SN Computer Science (2023) 4:425
https://doi.org/10.1007/s42979-023-01828-8

SURVEY ARTICLE

Data Preparation: A Technological Perspective and Review

Alvaro A. A. Fernandes[1] · Martin Koehler[1] · Nikolaos Konstantinou[1] · Pavel Pankin[1] · Norman W. Paton[1] · Rizos Sakellariou[1]

Received: 16 May 2022 / Accepted: 10 April 2023 / Published online: 2 June 2023

Abstract
Data analysis often uses data sets that were collected for different purposes. Indeed, new insights are often obtained by combining data sets that were produced independently of each other, for example by combining data from outside an organization with internal data resources. As a result, there is a need to discover, clean, integrate and restructure data into a form that is suitable for an intended analysis. Data preparation, also known as data wrangling, is the process by which data are transformed from its existing representation into a form that is suitable for analysis. In this paper, we review the state-of-the-art in data preparation, by: (i) describing functionalities that are central to data preparation pipelines, specifically profiling, matching, mapping, format transformation and data repair; and (ii) presenting how these capabilities surface in different approaches to data preparation, that involve programming, writing workflows, interacting with individual data sets as tables, and automating aspects of the process. These functionalities and approaches are illustrated with reference to a running example that combines open government data with web extracted real estate data.

Keywords Data preparation · Data engineering · Data wrangling · Data analysis

Introduction

Data preparation, the multi-faceted process by which the data required by an application are identified, extracted, cleaned and integrated, is often cumbersome and labor intensive [33, 50]. Indeed, surveys show that data scientists may spend up to 80% of their time on the process of extracting, collating and cleaning data that is a precursor to its use for analysis.[1]

Data preparation has been used in different settings for decades, for example for populating enterprise data warehouses, but the emergence of big data [27], and the movement towards data democratization [60], has encouraged the emergence of self-service data preparation tools that support exploratory data analysis [45]. In the context of self-service approaches, and sometimes more widely, data preparation is often referred to as *data wrangling*.

Data preparation is a prominent challenge facing data scientists and engineers. This is reflected in a large and growing market for commercial offerings. For example, Grand View Research predict that the data preparation tools market will be $8.47 Billion by 2025, growing at a Compound Annual Growth Rate (CAGR) of 25.1%.[2]

✉ Norman W. Paton
npaton@manchester.ac.uk

Alvaro A. A. Fernandes
fernandesaaa@gmail.com

Martin Koehler
koehler.martin@gmail.com

Nikolaos Konstantinou
nikolaos.konstantinou@manchester.ac.uk

Pavel Pankin
pankinpd@gmail.com

Rizos Sakellariou
rizos@manchester.ac.uk

[1] Department of Computer Science, University of Manchester, Manchester M13 9PL, UK

[1] https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/?sh=7187f386f637.

[2] https://www.grandviewresearch.com/press-release/global-data-preparation-tools-market.

Despite the significance of this area, we know of no previous review article that has attempted to characterize the main approaches to data preparation, explain the differences between the approaches, and illustrate how solutions are developed by different types of product. Surveys have been produced that address individual approaches (such as Extract–Transform–Load systems [3, 79] and visual interfaces for data transformation [50]) and individual steps within data preparation (such as entity resolution [26, 36] and data repair [30, 47]); in addition, there is a review of the functionalities offered by commercial data preparation systems [41]. Here, we seek to complement these earlier reviews by considering a variety of approaches that support comprehensive data preparation processes. Specifically, we introduce common steps within the data preparation process, and explore how they surface in different types of data preparation systems.

