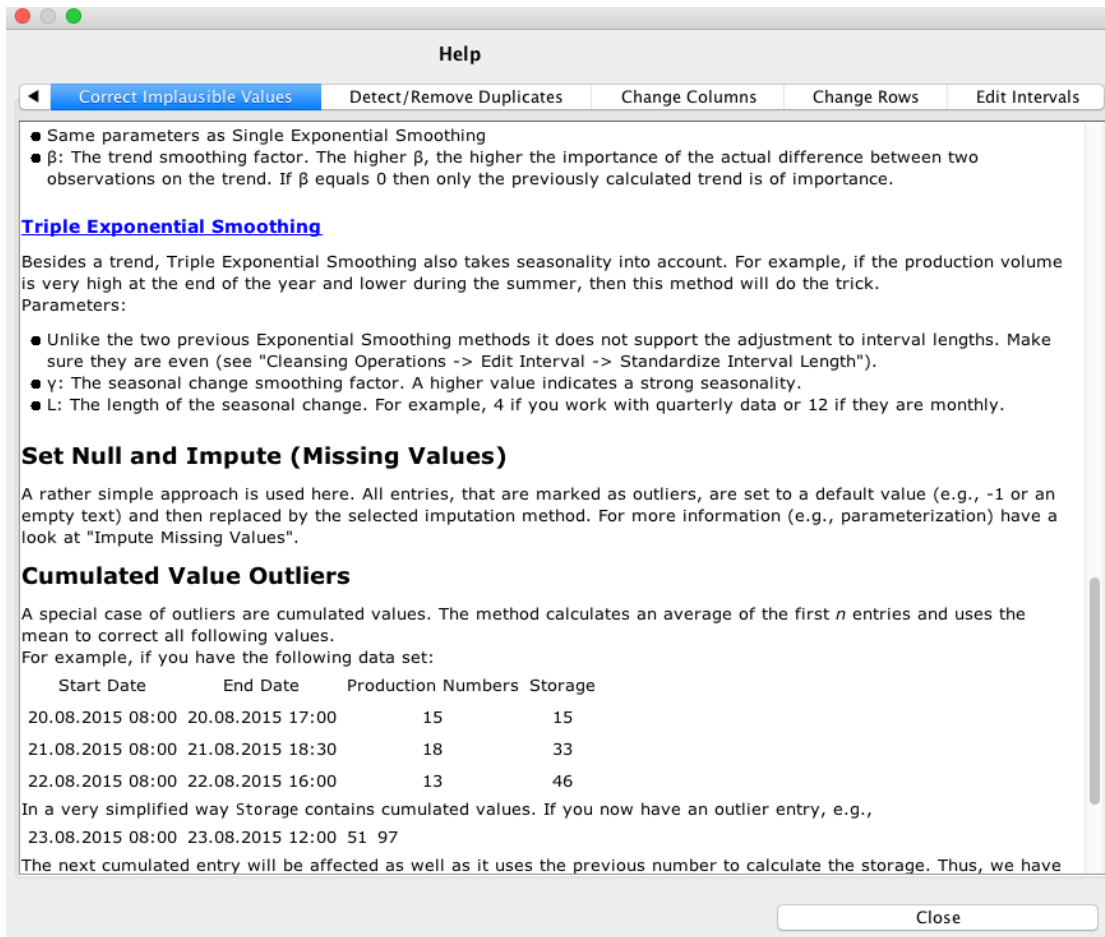


Figure 4.12: *Help* dialog. The dialog consists of explanations, links, and parameter descriptions for each category of cleansing operations.

Evaluation

In order to evaluate the usefulness of QualityTime we will look at it in two different ways. The first one is to see how many data problems can be tackled by applying the operations of the prototype. The second one is to conduct a heuristic evaluation that aims at finding out how effective the tool can be used in terms of usability.

5.1 Operation Coverage

In order to answer the research questions *which types of errors can be handled by automatic operations and how to support this task?* and *which types of errors needs to be handled manually and how to support this task?* we have to look at the operations QualityTime provides and which tasks can be solved by applying them.

We have already covered the supported operations with regards to the time-oriented data quality problems determined by Gschwandtner et al. [23] in Section 3. However, there are several problems that cannot be solved by using QualityTime. The reason behind this is that either a lot of knowledge is required concerning the ‘dirty’ data or the input cannot be processed properly. For a detailed list of unsupported errors see Table 5.1.

Overall, QualityTime can handle 87% (29 of 33) of single source problems with ‘dirty’ time-oriented data and additionally, provides means to prepare data to merge them with other sources (e.g., standardize the schema of two data sources).

5.2 Usability Testing

To evaluate the cleansing operations QualityTime provides, we decided to conduct a usability study. The study should cover several operations that may be executed within a realistic scenario and show their usefulness to the participants. We conducted a heuristic usability inspection [43] with three participants having different background knowledge

Category	Title	Description (<i>Example</i>)	Reason for Nonsupport
Outdated data	Temporal outdated data	Only old versions available <i>Same values from last year</i>	It is impossible to derive the correct values for duplicates of this sort. One option might be to use <i>exponential smoothing</i> , but that will only lead to some success if the data of many previous years are correct.
		New version replaced by old version <i>Project plan tasks overwritten by prior version</i>	Even though QualityTime provides a <i>history</i> it only helps to undo applied cleansing methods. Changes must be saved in order to be kept, otherwise they will be lost as old data is not stored.
Ambiguous data	Abbreviations or imprecise or unusual coding	Ambiguous time/interval/duration due to short format <i>(Date: 03-06-05) vs. (Date: 06-05-03); 5'interval encoded as '09:00': (interval: 8:55 - 9:00) vs. (interval: 9:00 - 9:05)</i>	As the date format is determined while reading the table it is not clear how the date should be interpreted. It would be necessary to define the date format beforehand, which is not part of the prototype. Thus, keeping a precise date format is a prerequisite. Also intervals should be kept within two columns, i.e., (start: 9:00 and end: 5') would be clear. A start time without an end time cannot be interpreted.
		Extra symbols for time properties <i>+ or * or 28:00 for next day</i>	+ and * would not be recognized as a time/interval and therefore, the prototype would not be able to interpret them accordingly. Dealing with 28:00 would be possible but requires two steps. At first the date would have to be increased for all times greater 24:00 and then 24 hours have to be subtracted from the time.

Table 5.1: (Single Source) Problems defined by Gschwandtner et al. [23] that cannot be solved by using QualityTime.

about cleansing tools. The first participant has already worked a lot with Open Refine [29] (see Section 2.3.4), the second one was familiar with the concepts of cleansing, whereas the third participant has not used any cleansing tools so far. In order to evaluate the prototype this variety was welcome, because ultimately anyone should be able to use the tool after all. Nevertheless, eventually data analysts will have to use the tool during the task of data preparation.

5.2.1 Testing Method

We used a set of 10 heuristics especially tailored to evaluate usability in the context of visualization approaches by Forsell and Johansson [16]. The participants absolved the study separately and were asked to assign each problem they detect to a heuristic as well as rate its severity.

After an explanation of the heuristics and what is expected of them, the participants

were introduced to QualityTime and the data set. The data represented sales figures of two imaginary coffee houses. It contained some errors that could easily occur within a real data set and the tasks during the study were targeted at solving them. The study contained seven tasks, which were logically ordered, i.e., it would not make much sense to complete task three before task one and two, because it is a logical consequence. These tasks contained the following operations:

1. Splitting intervals in order to get homogenous data.
2. Apply a calculation on a date column that has to fulfill a certain condition.
3. Limit the interval lengths, so the data represents the correct opening hours of a store.
4. Impute missing intervals so the data does not contain any gaps.
5. Correct implausible values to get rid of outliers.
6. Add a column and use an equation in order to add useful information to the data.
7. Change the date format.

Every time the participants encountered a problem or were not able to solve a task on their own (within reasonable time) they were asked to define the problem (and state a possible solution) and rate its severity. Afterwards, we helped them with their problem. For example, if the user was not able to find the correct menu, the naming of menus was explained to them, i.e., that operations regarding time were most likely to be found under *Edit Intervals*.

5.2.2 Results

All questions and statements were recorded while we monitored the participants. All usability problems the participants found were rated with a low to medium severity (one to three). The most prominent problem was *Orientation and Help* (9 out of 19 issues), followed by *Spatial Organization* (3 out of 19), and *Information Coding* (2 out of 19). The most important findings were:

- **Orientation and Help:** At the beginning all participants struggled to find the right menus for the task to solve (e.g., a common mistake was that they looked within the menu *Correct Implausible Values* when an interval should be adjusted). However, this was due to the fact that the preparation phase was rather short and they were asked to solve tasks before getting familiar with the prototype. Once they completed some tasks and understood the structure of the menu they were able to complete tasks a lot faster.

*‘At first it was confusing that time is treated differently,
but I got used to it now.’ (A1)*

- **Spatial Organization:** Some dialogs are structured differently (e.g., two rows of available operations instead of just one, see Figure 5.1), which confused the participants. Additionally, the table used to preview the results of *exponential smoothing* did not resize properly.
- **Information Coding:** Two participants thought it would be helpful to highlight entries that changed after applying a cleansing operation.
- **Acceptance of Command Line:** All participants preferred to use the menu over the command line, which is due to the fact that the command line is designed to cater the needs of more experienced users and getting used to the syntax would require more time than to click through menus.
- **Acceptance of Non-Cleansing Features:** Both non-cleansing features, *Help* and *History*, were used and appreciated by the participants. The first, and most experienced, participant was confident he would have been able to solve all tasks without any assistance, but with more time and the help menu.

*‘The history is great, very meaningful.
The help is amazing as well. All operations are explained in a sufficient way.’ (A2)*

- **Overall:** The prototype in general received very positive feedback regarding functionality and flexibility (especially with the introduction of calculations).

*‘I really liked the freedom the tool gives to the user.
It is very flexible because of calculations.’ A3*

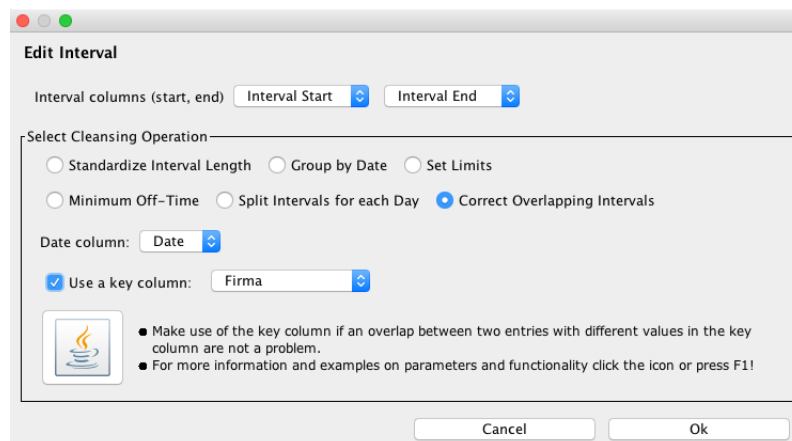


Figure 5.1: *Edit Interval* dialog. The available operations extend across two rows, which was confusing for the participants as they thought the second row was an additional setting after selecting an operation in the first row.

Future Work & Conclusion

6.1 Future Work

6.1.1 Enhancing Usability

During the usability study (see Section 5.2) several issues were identified that could be improved to make it easier for the user to navigate through the tool and find required operations faster.

- **Highlighting:** The feedback via the status bar is nice, but to mark changed rows until the execution of further operations would be helpful (e.g., using a color scheme similar to GitHub¹: red for removed columns/rows, yellow for modified entries and green for newly added columns/rows).
- **Menu Structure:** Instead of using one dialog to impute missing values and intervals, it would be more user friendly to use two separate dialogs and two menu entries. The same is true for changing a column - one menu entry for transformation and another one for calculation.
- **Labeling:** While it is hard to find meaningful labels for different kinds of menus or variables, we could try to improve the names with a small group of representative users. For example, one participant did not understand interval as time interval, but as a mathematical set of variables. The background of a user can alter the interpretation and as developer one can easily suffer from tunnel vision.
- **History:** The history is only visible in a dialog, but one participant suggested to (add an option to) make it visible the whole time (as it is implemented in Wrangler [32], see Section 2.3.5).

¹<https://github.com>

6.1.2 Improving Algorithms

Although the ,*Detecting Duplicates*, operation is working quite nicely when specifying different weights for each column (including ignoring columns by assigning to them a value of zero), the computation is slow when it comes to larger data sets that contain a lot of similar data. For instance, without any weights detecting duplicates in a table that contains about 500 entries and working times of each employee (they often start at very similar times), takes about 4 minutes to complete on a relatively fast computer (MacBook Pro² (late 2013): 2GHz Intel Core i7, 8GB 1600 MHz DDR3, 512GB SSD). Even though clustering is applied to avoid unnecessary comparisons of rows it does not work as desired in data sets with small differences. Additionally, displaying the current status of the comparisons might be helpful (e.g., comparing row 5 to row 488, 11% complete ...). Currently, only a busy mouse pointer and a note in the help overview inform the user that the process might take a while to complete.

6.1.3 Data Sources

At the moment all data is obtained from .csv files, but a lot of data is stored in different environments (e.g., relational databases (mySQL³, MS SQL⁴, Oracle⁵, ...) or even document-oriented databases (MongoDB⁶, DocumentDB⁷, ...). Since in TimeProfiler the data is stored in a so-called *DirtyTable*, only the way how the input is processed has to be changed, i.e., switching the model or rather the way to retrieve the model. However, the user would have to specify the table schema beforehand and it could become difficult and costly if a table references to many other tables.

6.2 Conclusion

In this thesis a research prototype to cleanse erroneous time-oriented data was developed and evaluated. The demand for such a tool was identified by literature research. The most important operations were found out by analyzing a taxonomy of time-oriented data by Gschwandtner et al. [23]. Some methods work equally well for any kind of data including time-oriented data while others are not useful without the information of time and intervals. Before implementing the cleansing operations, they were designed and refined, and thought was put into the needed parameterization. The prototype was then evaluated by a heuristic usability study to detect potential usability deficiencies.