Data preparation has been carried out in different settings and with different emphases over a considerable period. For example, data preparation has been prominent in the creation of data warehouses for decades [79], but an emerging emphasis on more opportunistic analyses of diverse data sets is leading to the development of toolsets that can be deployed more widely within organizations [45]. In this paper, we consider approaches to data preparation as fitting into the following categories:

- *Program based*: In such approaches, data preparation is considered to be a software development task, and the data scientist develops applications using libraries to support different functionalities (e.g., [52, 54]).
- *Workflow based*: In such systems, which include Informatica,[3] Talend [6], CloverDX [81] and Pentaho [16], the user uses a visual editor to write a workflow that brings together data preparation components, for example for accessing data sources, combining, cleaning or aggregating data sets, and for providing the results to databases or analysis tools.
- *Dataset based*: In such systems, which include Wrangler [51] and OpenRefine [80], the user principally interacts with a representation of a single data set, which is presented in a way that is similar to a table in a spreadsheet. Then, operations can be carried out on columns, e.g., by changing the formatting or splitting the values in one column over several.
- *Automation based*: In such systems, which include AutoPipeline [83], Tamr [75] and VADA [56], steps within the data preparation process are automated, typically building on some form of training data about the intended result of the preparation process. Automation may apply to individual steps within the data preparation process, or to complete wrangling pipelines.

Data preparation tools can also be supplied in platforms that combine preparation with other capabilities, such as data catalogs (e.g., as in Qlik[4]) or analytics tools (e.g., as in Tableau[5]). In terms of scope, in this paper we focus specifically on data preparation functionalities, and in terms of a series of steps, we start after data extraction [31], and stop before techniques that target specific types of analysis [7, 43]. These are technical areas in their own rights, with existing surveys.

To provide a consistent frame of reference, data wrangling steps and approaches are illustrated with reference to a running application relating to real estate data. Figure 1 contains some tables and sample tuples that are assumed to come from different data publishers, and that provide information about property sales and related open government data. The data preparation problem is to populate the table called *Target*, using data from the other tables. The three *Agency* tables contain data about properties that are for sale; each contains the properties that are available from a single real estate agency. The other tables, *DeprStats* (for *Deprivation Statistics*), *Postcodes* and *Schools* hold additional information that can be used to supplement the data in the estate agency sites.

The data sources manifest various inconsistencies that need to be taken into account during data preparation. For example, there is some missing data, the names of columns representing the same data are not necessarily the same, the values within the *street* columns have different levels of detail in the different agency sources, etc.

In seeking to clarify the features of each of these approaches, in Section "Data Preparation Functionalities" we outline the individual steps within a data preparation process, focusing in particular on data profiling, matching, mapping, format transformation and data repair. Thereafter, based on implementations of the running example using representative approaches to each of the categories listed above, in Section "Data Preparation Approaches", we describe how the different approaches support the different functionalities. In the light of this, conclusions are drawn in Section. "Conclusions".

[3] http://www.informatica.com.

[4] https://www.qlik.com/us/.

[5] https://www.tableau.com/.

Agency1

street	city	postcode	price	agency	contact
Whitfield Street	Greater London	W1T 5EF	137,495	A1 Branch1	020 898000
Biscayne Avenue	London	E1W 1AD		A1 Branch2	020 889000
Whitfield Street	London	W1T 5EF	137,495	A1 Branch1	020 898000
Whitfield Streeet	London	W1T 5EF	137,954	A1 Branch1	020 898000

Agency2

street	city	postcode	price	agency_PC	agency
9 Canton Street	London	E14 6JW	595,000	E14 3NE	A2 Branch1
2 Canton Street	London	E14 6JW	595,000	E14 3NE	A2 Branch1
20 South Drive		W1A 4AA		E14 6JW	A2 Branch1
44 Whitfield Street	London	W1T 5EF	137,000	E14 6JW	A2 Branch1
10 Canton Street	London	E14 6JW	590,000	E14 3NE	A2 Branch2
9 Canton Street	Greater London	E14 6JW	595,000	E14 3NE	A2 Branch1
44 Whitfield Street	Greater London	W1T 5EF	137,000	E14 6JW	A2 Branch1

Agency3

street	postcode	city	price	type	contact
Aytoun Street	M1 3DA	Manchester	1,200	Apartment	0161 654321
Brazil Street	M1 3DG	Manchester	1,550	Apartment	0161 654321

DeprStats

postcode	status	crimerank	crimedecile
M1 3DA	live	13187	5
M1 3DG	live	8079	3

Postcodes

postcode	county	ward
E1W 1AD	Greater London	St Katharines
W1T 5EF	Greater London	Bloomsbury

Schools

school	gender	ward	effectiveness
Oasis A Nunsthorpe	Girls	St Katharines	4
CATS College	Boys	Bloomsbury	3

Target

street	city	postcode	price	agency	crime	ward
Whitfield Street	London	W1 5EF	137,495	A1 Branch1	5	Bloomsbury
Biscayne Ave	London	E14 1AD	189,950	A1 Branch2	6	St Katharines
Canton Street	London	E14 6JW	595,000	A2 Branch1	7	Limehouse Ward
South Drive	London	W1A 0AA	575,000	A2 Branch2	6	West End

Fig. 1 Running example from the real estate domain

Data Preparation Functionalities

Data preparation involves a series of processing steps, that may start with some form of data set discovery, proceed through a number of integration and transformation tasks, and complete with the storage of the result. There is no universally accepted collection of data preparation steps, but certain functionalities are widely supported, and can be considered as representative of data preparation. Here, we focus on several processing and transformation tasks, in particular *data profiling*, *schema matching*, *schema mapping*, *format transformation* and *data repair*. Data preparation systems differ in exactly which tasks they support, but there tend to be significant overlaps in the functionalities provided, and thus different categories of system are more easily distinguished by *how* such tasks are specified and composed than by precisely *what* capabilities are provided.

The specific steps covered in this paper may be preceded by steps that, for example, discover relevant data sets through search mechanisms [18] or extract structured data from file formats or spreadsheets [15]. Furthermore, there may be application-specific post-processing steps, for example that seek to select or structure the data in ways that are particularly suitable for machine learning [72].

In this section, we introduce representative processing and transformation tasks with reference to the running example. We will primarily use relational database terminology, for example referring to tables, tuples, keys and attributes, even though actual data preparation systems may assume different data models. Furthermore, we do not consider domain-specific data preparation, for example for spatial data [14] or medical imaging [22], for which specialized techniques are required to complement those described here.

Data Profiling

Data preparation builds on knowledge of the data to be prepared. As data preparation may bring together data from diverse data sources, the amount and nature of the descriptive information that comes with sources varies. For example, among others, data preparation may be applied to comma separated value (csv) files with or without headers, to semi-structured data sets described using JSON or XML, or to relational databases. Thus, there may be different amounts of information provided with a data source about the names of attributes, their types, and the relationships between data items. As a result, it is often useful to refine structural descriptions of individual sources, and also to derive additional information that can inform different data preparation steps.

Data Profiling is the name given to the process of deriving properties of data sets and the relationships between data sets, that can inform processing decisions, whether these are manual or automated [1]. It transpires that much the same profiling data is useful for all the different approaches to data preparation described in Section "Data Preparation Approaches".

In data preparation, different preparation tasks may be informed by different types of profiling data. For example, functional dependencies can be used to identify potential anomalies in a data set [47], and inclusion dependencies can be used to identify potential join paths between data sets [21]. There are many different data profiling tasks [1]. Here we provide some examples, illustrate them with respect to the real estate example, and indicate how they may be relevant to data preparation.

This first group of tasks provides information about a single data set. We omit the discussion of some more rudimentary profiling activities, such as counting the numbers of rows, the numbers of null values, and the number of distinct values within a column, but note that such information can be useful for informing decisions relating to activities such as the ordering of data preparation tasks or the selection of data sets.

Candidate key discovery: A candidate key is a column or collection of columns, the values of which can uniquely identify a tuple within a data set. A profiling algorithm may infer several candidate keys for a table. In the example in Fig. 1, both *postcode* and *crimerank* may be inferred as candidate keys for *DeprStats*, though in this case the *postcode* would be the better choice for a designer, as the *crimerank* is likely to change over time. Furthermore, *postcode* could perhaps be inferred to be a candidate key for *Agency1*, even though in practice there may be several properties with the same postcode. This reflects the fact that data profiling techniques derive information about the available data, and are thus subject to drawing conclusions that hold for a snapshot or a sample, but may not be correct in general. In data preparation, *candidate key discovery* is potentially useful for identifying ways of joining data sets [71].