The prototype was developed in order to answer the research question, *‘How to support data analysts dealing with erroneous time-oriented data?’*: It provides means to cleanse many time-oriented data quality problems mentioned in [23] and the conducted

²<http://www.apple.com/macbook-pro/specs-retina/>

³<http://dev.mysql.com/>

⁴<http://www.microsoft.com/en-us/server-cloud/products/sql-server/>

⁵<https://www.oracle.com/database/index.html>

⁶<https://www.mongodb.com/>

⁷<https://www.documentdb.com/sql/demo>

study proves its suitability to tackle these problems. The further questions that arose from the main research question can also be answered:

- *Which methods need to be applied in order to improve the data quality?*
We derived existing approaches from a literature review (see Section 2.2) and adapted specific methods for cleansing time-oriented data. Moreover, we added further methods (e.g., calculations) during the design process (see Section 3). Thus, we end up with a collection of cleansing methods suited to improve data quality (see Section 4.3.2 for a complete list of implemented cleansing methods).
- *Which types of errors can be handled by automatic operations and how to support this task?*
We identified a comprehensive list of types of errors that can be supported by (semi-) automatic operations from literature research (see Section 3.2 for a complete list of error types handled by our prototype). However, there are also types of errors that are hard to tackle by automatic methods. These include all types of errors where human judgment is required, such as implausible values.
- *Which types of errors need to be handled manually and how to support this task?*
We evaluated the prototype regarding supported, but also unsupported operations. In Table 5.1 we give a list of errors that need to be handled manually (such as ambiguous time due to short format) together with explanations why they cannot be supported by automatic means.

In context of this thesis we have designed and developed QualityTime, a scientific prototype to tackle the problem of cleansing time-oriented data. In addition we successfully evaluated the prototype and derived answers to our research questions.

Appendix

A.1 Dialogs and Parameters

Parameter names in italic are optional. However, some parameters are only needed in some occasions, i.e., the date field is often required when interval start is not a datetime value, but only contains a time. For an explanation on how and when to use a key column see Section 4.3.2.1.

A.1.1 Impute Missing Values

Parameter	Explanation
Column that contains missing values	Name of the column that should be cleansed of missing values.
<i>Use interval duration to adjust missing values</i> <i>(Start, end)</i>	Tick if the mean of a number column should be adjusted to the respective interval size. The names of the corresponding start and end interval column.
<i>Use intervals instead of raw columns</i> <i>Reference column</i>	Tick if a time column should be adjusted to a reference column instead of simply imputing its mean. The name of the time column that has a reference to the column that should be cleansed of missing values.

Parameter	Explanation
<i>Date column</i>	The name of the date column that is needed if criteria for the mean calculation are given.
<i>Criteria</i>	A list of criteria (same time, same weekday/same day of the month/same day of the year) that an entry must fulfill in order to be heeded towards the calculation of the mean. Depending on the criteria a date column and/or the two interval columns have to be specified.

Table A.1: *Imputation of Missing Values - Mean* parameters.

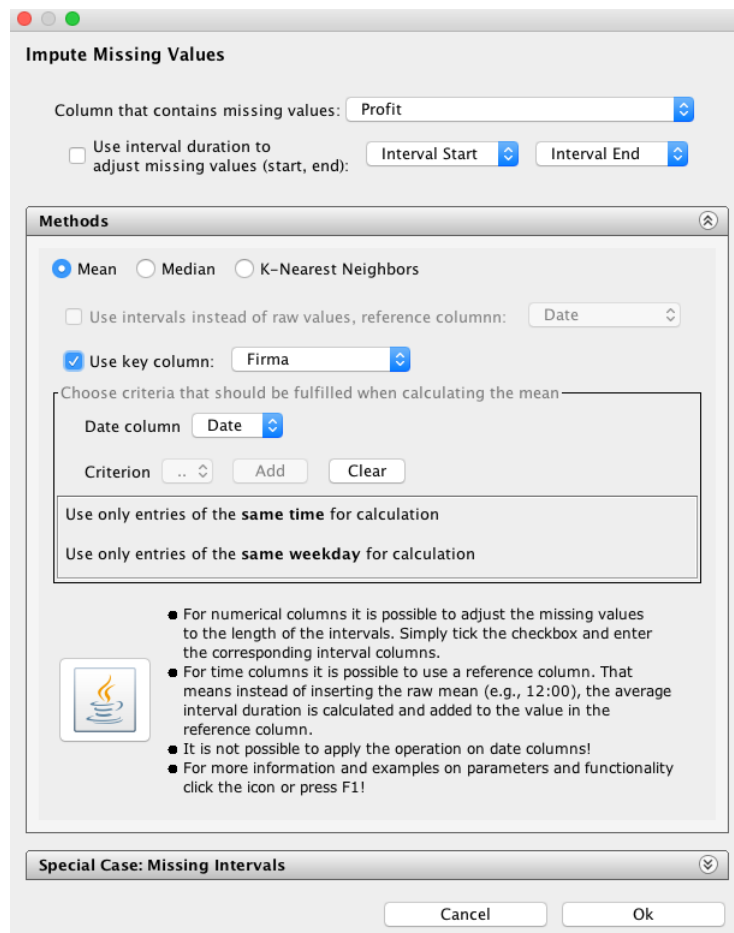


Figure A.1: *Imputation of Missing Values - Mean* dialog for a number respectively a time column.

Parameter	Explanation
Column that contains missing values	Name of the column that should be cleansed of missing values.
<i>Use interval duration to adjust missing values</i> <i>(Start, end)</i>	Tick if the mean of a number column should be adjusted to the respective interval size. The names of the corresponding start and end interval column.
<i>Use intervals instead of raw columns</i> <i>Reference column</i>	Tick if a time column should be adjusted to a reference column instead of simply imputing the median. The name of the time column that has a reference to the column that should be cleansed of missing values.

Table A.2: *Imputation of Missing Values - Median* parameters.

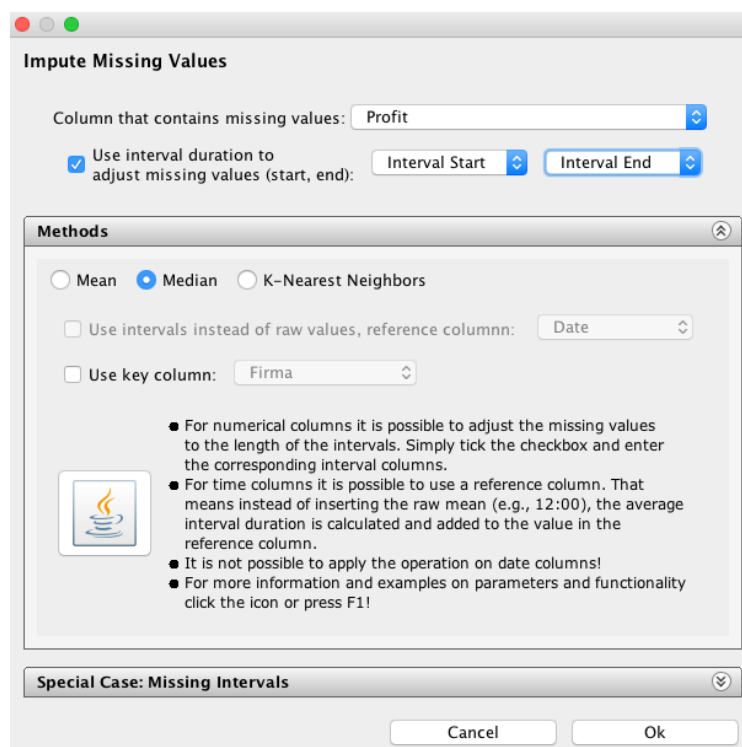


Figure A.2: *Imputation of Missing Values - Median* dialog for a number respectively a time column.

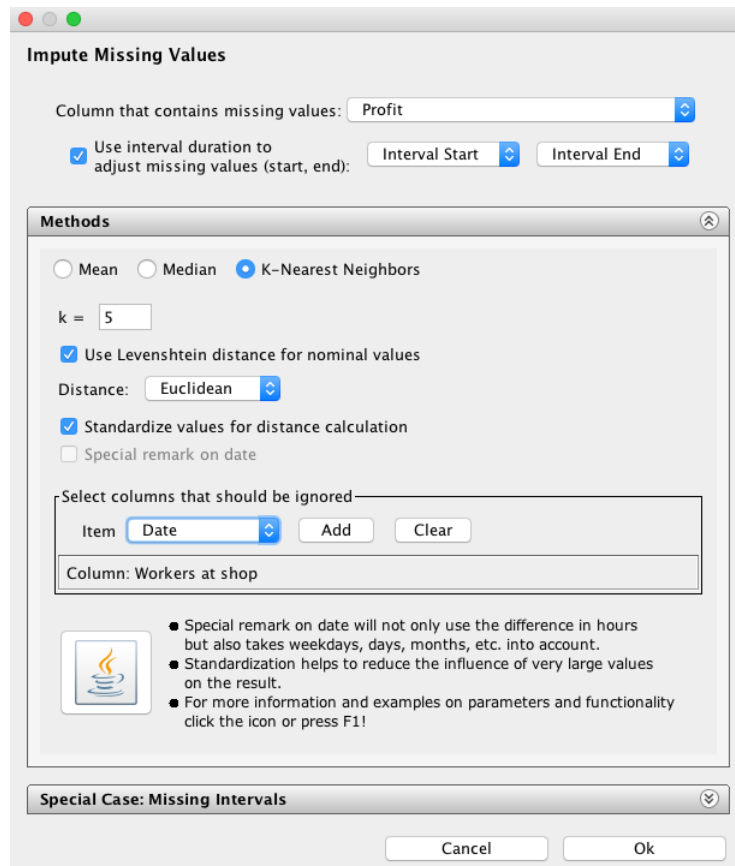


Figure A.3: *Imputation of Missing Values - KNN* Dialog for a number column.

Parameter	Explanation
Column that contains missing values	Name of the column that should be cleansed of missing values.
<i>Use interval duration to adjust missing values</i> <i>(Start, end)</i>	Tick if the mean of the k nearest neighbors of a number column should be adjusted to the respective interval size. The names of the corresponding start and end interval column.
k	Number of nearest neighbors that will take into consideration for the mean computation.

Parameter	Explanation
<i>Use Levenshtein distance for nominal values</i>	Tick if the Levenshtein distance should be calculated instead of using 0 (= match) or 1 (= different) for nominal values (e.g., Thomas vs. Tomas, Levenshtein = 1, default = 1; Thomas vs. Peter, Levenshtein = 6, default = 1)
Distance	Choose between Euclidean and Manhattan Distance [54].
<i>Standardize values for distance calculation</i>	All distances between values are between 0 and 1, thus, the impact of columns containing huge values is reduced (e.g., age would have a lower impact than salary, because it is most likely between 18 and 65, whereas salary could be between 20,000 and 60,000)
or	
<i>Special remark on date</i>	Instead of comparing dates hour-wise day, month, year, hour and minute are compared. Therefore, 2015-01-01 10:00 and 2014-01-01 10:00 would have a smaller distance than 2015-01-01 10:00 and 2014-12-12 23:00.
<i>Select columns that should be ignored</i>	Names of the columns that will not count towards the distance calculation.

Table A.3: *Imputation of Missing Values - KNN* parameters.

Parameter	Explanation
Interval start and end and	The names of the interval start and end columns.
Date column	The name of the date column (if the interval start column is time only).
<i>Use standardized interval length</i>	The length of each imputed interval (if possible) in minutes.
<i>Avoid overlapping intervals</i>	The inserted intervals will end before the next one starts, i.e., if an interval ends at 14:00 and another one starts at 14:00 they basically overlap in a strict sense. This option avoids such scenarios and the interval will end at 13:59.
<i>Fill non-interval/date fields using</i>	Choose between mean, median and KNN. As a new entry will only contain a date and two times all other fields will be empty, i.e., missing values. This option will see them automatically filled.

Parameter	Explanation
Select intervals that should exist for different days	Specify the desired interval frame for each day. You can also use weekday, weekend and week to generalize your specifications. A more precise input will ignore the less precise one, i.e., Monday is more important than weekday and weekday is more important than week.
Criteria	If mean imputation should be applied to newly inserted intervals then special criteria may be specified.

Table A.4: *Imputation of Missing Intervals* parameters.

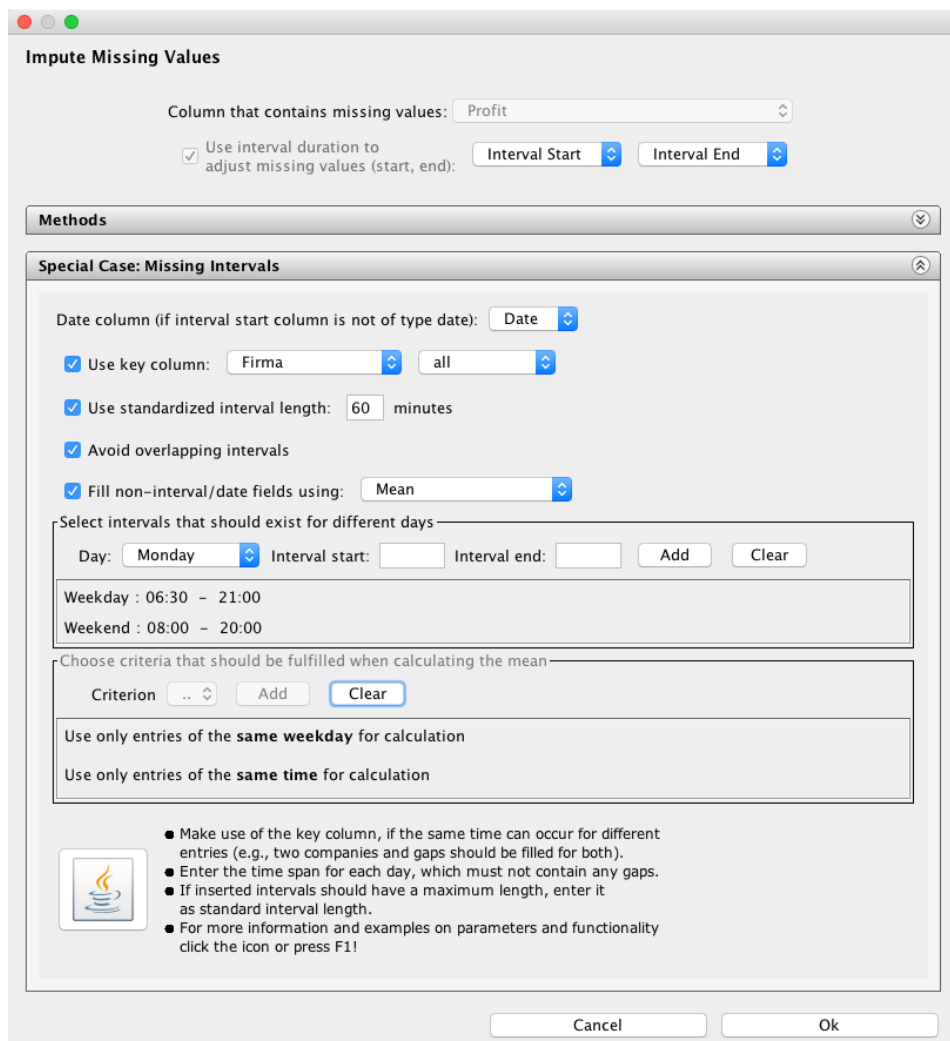


Figure A.4: *Imputation of Missing Intervals* dialog.