Value distributions: An attribute contains a collection of values, the distribution of which may be useful for users seeking to understand the way the column is being used, or which may be useful for algorithms in making estimates such as the size of the result of a join. For example, a *histogram* [48] on an attribute provides a distribution D of the form $\{(r_1, c_1), ..., (r_n, c_n)\}$, where each r_i represents a range, and the corresponding c_i captures the number of values within the range. For example, in the *Schools* table in Fig. 1, if the values for *effectiveness* contained outliers, this might suggest that there were errors in the data. In data preparation, *value distributions* can be useful for source selection, and for obtaining insights into dataset quality.

Inclusion dependency discovery: An inclusion dependency is a relationship of the form $T_u.A_v \subset_{level} T_q.A_r$, where eah T_i is a table, each A_j is an attribute, and *level* is the fraction of the values in $T_u.A_v$ that are contained in $T_q.A_r$. Where the $level = 1$, there is a full inclusion dependency, and where the $level < 1$ there is a partial inclusion dependency. For example, in Fig. 1, *Agency*1.*postcode* might be expected to be fully included in *DeprStats.postcode*. In data preparation, full or partial inclusion dependencies can be used to identify candidate joins between tables, in particular where one of the participating attributes is a *candidate key* [71].

Matching

Data preparation depends on the ability to discover relationships between different data sets, and also to reconcile semantic heterogeneity [53] between otherwise compatible datasets. By *semantic heterogeneity* we mean differences in the way a conceptualization is concretely represented. For example, in Fig. 1, the concept of *deprivation* is represented in both the *DeprStats* and the *Target* tables, but with different names, viz., *crimedecile* in the former and *crime* in the latter. This is an example of a semantic heterogeneity referred to as *different names for equivalent attributes (DNEA)*. For another example, the different concepts of *for-sale price* and *for-rent price* are everywhere represented by an attribute named *price*. It is the *for-sale* price in *Agency1*, *Agency2* and in the target, and the *for-rent* price in *Agency3*. This is an instance of a semantic heterogeneity referred to as *same name for different attributes (SNDA)*.

Matching is the process whereby conceptual equivalence is postulated between a construct $S.x$ in one schema S and a construct $S'.y$ in another schema S' [8, 68]. These are often referred to as *correspondences*, or *matches*, and we say that $S.x$ and $S'.y$ match, or are a match. For example, in Fig. 1, the attributes with name *postcode* in various tables represent the

same concept, and matching would postulate that all pairs formed by them are matches (e.g., *Agency1.postcode* and *Agency2.postcode*, and so on). Likewise matching may also postulate that pairs that manifest the DNEA semantic heterogeneity (such as *DeprStats.crimedecile* and *Target.crime*) are matches.

Matching is a hard problem insofar as there is an inherent difficulty in detecting whether two schematic constructs are conceptually equivalent or not. The difficulty stems, on the one hand, from the problem of inferring equivalence from data and metadata alone, and, on the other hand, from the many forms of conflict that arise from representational choices [39, 53]. For example, pairs that define an instance of the SNDA semantic heterogeneity (e.g., *Agency3.price* and *Target.price*) may be difficult to detect automatically.

There are many approaches to matching and the taxonomy introduced in [68] is still valid. According to it, matching is carried out by algorithms (or systems) called matchers. Matching can be based on the results of a single matcher or on combining different matchers, either into a hybrid matcher that outputs one decision (e.g., [5]) or as a composite of individual matchers whose independent decisions are then combined, say, by some aggregation function, into a single output decision (e.g., [23]).

Individual matchers are the building blocks for hybrid and composite matchers. They can be classified according to the sources of information they use in reaching a decision. Matchers can make use of schema-level information only or of instance-level information too. Schema-level information can be syntactic (e.g., string similarity between attribute names) or semantic (e.g., type similarity). As for instance-level information, the majority of matchers use syntactic similarity over, say, attribute values as the basis upon which to decide that the extents (or a representative sample thereof) provide evidence of conceptual equivalence (e.g., a matcher might use the overlap of the range of values in *DeprStats.crimedecile* and in *Target.crime* in addition to the shared substring 'crime' in the attribute names as mutually reinforcing evidence of a DNEA conflict and, hence, postulate that the two attributes match).

Broadly speaking, a matcher (which could be a person or an algorithm) can be seen as performing, or comprising, the following major tasks:

Eliciting evidence of similarity: This typically involves the use of techniques for measuring string similarity (e.g., Levenshtein distance, q-grams, etc.) to yield a similarity score. So, in Fig. 1, on the basis of one or more string comparison algorithms, one would expect the pair *street* and *city* to have a very low similarity score; the pair *crimedecile* in *DeprStats* and *crime* in *Target* to have mid-scale similarity scores, and the pair *price* in *Agency3* and *price* in *Target* to have the maximal similarity score.

Combining evidence: This involves deciding how to combine, say, the fact that the pair *price* in *Agency3* and *price* in *Target* are identical attribute names with the fact that from instance-level evidence, they have non-overlapping extents with value intervals that are separated by two orders of magnitude. There are various weighting schemes and aggregation functions that can be used to combine individual quantifications of evidence into a single figure (see, e.g., [5, 34]).

Postulating equivalence: This involves projecting either individual scores or an aggregated one onto a decision axis such that, given one or more scores, the projection returns a decision as to whether a pair of constructs are, or are not, a match. The simplest way of defining such a decision axis is to use a threshold, such that a score above the threshold implies that the pair is a match, otherwise they are not. Finding a robust universal threshold is difficult in general and mitigation approaches exist (see, again, [5]).

Proposing semantic correspondences: Once it is decided which pairs are matches, a decision needs to be made as to which matches are used. For example, one construct in a source may match more than one construct in the target schema. An example would arise in Fig. 1 if *Agency2.agency* and *Agency2.agency_PC* were both postulated matches to *Target.agency*). Thus, the general case is not for there to be a 1:1 alignment but an n:m one, in which there is a question whether the matcher derives a 1:1 alignment from an n:m one (e.g., by somehow picking the strongest matches, possibly globally over all pairs rather than locally over the pairs in a particular source and a particular target table).

Mapping

In the context of data preparation (and, in particular, data integration [24]), a *mapping* is an expression that, when evaluated over source datasets, returns a result that is suitable for populating a target dataset. As such, a mapping allows existing, autonomously designed, populated and maintained datasets to be integrated into a target dataset that defines the structure required for the task at hand.

In the example in Fig. 1, the source datasets (i.e., *Agency1*, *Agency2*, and *Agency3*) were autonomously designed and populated. As has been pointed out, there are semantic heterogeneity conflicts between them, taken as potential sources, and the *Target* table. For example, while *Agency1* and *Agency2* are likely to record properties for sale, it is possible that *Agency3* records properties for rent. If so, and if the target is meant to be about properties for sale only, a suitable mapping might be centered on taking the union (of appropriate attributes from) *Agency1* and *Agency2* and excluding *Agency3*. Likewise, the mapping might include a join on the *postcode* columns of agency data and deprivation data in order to obtain the values with which to populate the *crime* column in the target.

Research on mappings has been significant and extends over a long period (for useful background, see [29, 40, 63]). The motivation for research on mappings has come from two important real-world problems, viz., data integration [24] and data exchange [4]. In *data integration*, mappings tend to be seen as views that define a virtual, integrated resource over a collection of autonomous, heterogeneous, typically distributed resources. These views may be subject to rewriting prior to evaluation [40], or may be generated taking into account properties of the sources and the target [29]. In *data exchange* [28], mappings tend to be seen as transformation scripts whose evaluation has the purpose of converting an instance I_S of a source schema S into an instance I_T of a target schema T, with minimal redundancy in I_T.

Broadly speaking, a mapper (again, a person or an algorithm) can be seen as performing, or comprising, the following major tasks:

Detecting merge opportunities: This involves detecting, given the semantic correspondences identified by matching and profiling data such as inclusion dependencies, whether, e.g., one might usefully union two source tables and join the result of the union with another source table. As such, both manual and automatic mapping generation tend to follow and be informed by both data profiling and matching. For example, in Fig. 1, assume that matching has postulated that the attributes with name *street*, *city*, *postcode*, *price* and *agency* in both the *Agency1* and *Agency2* source datasets match with (i.e., semantically correspond to) the identically named attributes in the *Target*. If so, the union of the projections of these attributes over *Agency1* and *Agency2*, call it U, seems a useful merge opportunity. If we further assume the attributes named *postcode* in U and *Postcodes* semantically correspond and that it is known that *postcode* is a candidate key in *Postcodes* and that $U.postcode \subseteq Postcodes.postcode$, then taking the natural join of U and *Postcodes* is another useful merge opportunity under the assumption that *Postcodes.ward* and *Target.ward* were deemed by matching to semantically correspond.