A.1.2 Correct Implausible Values

Correct Implausible Values

Column with incorrect values: Profit

Select Cleansing Method

Average of Last N Moving Average Set Null and Impute Cumulated Value Outliers

n = 5

Use weight on average

Use key column: Date

Date column: Date


Interval start: Interval Start Interval end: Interval End

Choose criteria that should be fulfilled by the previous n entries

Criterion .. Add Clear

Use last n entries that have the Same Time

Use last n entries that have the Same Day of the Month



- Please note that you have to run "Check -> Time relative values" first!
- If the values are not sorted, please do so before running the operation!
- Usage of the key column: if there are two or more rows with identical dates e.g., several company names
- Weight on average guarantees that the most recent observations have a greater impact than the older ones
- For more information on parameters and functionality click the icon or press F1

Cancel Ok

Figure A.5: *Correct Implausible Values - Average of Last N* dialog.

Parameter	Explanation
Column with incorrect values	The name of the (number) column that contains implausible values.

Parameter	Explanation
n	The number of the last entries that fulfill the specified requirements under criteria.
<i>Use weight on average</i>	The most recent entries count more towards the calculation of the new value than the older ones.
Date column and Interval start, interval end	The name of the date column (if the interval start column is time only). The name of the column that represents the interval start respectively the end of the interval.
Choose criteria that should be fulfilled by the previous n entries	The possibilities are: <ul style="list-style-type: none"> • same time, i.e., same interval start and end but different date, • same day of the week, i.e., if the incorrect value belongs to a Monday all Mondays are taken into consideration, • same day of the month, i.e., all entries that are on the same day and week of the month (e.g., 3rd Wednesday), • same day of the year, i.e., all entries on the same date, but different year. <p>Same time can be combined with each of the other three choices whereas only of these three can be selected at once.</p>

Table A.5: *Correct Implausible Values - Average of Last N* parameters.

Parameter	Explanation
Column with incorrect values	The name of the (number) column that contains implausible values.
Method	Single, double or triple exponential smoothing (see Section 2.2.4).
Method specific parameters	α , β , γ and l depending on the chosen method.
<i>Adjust the calculated values to the intervals (from, to)</i>	In case the values are assigned to intervals of different lengths, the calculation will be executed with remark on the interval length (only available for single and double exponential smoothing).
<i>Replace values that deviate from the calculated average by (in %)</i>	This criteria can be used to detect outliers that were not detected through checks.

Table A.6: *Correct Implausible Values - Exponential Smoothing* parameters.

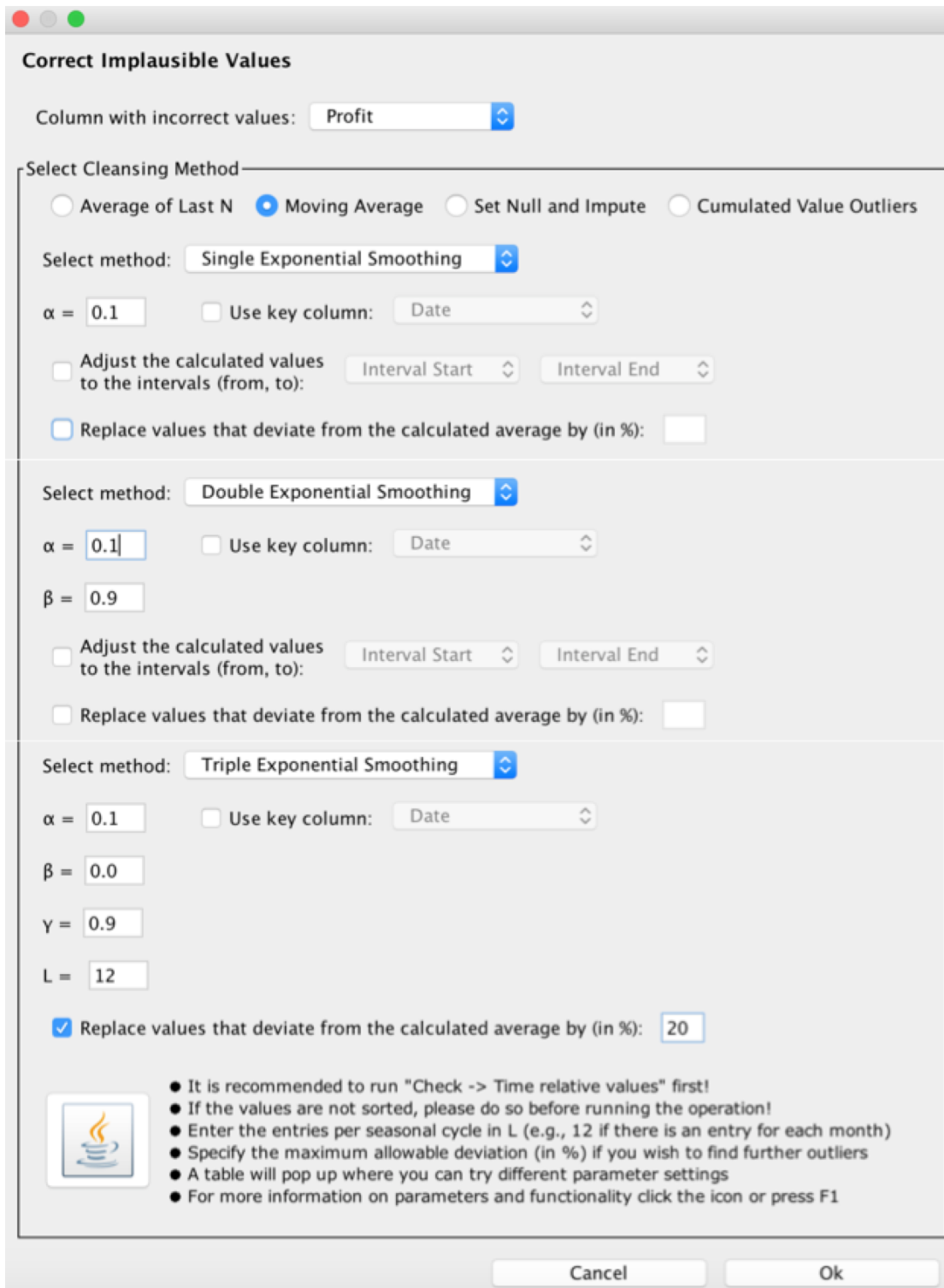


Figure A.6: *Correct Implausible Values - Exponential Smoothing* dialog.

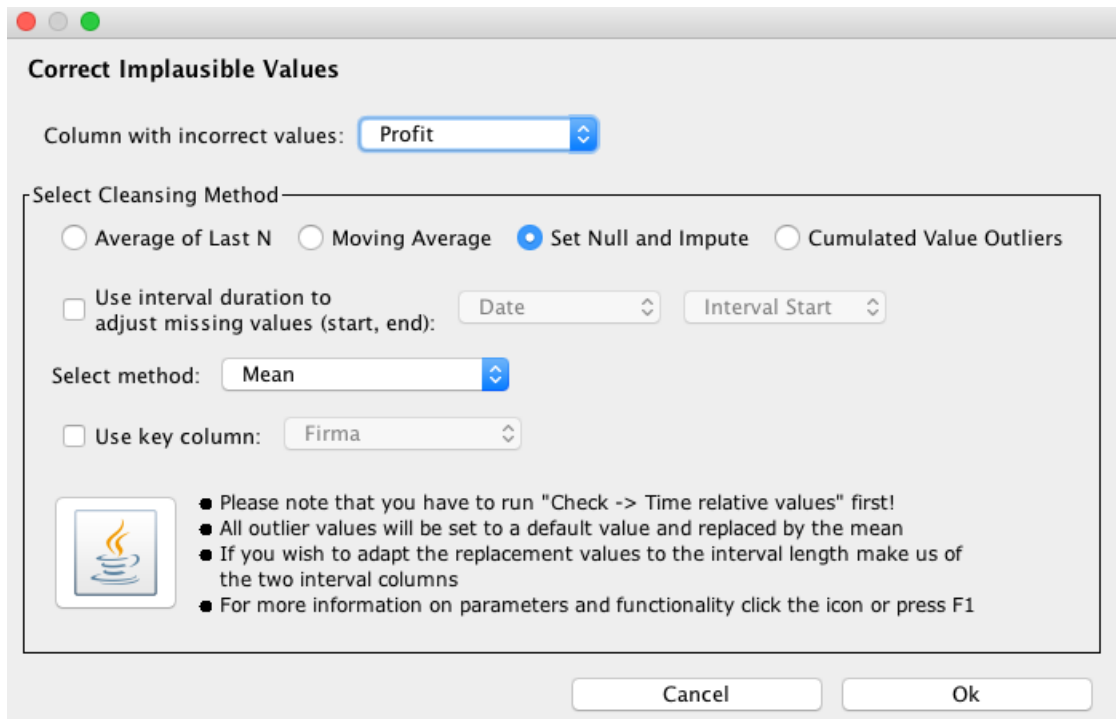


Figure A.7: *Correct Implausible Values - Set Null and Impute* dialog.

Parameter	Explanation
Column with incorrect values	The name of the (number) column that contains implausible values.
<i>Use interval duration to adjust missing values</i> <i>(Start, end)</i>	Tick if the mean of the k nearest neighbors of a number column should be adjusted to the respective interval size. The names of the corresponding start and end interval column.
Method	Mean, Median or KNN for imputation after setting outliers to NULL.
Method specific parameters	See Section A.1.1 for further information.

Table A.7: *Correct Implausible Values - Set Null and Impute* parameters.

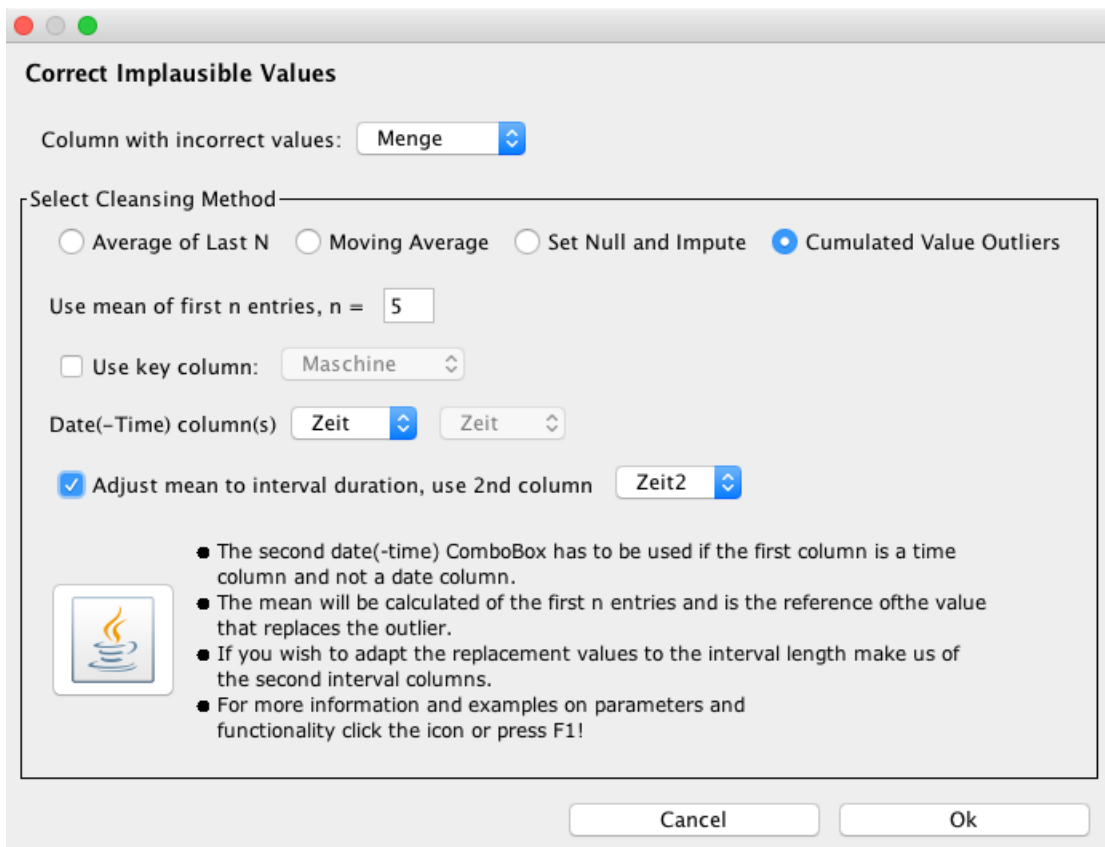


Figure A.8: *Correct Implausible Values Cumulated Value Outliers* dialog.

Parameter	Explanation
Column with incorrect values	The name of the (number) column that contains implausible values.
n	The number of entries that are used to calculate the mean, which will be used on the following entries.
Date(-Time) column(s)	The name of the column that represents the start of the interval. If the column is a date no further input is required, if it is of type time a date column must be specified.
<i>Adjust mean to interval duration</i>	Whether the mean should be calculated with respect to the interval duration.
<i>2nd column</i>	The name of the column that represents the end of the interval.

Table A.8: *Correct Implausible Values - Cumulated Value Outliers* parameters.