Constructing/selecting the mappings: This involves exploring the space of possible mappings given the detected merge opportunities to, broadly speaking, decide on the quality of each such mapping. Note that, often, merge opportunities come at the cost of introducing null values not already in the sources. In this light, one might wish to rank merge opportunities on whether they might lead to the introduction of more, or fewer, null values not already in the sources. Another goal, often paramount in data exchange, is to minimize the size of the materialized target instance. Different strategies can be used, e.g., preferentially following join opportunities (e.g., [29]), possibly also relying on a subsequent verification/optimization phase (e.g., [59]). Yet another important concern is that the mapping evaluates to instances that conform to key and referential constraints that may exist when the target schema comprises many tables.

Format Transformation

The role of format transformation is to resolve representational inconsistencies in attribute values. A *format transformation rule* is an expression in which the antecedent is a syntactic pattern to be matched over values in one or more source columns of an input dataset, such that the result of the rule is a different representation for the information in one or more columns of the output dataset.

In the example in Fig. 1, the values in the *Target.street* column do not contain the number of the property (as in the case with those in *Agency2.street*). If this formatting is a requirement, when data from *Agency2.street* is used, through a mapping, to populate *Target.street*, it will be seen to violate it (unlike data coming from *Agency1.street*). To meet the requirement, one could apply the following (admittedly simplistic) format transformation rule to the column of interest (viz., *Target.street*)

Target.street: `"[0-9]+"` → `""`

where the antecedent consists of a regular expression (viz., `"[0-9]+"`) that matches integers, and the consequent consists of a replacement string (viz. `""`). In this example, therefore, the effect is to remove any number appearing in prefix position in a *street* value in the target, so that `20 South Drive` becomes `South Drive`.

Broadly speaking, a format transformer (still, a person or an algorithm) can be seen as performing, or comprising, the following major tasks:

Deciding on the correct format: This involves either choosing (among those occurring) or imposing (in case it does not occur) a format for the data that is deemed correct for the task in hand. For example, in Fig. 1, as we have seen, there seems to have been a decision that the correct format for *Target.street* is one in which a prefixed number is absent.

Detecting format inconsistencies: This involves the recognition of patterns that do not match the correct one. For example, one in which the street is prefixed by a number.

Defining the required format transformation: This involves the provision of a format transformation rule that given an incorrectly formatted value transforms it into a correctly formatted one. The expressiveness of the language in which format transformation rules are formulated may vary (see, e.g., [51]). Impressive work exists on inducing format transformation rules using programming-by-example techniques (see, e.g., [37, 38, 49, 73, 82]). If examples are provided by humans (as is the case with most proposals), the risk of unrepresentative samples could be high due to human factors relating to tiredness, inattention, etc. The

work reported in [12] addresses this by inducing transformation examples.

Validating the transformation: Transformation components often provide immediate feedback by showing the results of transformations. When format transformation rules are induced from examples, there is the risk that the sample used is not representative, leading, therefore, to erroneous applications. In [12], in which examples themselves are induced, *k*-fold cross validation is used to identify rules that give rise to invalid transformations.

Format transformation can take place at different points in a data preparation pipeline. Early application of format transformations, e.g., before matching, can be beneficial because then matching steps stand to benefit from more consistent representations, and join operations stand to benefit from consistent representations for join attributes. However, format transformations may be carried out to increase consistency between integrated sources, and these inconsistencies may only become evident after the data has been integrated by mappings.

Data Repair

Data preparation must contend with different aspects of data quality. In the previous section, format transformation techniques sought to address the situation in which the data was correct, but was represented in inconsistent ways. In this section, we consider the situation in which there are inconsistencies in the data that can be detected using integrity constraints [30, 47].