A.1.3 Deduplication

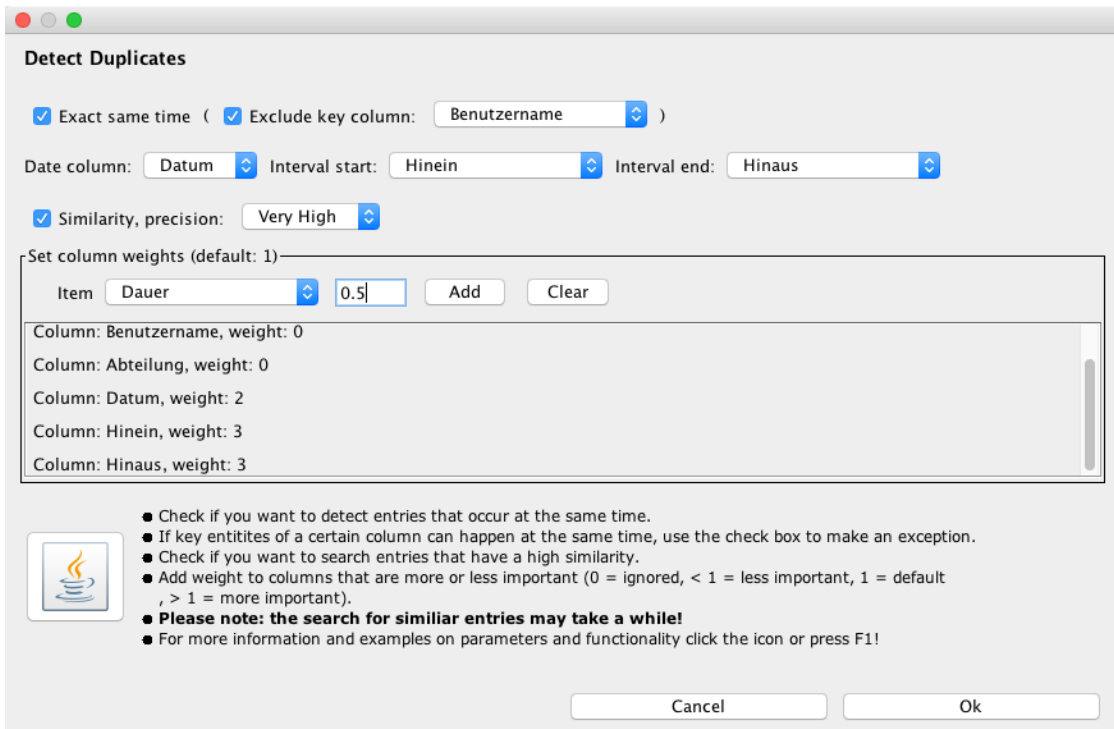


Figure A.9: *Deduplication Detection* dialog.

	Parameter	Explanation
Same Time	<i>Exact same time</i>	Whether it is forbidden that two entries occur at the same time.
	<i>Exclude key column</i>	Unless their key column entry is different.
	<i>Date column</i>	The name of the date column (not needed if interval start is a date).
	Interval start	The name of the (date-)time column that represents the interval start.
	Interval end	The name of the (date-)time column that represents the interval end.

	Parameter	Explanation
Similarity	<i>Similarity</i>	Whether entries should be checked on how similar they are.
	Precision	How similar they should be (it is suggested to start with a higher similarity and lower it if the found matches were not satisfying)
	<i>Set column weights</i>	How much weight should be assigned to a column. Default value is 1, 0 means ignoring a column and every value greater 1 is especially important for duplicate detection. <i>It is recommended to assign weights, especially ignoring unimportant columns.</i>

Table A.9: *Deduplication Detection* parameters.



Figure A.10: *Deduplication Method* dialog.

	Parameter	Explanation
Strategy	Remove all but the first entries of possible duplicates	As the name says; the first entry is always the entry that appeared first, i.e., the one with the lower row number (does not require further input).
	Remove the most conflicted entries first	Entries that conflict with many others are removed first (does not require further input).
	Manual selection of entries that should be kept or deleted	See Figure A.11 and Table A.11.

Table A.10: *Deduplication Detection* parameters.



Figure A.11: *Manual Duplicate Removal* dialog.

Parameter	Explanation
<i>Master Entry</i>	The entry to be kept in case the found duplicates are actual duplicates.
<i>Merge/Delete</i>	The entry/entries that should be deleted or in case the master entry does contain empty fields these will be used to fill them.
<i>Next Duplicates</i>	Continue to the next possible matches.
<i>Skip Other Findings</i>	Cancel the cleansing operations. Every match that was marked as duplicates, i.e., master and merge/delete entries were specified, will be merged.

Table A.11: *Manual Duplicate Removal* parameters.

A.1.4 Change Column

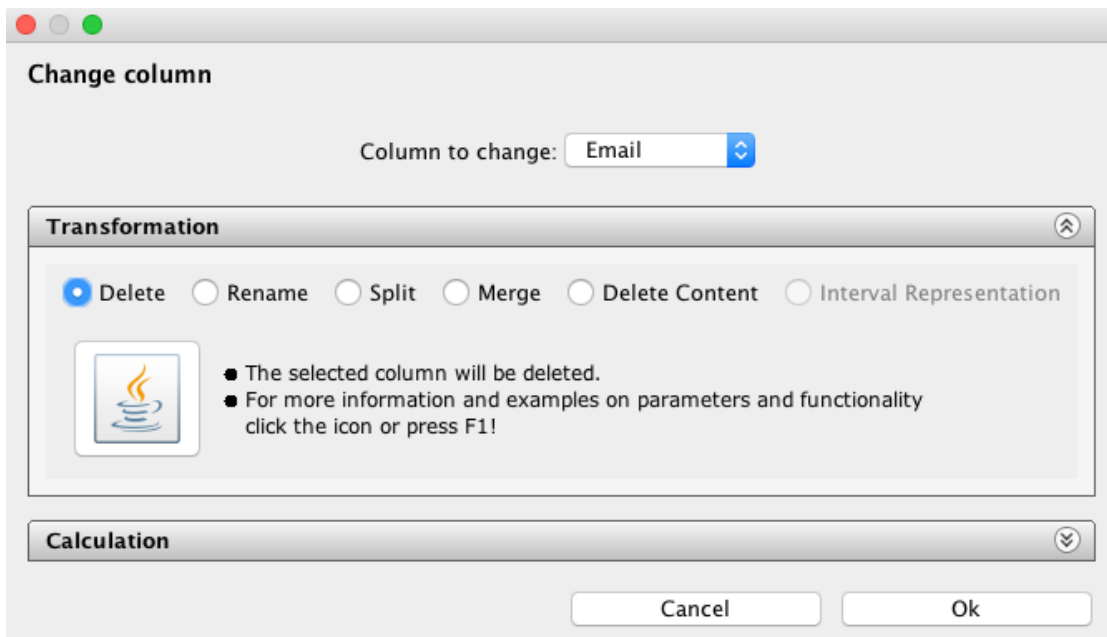


Figure A.12: *Change Column - Delete* dialog.

Parameter	Explanation
Column to change	The name of the column to delete.

Table A.12: *Change Column - Delete* parameters.

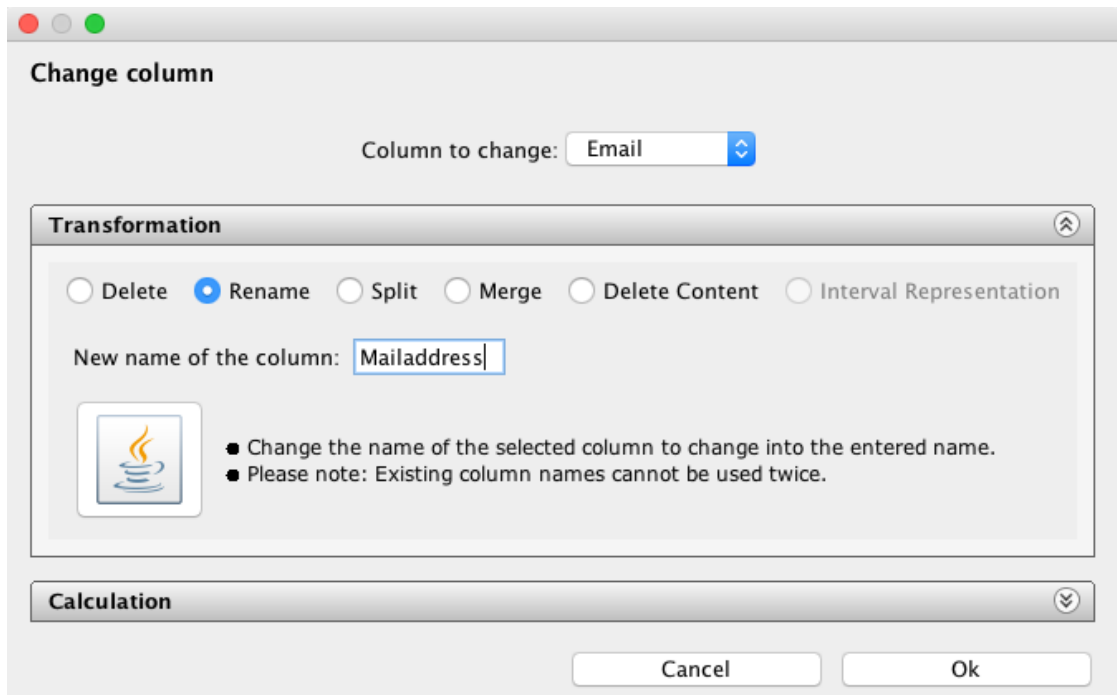


Figure A.13: *Change Column - Rename* dialog.

Parameter	Explanation
Column to change	The name of the column to rename.
New name of the column	The name the column should carry (must not exist).

Table A.13: *Change Column - Rename* parameters.

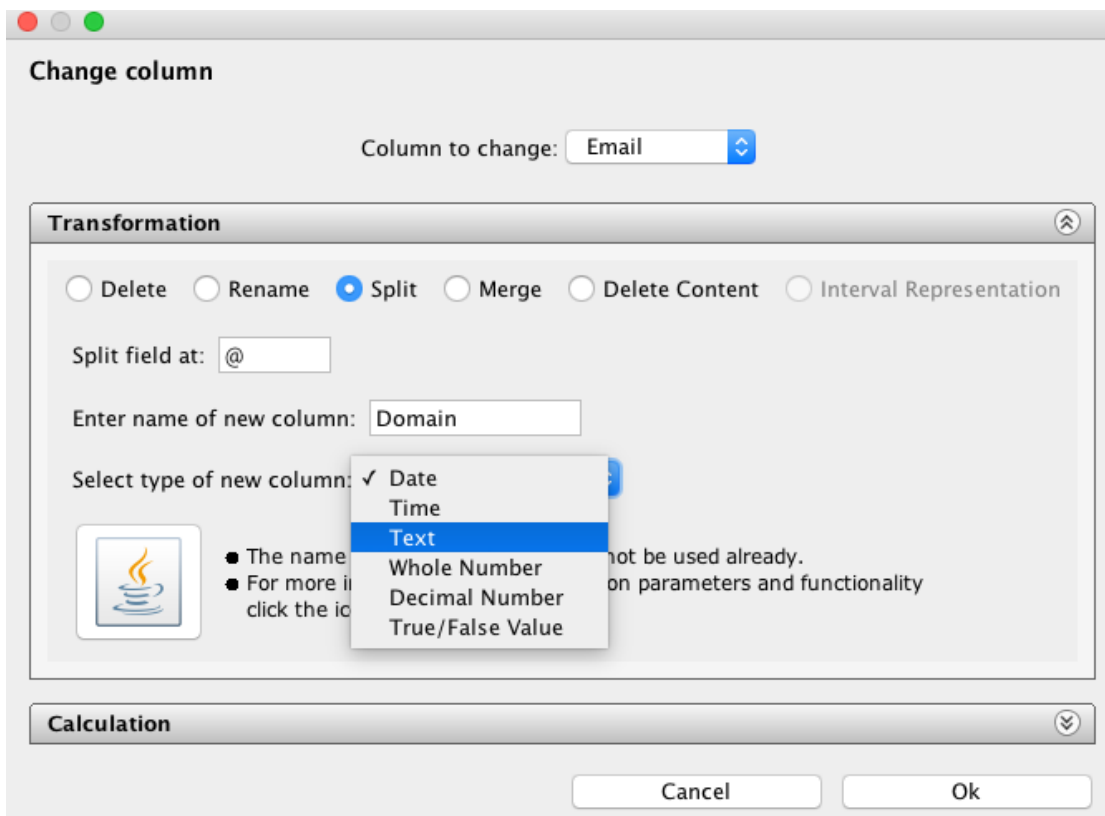


Figure A.14: *Change Column - Split* dialog.

Parameter	Explanation
Column to change	The name of the column to split.
Split field at	A character or a String that separates the old values from the new (as in the values of the new column) values.
Name of the new column	The name the column should carry (must not exist).
Select type of the new column	The type of the new column (must fit the split, i.e., cannot assign a text to a number column).

Table A.14: *Change Column - Split* parameters.

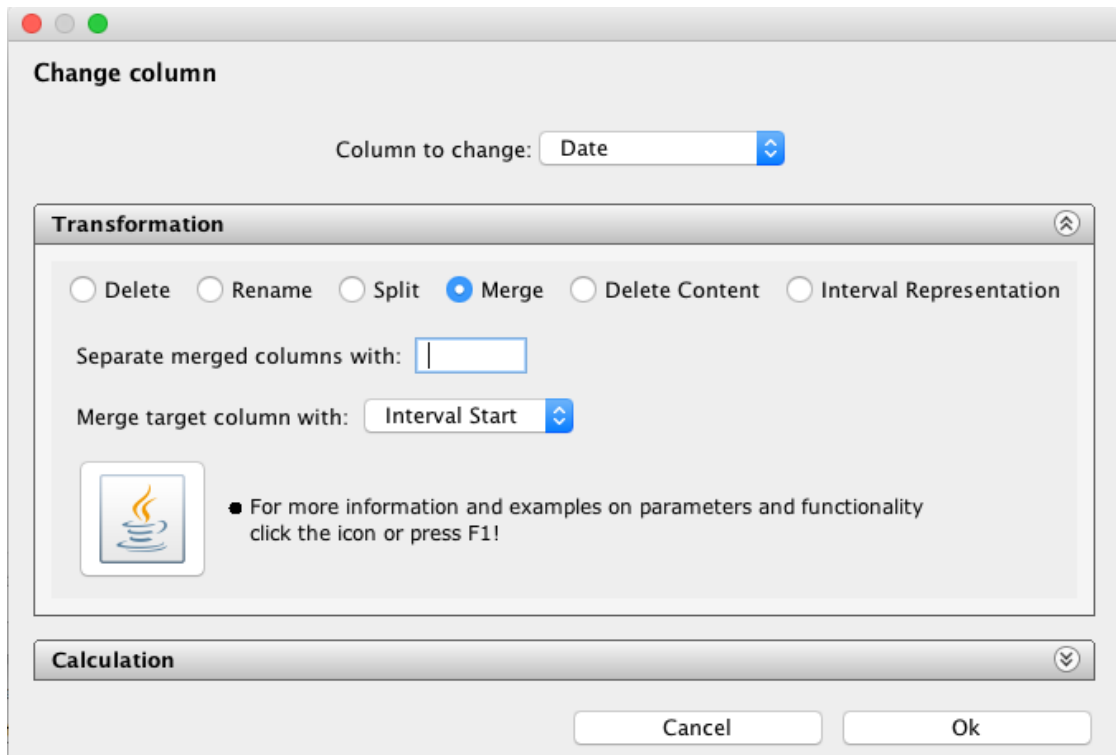


Figure A.15: *Change Column - Merge* dialog.

Parameter	Explanation
Column to change	The name of the column that is the target to the merge.
Separate merged columns with	A character or a String that separates the current values from the new (as in the values of the column to merge) values (when merging a date and a time column that sign will automatically be set as space character!).
Merge target column with	The name the column should be merged into the .