Languages such as functional dependencies and conditional functional dependencies [30], can be used to express properties that should be exhibited by a data set, and the integrity constraints that capture these properties are sometimes written manually and sometimes inferred. In this section, we will consider *functional dependencies*, as a straightforward example of a language for expressing integrity constraints. A *functional dependency* φ is of the form $X \rightarrow Y$, where X and Y are sets of one or more attributes from a table. A table satisfies a functional dependency if every pair of tuples that have the same values for the attributes in X have the same values for the attributes in Y.

For example, in Fig. 1, we could have the functional dependency *postcode* → *city*, associated with each of the *Agency* sources. This represents the semantics of the application, as every postcode should be associated with exactly one city.

Integrity constraints can be used either to impose properties on a data set or to detect inconsistencies. For example, in relational databases, integrity constraints can be expressed, and the database management system will reject any transaction that seeks to violate any of the constraints. However, in data preparation there is often little control of the data sources being used, and thus it is of more interest to *detect* and *repair* violations of constraints.

Detecting violations: Detecting violations typically involves running a query over the data set, and thus is straightforward. In the case of a functional dependency of the form $X \rightarrow Y$, a query to detect violations groups the table by the attributes in X, and counts the number of distinct Y values; where there is more than one Y value for a group, this is a violation of the constraint. Returning to the example in Fig. 1, and the functional dependency *postcode* → *city*, *Agency3* is consistent with this constraint, but *Agency2* is inconsistent with the constraint for postcode *E14 6JW*.

Repairing violations: Where a constraint is violated, there is then the question as to how to repair the violation, with a view to reducing the number of inconsistencies with the constraints. One option is to delete all the tuples that violate the constraint, but this is likely to have the side-effect of discarding valuable data. Alternatively, for functional dependencies, an option is to change the left or right-hand side of the violating tuples, in such a way as to remove the inconsistency. In the case of *Agency2* and the postcode *E14 6JW*, a suitable repair would likely be to replace *Greater London* with *London* for the property at *2 Canton Street*. In practice, automating data repair is not straightforward. Where a functional dependency is violated, several different actions may be able to repair it: changing the left-hand side (i.e., changing the *postcode* in our example), changing the right-hand side (i.e., changing the *city* in our example), or changing the constraint. In addition, fixing one constraint may violate another. As a result, there has been significant work on data repair algorithms that seek to reduce the number of inconsistencies with the integrity constraints, while satisfying some other objective, such as minimizing the number of changes to the original database. For functional dependencies, several proposals have been made, some of which also support changes to constraints (e.g., [10, 19]), and reviews report techniques for other languages [30, 47]. There have also been proposals for techniques and systems that bring together a variety of data repair techniques [35, 70].

Data repair may be able to be applied to individual sources, but typically the integrity of the end data product is a priority, and thus data repair tends to take place quite late in the data preparation process.

Data Preparation Approaches

This section describes how the functionalities from Section "Data Preparation Functionalities" surface in different approaches to data preparation, specifically program based, workflow based, dataset based and automation based. An overview of how the functionalities are supported is provided in Table 1, and the details follow below.

Program Based

In program-based approaches, the data preparation task is manually encoded by a software developer, using the functionalities and libraries of a programming language. As a result, the way in which a task is encoded can be significantly influenced by the available libraries, skills and time for developing a solution, though in general software developers take fine-grained control over the data preparation process. In this section, Python is used to illustrate how some of the pivotal tasks in the example real estate domain could be implemented. Where suitable, we adopt practices proposed in [52, 62]. However, in the code snippets we present we have aimed for clarity of understanding and purpose, not for performance or scalability.

Data Profiling

Given data sets to be wrangled, such as those in Fig. 1, the developer will be interested in achieving a better understanding of the properties of the (potentially numerous) data sets. For example, as it may be necessary to combine different data sets, it is likely to be useful to answer questions such as:

- How many tuples are there in the different tables containing agency data?
- Which attributes contain high numbers of nulls, as such missing data may need to be filled in, or may reduce the suitability of specific data sets?
- Which inclusion dependencies exist on postcodes, as these provide potential ways of joining data sets?
- Which inclusion dependencies exist between the agency tables, as these may help us to establish if there are significant overlaps between the tables?