Table A.15: *Change Column - Merge* parameters.

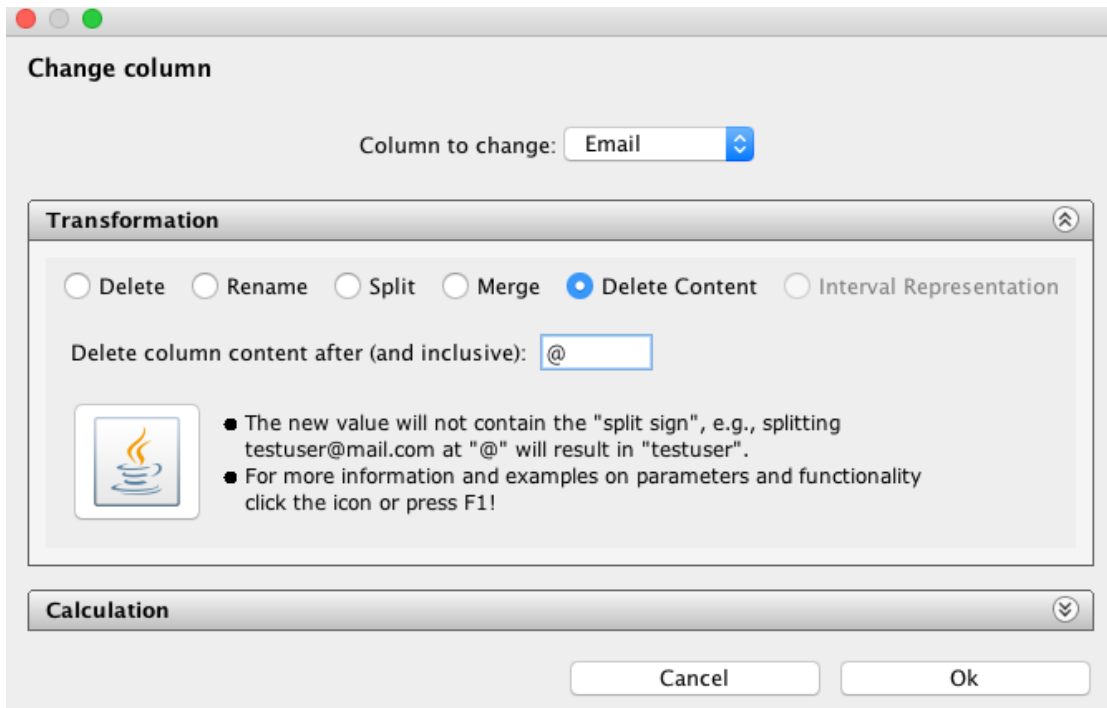


Figure A.16: *Change Column - Delete Content* dialog.

Parameter	Explanation
Column to change	The name of the column of which content should be deleted.
Delete column content after (and inclusive)	A character or String and all that follows will be removed.

Table A.16: *Change Column - Delete Content* parameters.

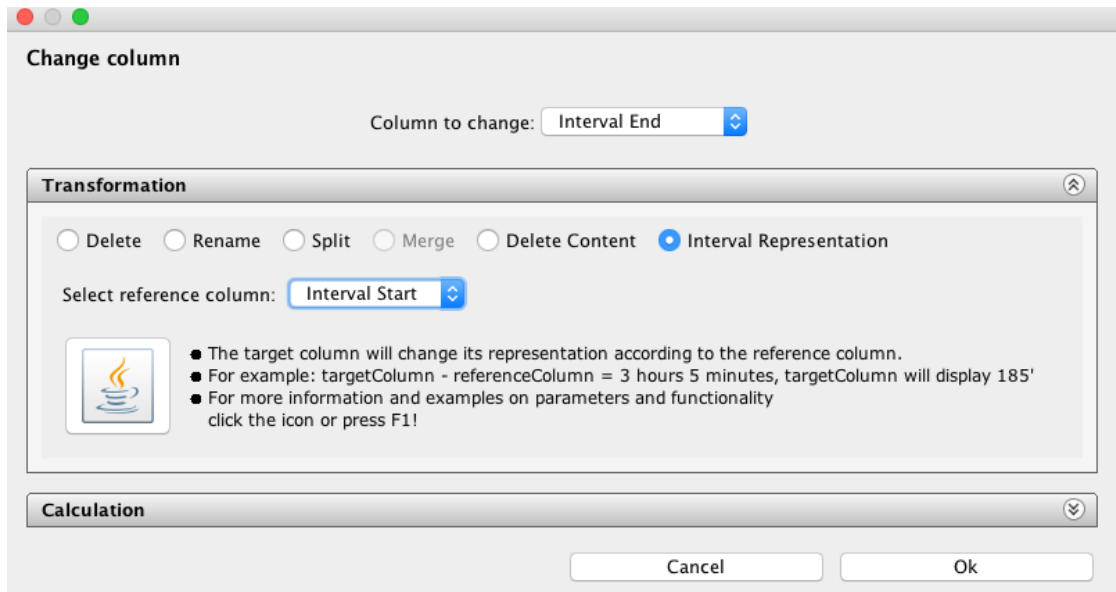


Figure A.17: *Change Column - Interval Representation* dialog.

Parameter	Explanation
Column to change	The name of the column to change its representation.
Reference column	The name of the column that defines the start point of the interval (e.g., Column 'Interval Start': <i>08:00</i> and Column 'Interval End': <i>10:00</i> will result in changing 'Interval End' to <i>120'</i>).

Table A.17: *Change Column - Interval Representation* parameters.

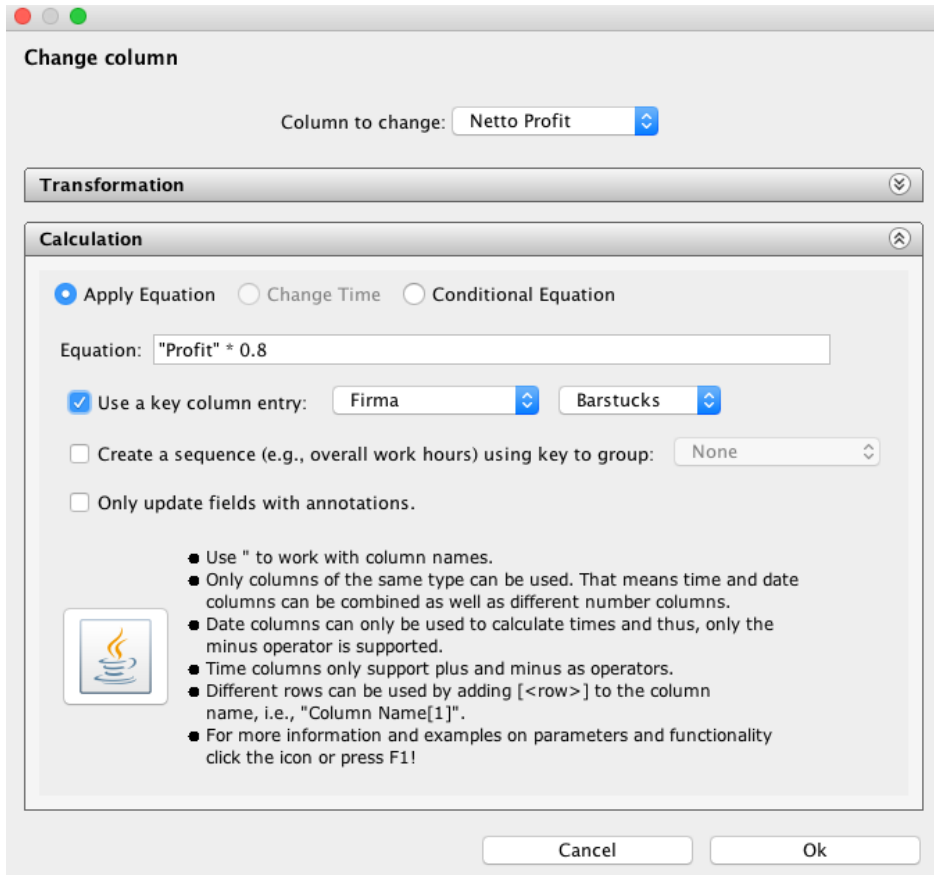


Figure A.18: *Change Column - Apply Equation* dialog.

Parameter	Explanation
Column to change	The name of the column that is the goal of the calculation.
Equation	The equation to be applied.
<i>Create a sequence</i>	Whether the result of the calculation should be cumulated (e.g., here, all net profits are added up).
<i>Using key to group</i>	For which key entries the accumulation should be created.
<i>Only update fields with annotations</i>	Whether only annotated fields should be updated with the result of the calculation.

Table A.18: *Change Column - Apply Equation* parameters.

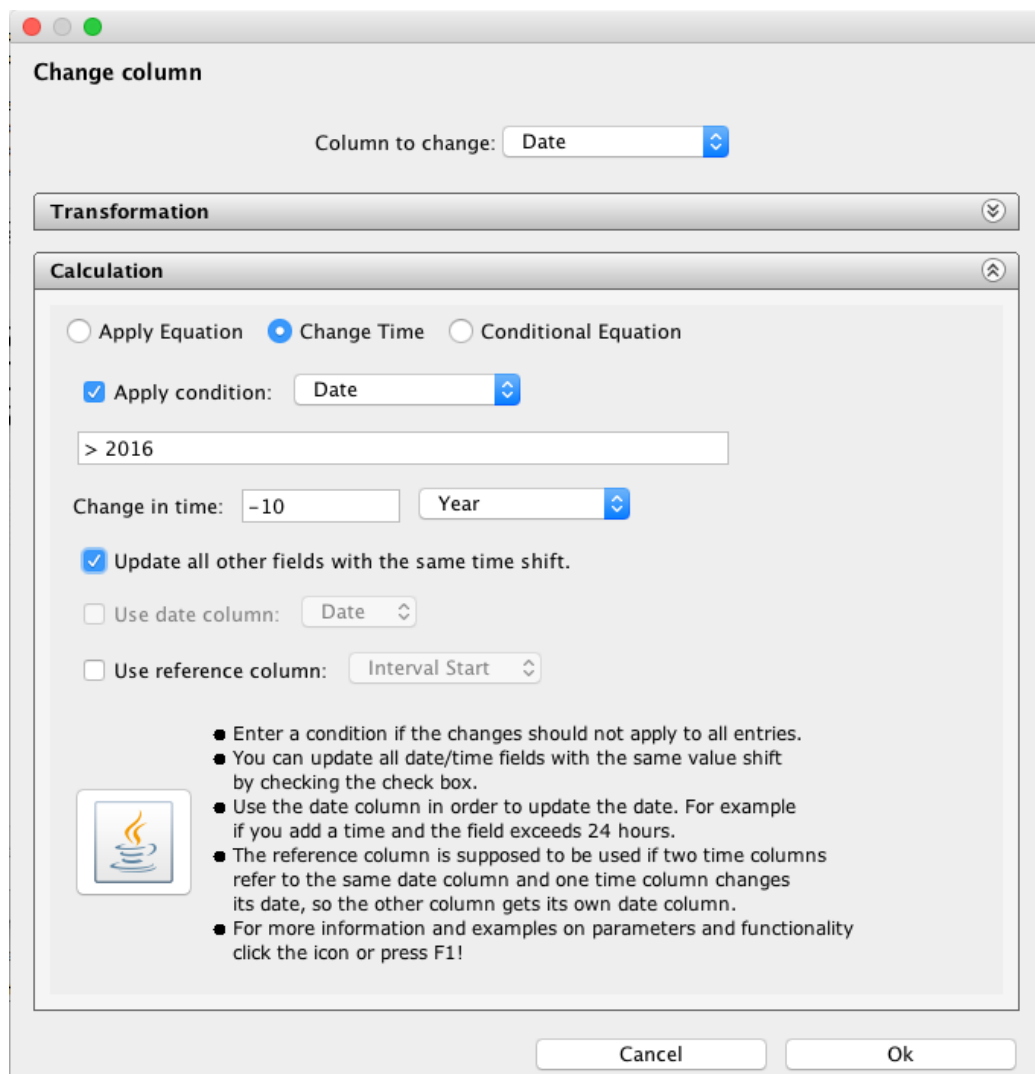


Figure A.19: *Change Column - Change Time* dialog.

Parameter	Explanation
Column to change	The name of the column that is the goal of the change in time.
<i>Apply condition Condition</i>	Whether an condition should be applied. The conditional input an entry must fulfill in order to be target of the changes.
Change in time	The number and the unit of the change in time (- can be used if time should be subtracted).
<i>Update all other fields with the same time shift</i>	Whether the change in time should be applied to all columns of the same type.

Parameter	Explanation
<i>Use date column</i>	Whether a date column is linked to the column to change. Otherwise changes that will result in a shift of date (e.g., adding 24 hours) will not be noted properly).
<i>Date column</i>	The name of the linked date column.
<i>Use reference column</i>	Whether another (time) column is linked to the column to change.
<i>Reference column</i>	The name of the linked (time) column (in case the date changes for one time column the second time column will be linked to a new date column so it becomes obvious that only one column was changed).

Table A.19: *Change Column - Change Time* parameters.

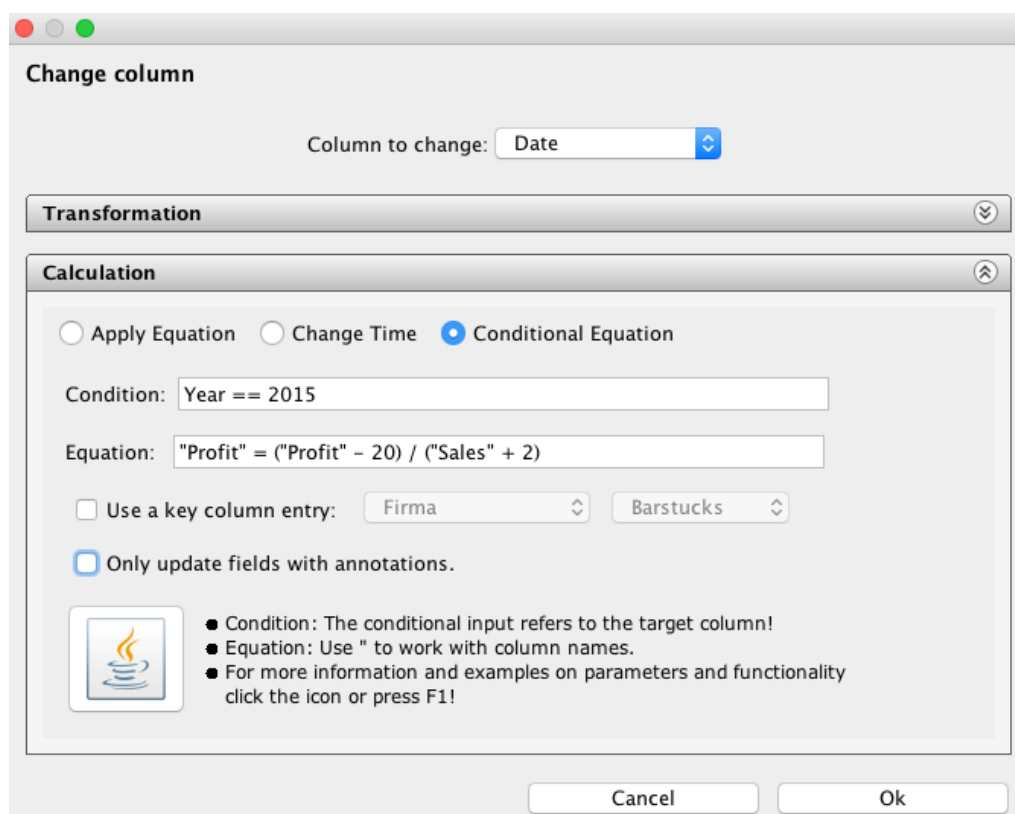


Figure A.20: *Change Column - Conditional Equation* dialog.

Parameter	Explanation
Column to change	The name of the column that is used for the condition.
Equation	The equation to be applied (including the column that is the goal of the equation).
<i>Only update fields with annotations</i>	Whether only annotated fields should be updated with the result of the calculation.

Table A.20: *Change Column - Conditional Equation* parameters.

A.1.5 Add Column

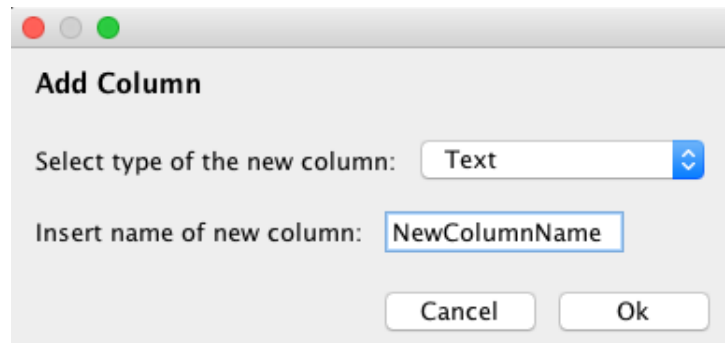


Figure A.21: Add column dialog.

Parameter	Explanation
Type of the new column	Whether the new column should hold date, time, text, integer, double or Boolean content.
Name of the new column	The name of the new column.

Table A.21: *Add Column* parameters.

A.1.6 Change Rows

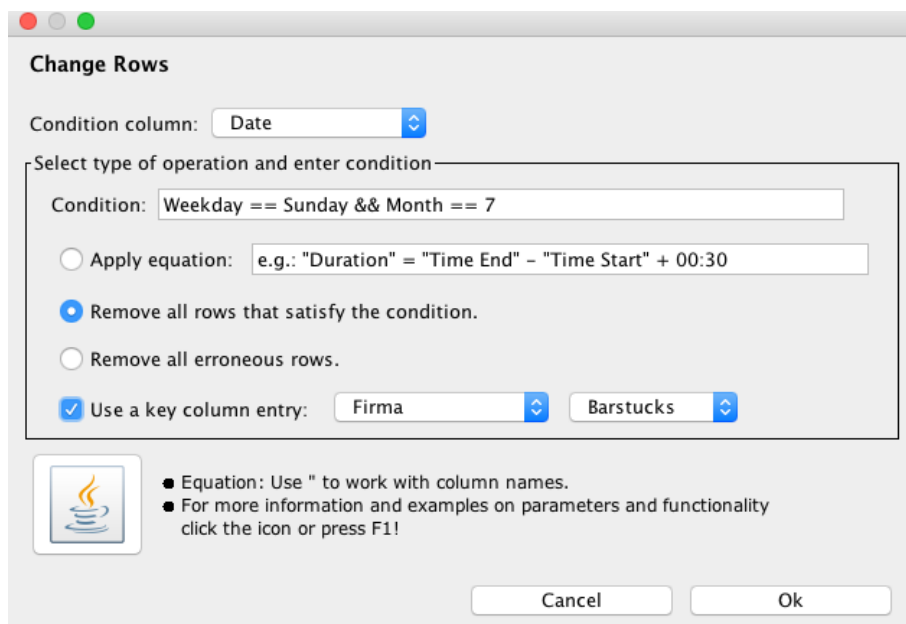


Figure A.22: Change rows dialog.

	Parameter	Explanation
	Condition column	The name of the column that is used for the condition.
Operation	Apply equation	Whether an equation should be applied to the relevant rows.
	Equation	The equation (including the target column).
	Remove all rows that satisfy the condition	All rows for which the condition is true are removed.
	Remove all erroneous rows	All rows that contain dirty data and match the condition are removed.

Table A.22: *Change Rows* parameters.

A.1.7 Sort Rows

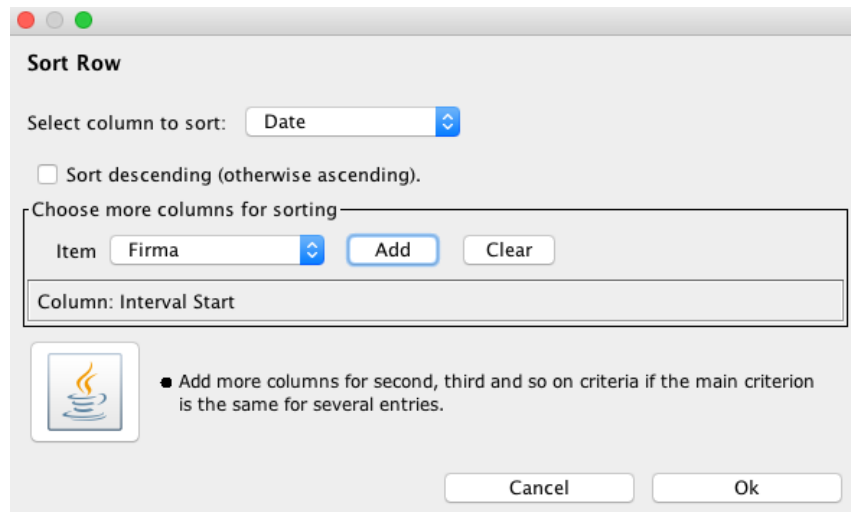


Figure A.23: Sort rows dialog.

Parameter	Explanation
Column to sort	The name of the column that is the main criterion for sorting.
Choose more columns for sorting	Add further names that will determine which row comes first if the main criterion is equal.

Table A.23: *Sort Rows* parameters.

A.1.8 Edit Intervals

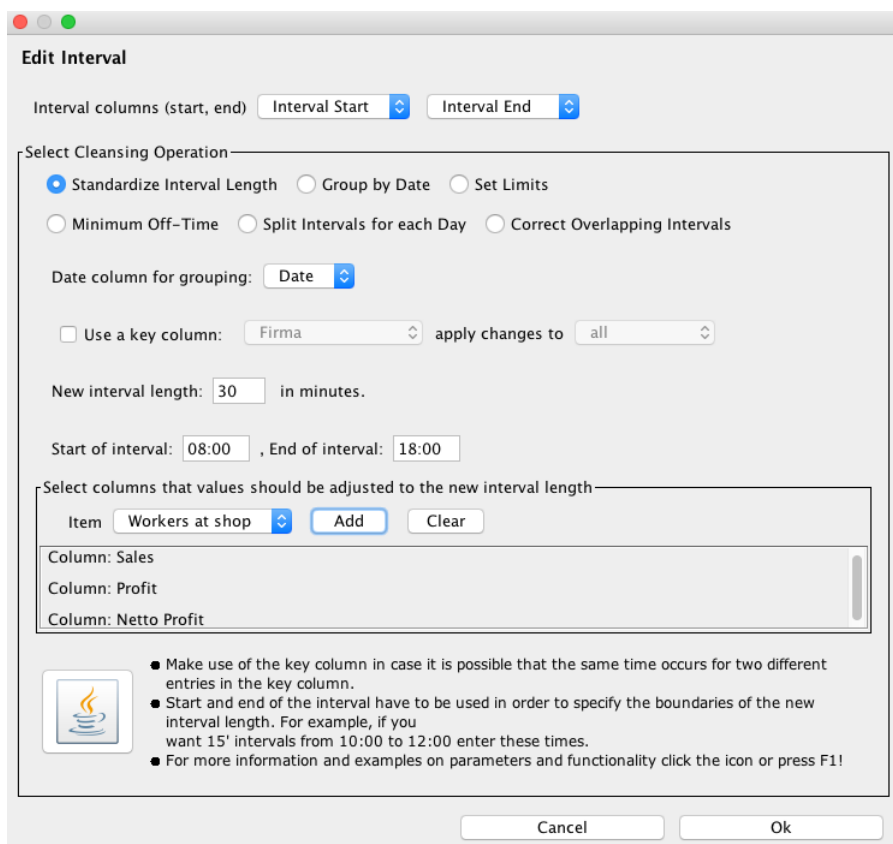


Figure A.24: *Edit Interval - Standardize Interval Length* dialog.

Parameter	Explanation
Interval columns (start, end)	The name of the start respectively the end interval column.
<i>Date column</i>	The name of the (key/main) date column.
Interval length (in minutes)	The length that intervals should have after the operation.
Start/End of interval	The range of intervals that should be transformed to the new length.
<i>Value adjustment</i>	Add all columns that should be adjusted to the new length (e.g., if the interval was one hour and the new length is two then the values of the selected column(s) will be doubled).

Table A.24: *Edit Interval - Standardize Interval Length* parameters.

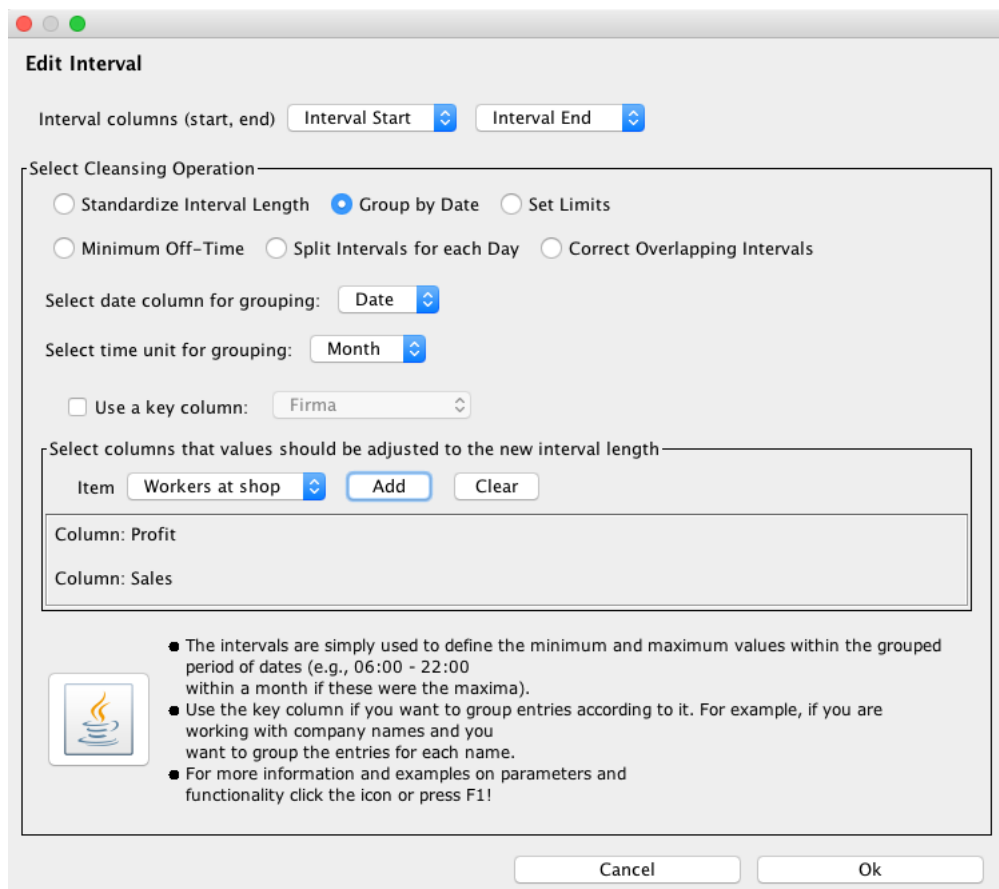


Figure A.25: *Edit Interval - Group by Date* dialog.

Parameter	Explanation
Interval columns (start, end)	The name of the start respectively the end interval column.
Date column	The name of the (key/main) date column.
Unit for grouping	Whether the entries should be grouped by day, month or year.
<i>Value adjustment</i>	Add all columns that should be adjusted to the new length (e.g., if the interval was one hour and the new length is two then the values of the selected column(s) will be doubled).

Table A.25: *Edit Interval - Group by Date* parameters.

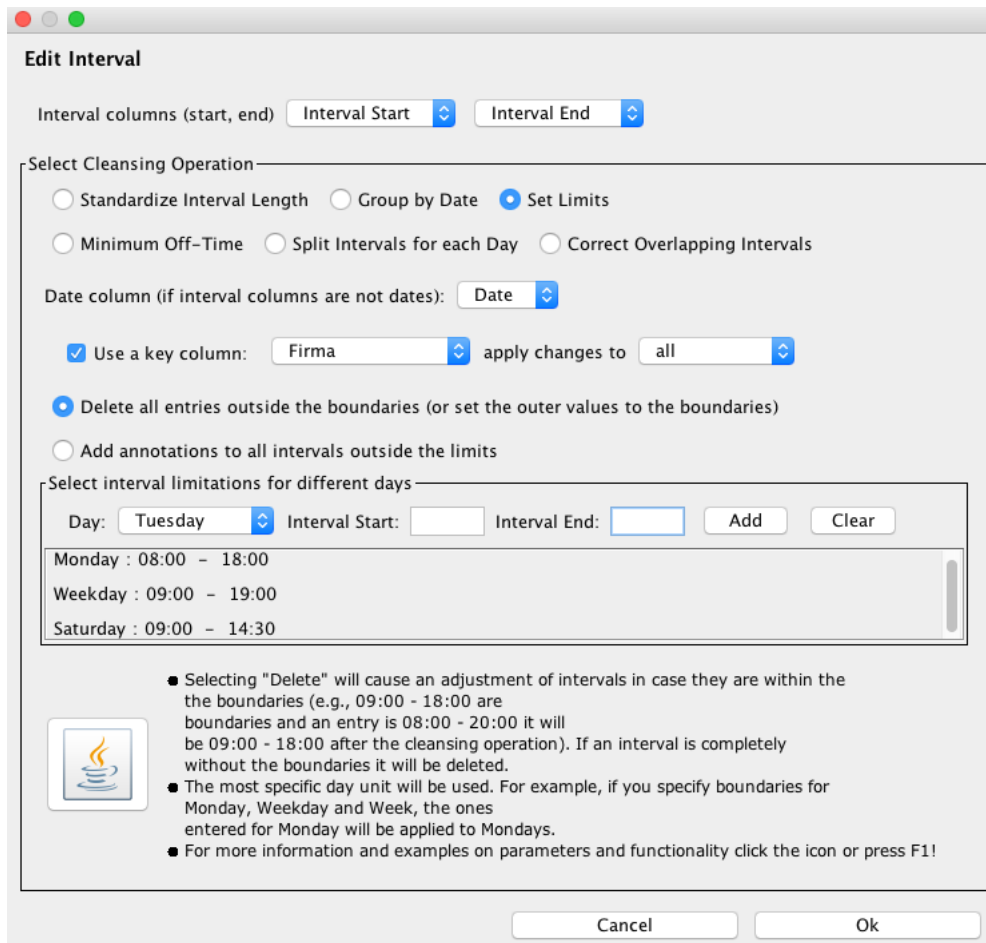


Figure A.26: *Edit Interval - Set Limits* dialog.

	Parameter	Explanation
	Interval columns (start, end)	The name of the start respectively the end interval column.
	Date column	The name of the (key/main) date column.
Method	Delete all entries outside the boundaries	Remove all entries that do not fit the limits.
	Add annotations to all intervals outside the limits	Annotates entries without deleting them.
Selection	Day	To which day (weekday/weekend/week) the limits should be applied.
	Interval start	The lower boundary of the limit for a certain day.
	Interval end	The upper boundary of the limit for a certain day.

Table A.26: *Edit Interval - Set Limits* parameters.

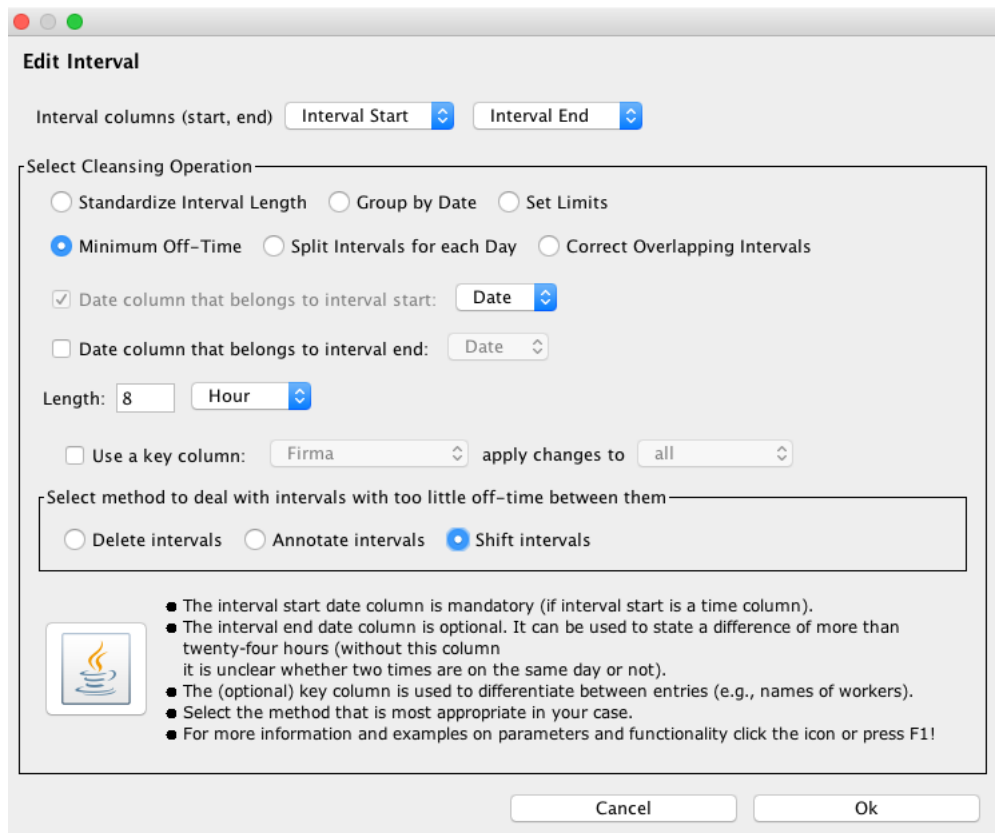


Figure A.27: *Edit Interval - Minimum Off-Time* dialog.

	Parameter	Explanation
	Interval columns (start, end)	The name of the start respectively the end interval column.
	Date column	The name of the (key/main) date column.
	<i>Date column (interval end)</i>	If the interval end column has a date column linked to it use the name of it here.
	Length & unit	How much time should pass between the end of an entry and the start of the next one.
Method	Delete intervals	All entries that violate the minimum-off time requirement are deleted.
	Annotate intervals	Violations are annotated, but no rows are deleted.
	Shift intervals	Violated entries are shifted by the exact amount of time, so they do not violate the requirement anymore.

Table A.27: *Edit Interval - Minimum Off-Time* parameters.

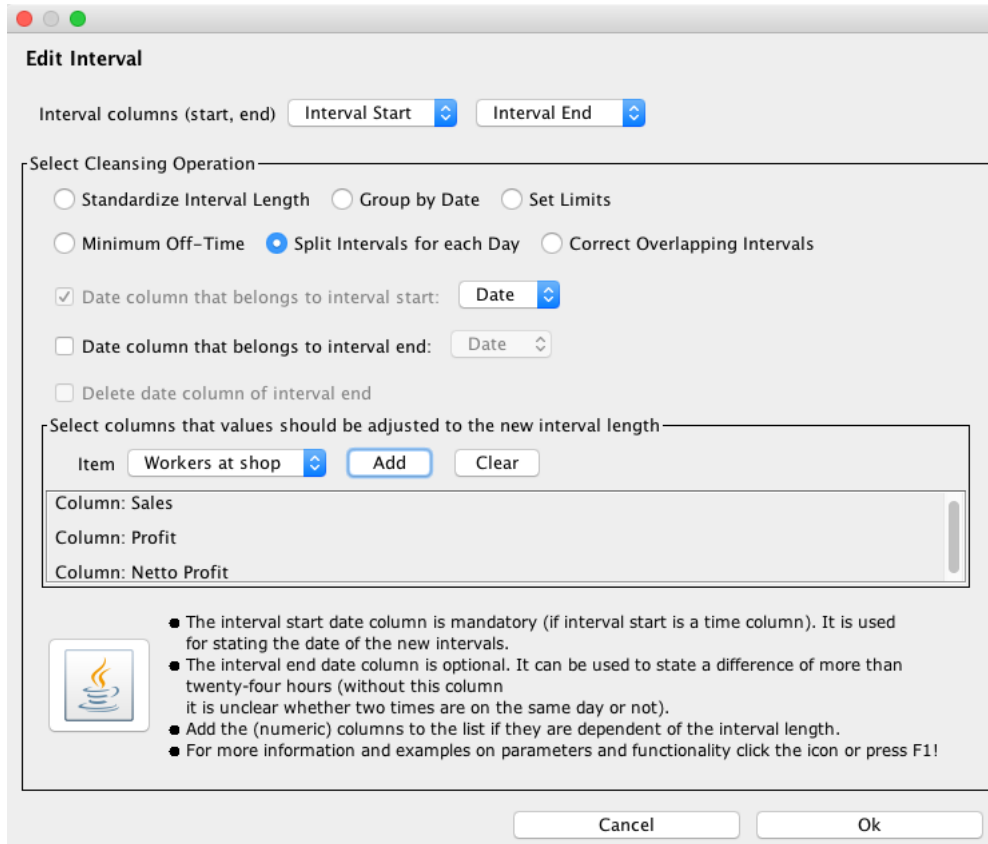


Figure A.28: *Edit Interval - Split Intervals for each Day* dialog.

Parameter	Explanation
Interval columns (start, end)	The name of the start respectively the end interval column.
Date column	The name of the (key/main) date column.
<i>Date column (interval end)</i>	If the interval end column has a date column linked to it use the name of it here.
<i>Delete date column (interval end)</i>	Deletes the date column that is linked to interval end, because after the split the date of interval start and interval end is the same.
<i>Value adjustment</i>	Add all columns that should be adjusted to the new length (e.g., if the interval was one hour and the new length is two then the values of the selected column(s) will be doubled).

Table A.28: *Edit Interval - Split Intervals for each Day* parameters.

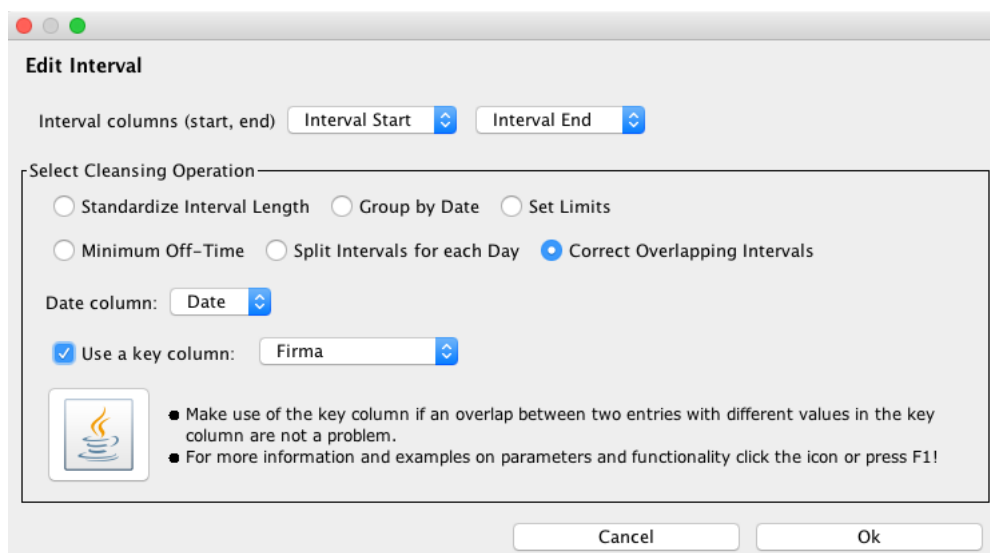


Figure A.29: *Edit Interval - Correct Overlapping Intervals* dialog.

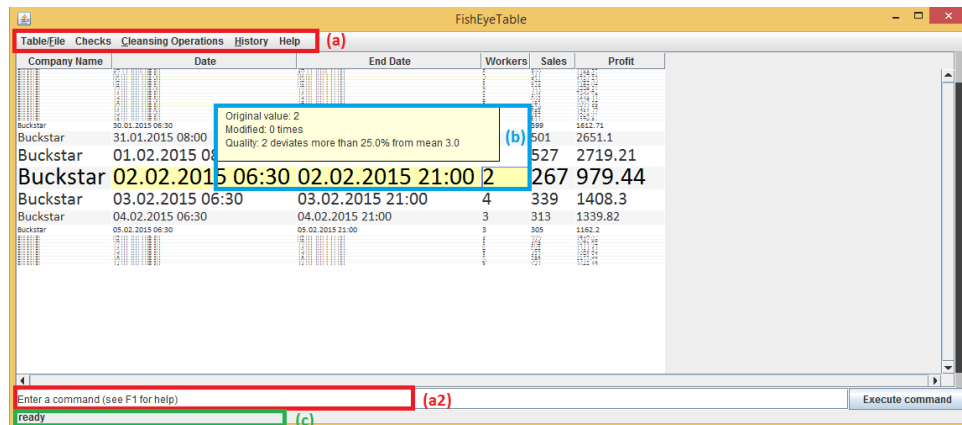
Parameter	Explanation
Interval columns (start, end)	The name of the start respectively the end interval column.
Date column	The name of the (key/main) date column.

Table A.29: *Edit Interval - Correct Overlapping Intervals* parameters.

A.2 User Study Material

QualityTime - Introduction

QualityTime is a tool that supports the recognition and cleansing of 'dirty' time-oriented data. The interface is structured in the following way:



It consists of (a) a menu that allows access to *Checks* and *Cleansing Operations*, (a2) which can also be accessed via the command line - however, only basic operations are supported this way, complicated operations with many input parameters must be executed using the menu. The data is presented in form of a fish-eye-table and shows (b) annotated, i.e., erroneous, entries and gives hints, which problems exist. At last, (c) a status bar visualizes what happened during the last check or cleansing operation (e.g., number of changed rows).

Further help regarding cleansing operations can be accessed by pressing F1 or navigating to *Help* via the menu. Additionally, each cleansing operation dialog will offer a short explanation of the most important input parameters - if any question should arise "F1" will provide more information.

In case a mistake happens (e.g., wrong parameterization leads to undesired results) the *History* can be used to undo (or redo) operations.

QualityTime - The Data

QualityTime reads .csv files and visualizes them in the table. The file (*Data.csv*) you will be working with is purely fictional and has no reasoning behind it. The data itself is supposed to present sales and profit figures of two coffee houses that were recorded from 29.12.2014 to 31.03.2015. Sometimes programs used to record data produce faulty values, moreover, user input can be wrong (typos) and thus, several data problems can occur and will be introduced step by step of the tasks that need to be completed during the study.

QualityTime - User Study

Task 1 - Two or more days and one entry

Sometimes the software did not correctly recognize the closing of the shop and did not create a new entry for a new day (e.g., the shop closed on the 1. January 2015 and reopened at the 2. January 2015, but the data lists them as one entry ranging from 1. January 2015 to 2. January 2015). The first task is to remove such errors and list them as two separate entries. However, splitting one entry into two separate ones results in problems with parameters. Think about which parameters have to be adjusted when splitting them.

Company Name	Date	End Date	Workers	Sales	Profit
Buckstar	28.12.2014 06:30	29.12.2014 21:00	3	312	1107.88
Buckstar	30.12.2014 06:30	30.12.2014 21:00	3	300	1138.7
Buckstar	31.12.2014 06:30	31.12.2014 21:00	4	317	1299.41
Buckstar	01.01.2015 06:30	02.01.2015 21:00	8	722	2859.91
Buckstar	03.01.2015 08:00	03.01.2015 21:00	5	532	2935.33
Buckstar	04.01.2015 08:00	04.01.2015 21:00	5	612	3222.27
Buckstar	05.01.2015 06:30	05.01.2015 21:00	3	321	1131.91

Task 2 - Opening hours

Buckstar's systems did not correctly set the time after changes were applied on the 28. February 2015. The time now is one hour behind the actual time. Make sure both date columns, *Date* and *End Date* are correctly set.

Buckstar	26.02.2015 06:30	26.02.2015 21:00	5		
Buckstar	27.02.2015 06:30	27.02.2015 21:00	3		
Buckstar	28.02.2015 08:00	28.02.2015 21:00	7		
Buckstar	01.03.2015 09:00	01.03.2015 22:00	8		
Buckstar	02.03.2015 07:30	02.03.2015 22:00	3		
Buckstar	03.03.2015 07:30	03.03.2015 22:00	3		
Buckstar	04.03.2015 07:30	04.03.2015 22:00	3		

Task 3 - Limit the interval to the regular opening hours

In Task 1 you were asked to split days. However, the days now do not contain the correct opening hours. Moreover, some opening hours are not correctly set for weekends.

The opening hours for *Buckstar* are:

Monday - Friday: 06:30 - 21:00

Saturday & Sunday: 08:00 - 21:00

Buckstar	31.12.2014 06:30	31.12.2014 21:00	4	317	1299.41
Buckstar	01.01.2015 06:30	01.01.2015 23:59	3	328	1299.41
Buckstar	02.01.2015 00:00	02.01.2015 21:00	4	393	1559.95
Buckstar	03.01.2015 08:00	03.01.2015 21:00	5	532	2935.33

Task 4 - Impute Missing Intervals

Unfortunately, some records were lost and could not be retrieved. Thus, it is necessary to impute missing intervals.

The opening hours for *MacD Cafe* are:

Monday - Sunday: 09:00 - 23:00

In order to save some work impute all missing values during the same step. Consider that the sales figures strongly depend on the weekday (more sales on weekdays, but more profit on weekends (because people take more time and order cake instead of only a coffee to go)).

MacD Cafe	23.03.2015 09:00	23.03.2015 23:00	2	173	526.5
MacD Cafe	24.03.2015 09:00	24.03.2015 23:00	2	196	599.61
MacD Cafe	26.03.2015 09:00	26.03.2015 23:00	2	178	512.08
MacD ...	27.03.2015 09:00	27.03.2015 23:00	2	202	616.89
MacD Cafe	28.03.2015 09:00	28.03.2015 23:00	3	135	677.26
MacD Cafe	29.03.2015 09:00	29.03.2015 23:00	3	136	682.72
MacD Cafe	31.03.2015 09:00	31.03.2015 23:00	2	202	604.35

Task 5 - Correct Implausible Values

Run "Checks - Find Outliers" to detect implausible values throughout the data set w.r.t. to Sales figures (parameters see Figure).

After detecting the problems use two different methods in order to resolve them:

- Apply Moving Average - **Triple Exponential Smoothing** on *Buckstar* (only).
 - Note: L stands for the number of entries during one cycle.
 - α : assigns a weight to the current (perhaps faulty) entry.
 - Allow deviations in the range of 20 to 25 percent.
 - The table needs to be sorted ascending by *Company Name* and *Date*.
- Apply Set Null - Impute Mean afterwards (please note: since Buckstar was cleansed of incorrect values only MacD Cafe will be affected by these changes, hence it is not necessary to specify MacD Cafe exclusive (however, specifying the key column is important regarding the mean calculation!)).

Check Outliers

Column to check: Sales

Select type of operation and enter condition

Annotate values that deviate from the calculated average by (in %): 20

Consider interval duration? (start, end) Date End Date

Use key column Company Name

Choose criterion that should be fulfilled when calculating the mean

Date column: Date

Criterion Add Clear

Use only entries of the same weekday for calculation

● Key column: Makes use of it if different entries are in the data set.
 ● Criteria: Add a time criterion if the mean is not equal for certain dates.

Cancel Ok

Task 6 - Advanced calculation: Calculate donations

For this task three separate steps are necessary. In case you are stuck with task 6b make sure to check out the provided help.

Task 6a - Add new column: *type: Double (Decimal), name: Donation sum* (hint: if you prefer entering commands textually do so)

Task 6b - Advanced calculation: each company donates 0.05€ per sale. The donations should be summed up, i.e., 1€ donations on the first day, 2€ donations on the second day should be displayed as 1€ on the first day, 3€ on the second, etc.

Task 7 - Change date format

Because both companies are located in the US, the company headquarters wants to read *Date* as 29-12-2014 for example (but keep the time as it is!).

Thank you for participating in the study!

Bibliography

- [1] Edgar Acuña and Caroline Rodriguez. The Treatment of Missing Values and Its Effect on Classifier Accuracy. In David Banks, Frederick R. McMorris, Phipps Arabie, and Wolfgang Gaul, editors, *Classification, Clustering, and Data Mining Applications*, Studies in Classification, Data Analysis, and Knowledge Organisation, pages 639–647. Springer Berlin Heidelberg, 2004.
- [2] Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, and Christian Tominski. *Visualization of Time-Oriented Data*. Human-Computer Interaction Series. Springer London, London, London, 2011.
- [3] James F. Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 23(2):123–154, July 1984.
- [4] José Barateiro and Helena Galhardas. A Survey of Data Quality Tools. *Datenbank-Spektrum*, 1:14:15–21, 2005.
- [5] Richard A. Becker and William S. Cleveland. Brushing Scatterplots. *Technometrics*, 29(2):pp. 127–142, 1987.
- [6] Jürgen Bernard, Tobias Ruppert, Oliver Goroll, Thorsten May, and Jörn Kohlhammer. Visual-Interactive Preprocessing of Time Series Data. In Andreas Kerren and Stefan Seipel, editors, *SIGRAD*, volume 81 of *Linköping Electronic Conference Proceedings*, pages 39–48. Linköping University Electronic Press, 2012.
- [7] Robert Goodell Brown. Exponential Smoothing for Predicting Demand. *Arthur D. Little inc.*, 1956.
- [8] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.
- [9] Silvio Cesare and Yang Xiang. Software birthmark similarity. In *Software Similarity and Classification*, SpringerBriefs in Computer Science, pages 63–70. Springer London, 2012.
- [10] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth. *CRISP-DM 1.0*. CRISP-DM Consortium, 1999.

- [11] James W. Cooper. *Java Design Patterns: A Tutorial*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.
- [12] DataCleaner. Datacleaner. <http://www.datacleaner.org>. Accessed: 2015-06-14.
- [13] João Lobato dos Santos Dias. Support for User Interaction in a Data Cleaning Process. Master's thesis, Universidade Técnica de Lisboa, Portugal, October 2012.
- [14] Wayne Eckerson. Data Quality and the Bottom Line: Achieving Business Success through a Commitment to High Quality Data. Technical report, The Data Warehousing Institute, 2002.
- [15] Jean-Daniel Fekete, Jarke J. Van Wijk, John T. Stasko, and Chris North. The Value of Information Visualization. *Information Visualization*, 4950(2):1–18, 2008.
- [16] Camilla Forsell and Jimmy Johansson. An heuristic set for evaluation in information visualization. In *Proceedings of the International Conference on Advanced Visual Interfaces, AVI '10*, pages 199–206, New York, NY, USA, 2010. ACM.
- [17] A. J. Fox. Outliers in Time Series. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(3):pp. 350–363, 1972.
- [18] Helena Galhardas. Data Cleaning and Transformation Using the AJAX Framework. In *Proceedings of the 2005 International Conference on Generative and Transformational Techniques in Software Engineering, GTTSE'05*, pages 327–343, Berlin, Heidelberg, 2006. Springer-Verlag.
- [19] Helena Galhardas, Daniela Florescu, Dennis Shasha, and Eric Simon. AJAX: An Extensible Data Cleaning Tool. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, SIGMOD '00*, pages 590–, New York, NY, USA, 2000. ACM.
- [20] Everette S. Gardner. Exponential Smoothing: The State of The Art. *Journal of Forecasting*, 4(1):1–28, 1985.
- [21] Everette S. Gardner. Exponential Smoothing: The State of The Art–Part II. *International Journal of Forecasting*, 22(4):637–666, 2006.
- [22] Theresia Gschwandtner, Wolfgang Aigner, Silvia Miksch, Johannes Gärtner, Simone Kriglstein, Margit Pohl, and Nik 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, i-KNOW '14*, pages 18:1–18:8, New York, NY, USA, 2014. ACM.
- [23] Theresia Gschwandtner, Johannes Gärtner, Wolfgang Aigner, and Silvia Miksch. A Taxonomy of Dirty Time-Oriented Data. In Gerald Quirchmayr, Josef Basl, Ilun You, Lida Xu, and Edgar Weippl, editors, *Multidisciplinary Research and Practice for Information Systems*, volume 7465 of *Lecture Notes in Computer Science*, pages 58–72. Springer Berlin Heidelberg, 2012.

- [24] G. K. Gupta. *Introduction to Data Mining With Case Studies*. Prentice-Hall Of India Learning Pvt. Limited, 2006.
- [25] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1), 2009.
- [26] Jiawei Han, Jian Pei, and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2011.
- [27] Joseph M. Hellerstein. Quantitative Data Cleaning for Large Databases. pages 1–42, 2008.
- [28] Mauricio A. Hernandez and Salvatore J. Stolfo. Real-World Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.
- [29] David Huynh and Stefano Mazzocchi. Open Refine. <http://openrefine.org>. Accessed: 2015-11-27.
- [30] David Huynh and Stefano Mazzocchi. Open Refine, Server Side Architecture. <https://github.com/OpenRefine/OpenRefine/wiki/Server-Side-Architecture>. Accessed: 2015-11-27.
- [31] Prajakta S. Kalekar. Time Series Forecasting Using Holt-Winters Exponential Smoothing. *Kanwal Rekhi School of Information Technology, Tech. Rep.*, 4329008:1–13, 2004.
- [32] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive Visual Specification of Data Transformation Scripts. *Human factors in computing systems. ACM*, pages 3363–3372, 2011.
- [33] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. Profiler : Integrated Statistical Analysis and Visualization for Data Quality Assessment. *Proceedings of Advanced Visual Interfaces, AVI*, pages 547–554, 2012.
- [34] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. Visual Analytics: Definition, Process, and Challenges. In Andreas Kerren, John T. Stasko, Jean-Daniel Fekete, and Chris North, editors, *Information Visualization*, volume 4950 of *Lecture Notes in Computer Science*, pages 154–175. Springer Berlin Heidelberg, 2008.
- [35] Daniel A. Keim, Florian Mansmann, Jörn Schneidewind, Jim Thomas, and Hartmut Ziegler. Visual Data Mining. chapter Visual Analytics: Scope and Challenges, pages 76–90. Springer-Verlag, Berlin, Heidelberg, 2008.
- [36] Won Kim, Byoung Ju Choi, Eui Kyeong Hong, Soo Kyung Kim, and Doheon Lee. A Taxonomy of Dirty Data. *Data Mining and Knowledge Discovery*, 7(1):81–99, 2003.

- [37] Data Ladder. Data match 2013. <http://dataladder.com/data-matching-software/>. Accessed: 2015-06-15.
- [38] Jonathan I. Maletic and Andrian Marcus. Data Cleansing: Beyond Integrity Analysis. In *Proceedings of the Conference on Information Quality*, pages 200–209, 2000.
- [39] Chris Mayfield, Jennifer Neville, and Sunil Prabhakar. ERACER: A Database Approach for Statistical Inference and Data Cleaning. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 75–86, New York, NY, USA, 2010. ACM.
- [40] Microsoft. Excel. <http://office.microsoft.com/en-us/excel/>. Accessed: 2015-06-03.
- [41] Heiko Müller and Johann-Christoph Freytag. Problems, Methods, and Challenges in Comprehensive Data Cleansing. *Challenges*, (HUB-IB-164):1–23, 2003.
- [42] Jennifer Neville and David Jensen. Relational Dependency Networks. *J. Mach. Learn. Res.*, 8:653–692, May 2007.
- [43] Jacob Nielsen. 10 Usability Heuristics for User Interface Design. <https://www.nngroup.com/articles/ten-usability-heuristics/>, 1995. Accessed: 2016-04-06.
- [44] Andrew Odewahn. *Beautiful Visualization*, chapter Visualizing the US Senate Social Graph (1991-2009), pages 123–142. O'Reilly Media, 2010.
- [45] Paulo Oliveira, Fátima Henriques, and Pedro Henriques. A Formal Definition of Data Quality Problems. *2005 International Conference on Information Quality*, pages 1–14, 2005.
- [46] Jack Prins. Introduction to Time Series Analysis. <http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc4.htm>. Accessed: 2015-05-20.
- [47] Erhard Rahm and Hong Hai Do. Data Cleaning: Problems and Current Approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.
- [48] Vijayshankar Raman and Joseph M. Hellerstein. Potter's Wheel: An Interactive Framework for Data Cleaning and Transformation. pages 1–23, 2000.
- [49] Vijayshankar Raman and Joseph M. Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 381–390, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
- [50] Aistis Raudys, Vaidotas Lenčiauskas, and Edmundas Malčius. Moving Averages for Financial Data Smoothing. In Tomas Skersys, Rimantas Butleris, and Rita Butkiene, editors, *Information and Software Technologies*, volume 403 of *Communications in Computer and Information Science*, pages 34–45. Springer Berlin Heidelberg, 2013.

- [51] Thomas C. Redman. The Impact of Poor Data Quality on the Typical Enterprise. *Communications of the ACM*, 41(2):79–82, February 1998.
- [52] Kai-Uwe Sattler, Stefan Conrad, and Gunter Saake. Adding Conflict Resolution Features to a Query Language for Database Federations. *Australasian Journal of Information Systems*, 8(1), 2000.
- [53] Kai-Uwe Sattler and Eike Schallehn. A Data Preparation Framework Based on a Multidatabase Language. In *Database Engineering and Applications, 2001 International Symposium on.*, pages 219–228, July 2001.
- [54] Saed Sayad. K-Nearest Neighbors. [http://chem-eng.utoronto.ca/ data mining.](http://chem-eng.utoronto.ca/data_mining/) Accessed: 2015-11-28.
- [55] Ben Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, VL '96, pages 336–, Washington, DC, USA, 1996. IEEE Computer Society.
- [56] Jinyong Wang, Zhibo Wu, Yanjun Shu, Zhan Zhang, and Lixing Xue. A Study on Software Reliability Prediction Based on Triple Exponential Smoothing Method (WIP). In *Proceedings of the 2014 Summer Simulation Multiconference*, SummerSim '14, pages 61:1–61:9, San Diego, CA, USA, 2014. Society for Computer Simulation International.
- [57] Peter R. Winters. Forecasting Sales by Exponential Weighted Moving Averages. *Management Science*, 6(3):324–342, April 1960.