Answers to such questions could be obtained by profiling the available data sets. In the program-based approach, comprehensive data profiling could make use of a third party tool, such as Metanome [66]. However, within Python itself, the `pandas` [62] library provides descriptive statistics (including histograms) as well as some time series operations. The snippet of Python code in Fig. 2 illustrates how null-value information can be computed using `pandas`. Here, we assume that the first row in the CSV file associates an attribute name to each column and the second row associates a type with each attribute.

For the example in Fig. 1, a call to `get_null_info(Agency1)` would return the information that `price` is the only attribute with null values: it has one null in four tuples, and therefore a null ratio of 0.25.

The above code snippet shows that it is a relatively simple task to code solutions to the first two questions posed above (i.e., those relating to value distributions). However, the last two questions (i.e., those centered on inclusion dependencies) would be much harder to code solutions for, as the required algorithms are significantly more complex.

Matching

Understanding the matches between attributes is essential for several tasks. For example, matches between source and target attributes (e.g., *DeprStats.crimedecile* and *Target.crime*) can be used when selecting data from sources to populate the target. As for data profiling, sophisticated third party tools, such as COMA++ [5], could be used to identify candidate matches, though here we consider inferring matches directly in Python.

To illustrate this, consider the snippet of Python code in Fig. 3. It uses `Jellyfish`,[6] a library for approximate and phonetic matching of strings, to illustrate how string similarity, as captured by distance metrics, can underpin the matching of attribute names. In the snippet, assume `x` and `y` to be attribute names, and similarity to lie in the [0,1]-interval, with 0 denoting identity. For one example in Fig. 1, the similarity score for *crimedecile* and *crime* is 0.2787.

This code snippet performs, somewhat simplistically, the tasks of eliciting and combining evidence of similarity. A more comprehensive approach would also analyze the similarity between attribute values or properties. Furthermore, for the task of proposing semantic correspondence, one needs to decide which pairs of matches to select for each given target column, for example, by adopting selection heuristics from the literature [5].

Mapping

The main tasks that comprise the mapping stage of a data preparation process are the detection of merge opportunities between source tables, and the construction and selection of mapping expressions.

For the example in Fig. 1, from the data profiling and matching stages, one will have gathered the various value distributions and the postulated semantic correspondences. The semantic correspondences may suggest that projecting out *street*, *city*, *postcode*, *price* and *agency* from *Agency1* and *Agency2* and taking their union is a good mapping for populating the corresponding attributes in the target. Moreover, the value distributions may suggest that relevant referential constraints hold (e.g., with *Postcodes.postcode* as a primary key referenced by a foreign key, e.g., *Deprivation*

[6] https://www.github.com/jamesturk/jellyfish.

Statistics.postcode). Indeed, populating the attributes *Target.crime* and *Target.ward* will require the use of joins.

Assume that a decision has been reached to use the following mapping (expressed in relational algebra, and, for conciseness, using abbreviated names for relations and attributes):

$$\pi_{st,ci,po,pr,ag}(A_1) \cup \pi_{st,ci,po,pr,ag}(A_2)) \bowtie (\pi_{po,wa}(P) \bowtie \pi_{po,cr}(D)).$$

In relational algebra, $\pi_{a_1,...,a_n}(T)$ retains the attributes $a_1, ..., a_n$ from the table T, $\cup$ unions its operands, and $\bowtie$ joins the tuples from its operand tables when they have equal values for attributes with the same names. Then, the Python code snippet in Fig. 4 shows how `pandas` functionality can be used to implement (simplistic) versions of the relational-algebraic operators required (viz., rename, project, union and join). Then, using the matches to pick columns to project with renaming, the mapping can be written as an operator tree with eleven nodes (N1–N11, in Fig. 4).

Evaluating the code in Fig. 4 over the sources when the instances are as shown in Fig. 1 would populate the target with a single tuple, as illustrated in Table 2.

If developers wanted, they could code various mappings in the way exemplified above and select the most appropriate for the data preparation task in hand.

Format Transformation

In the program based approach, format transformations are handcrafted. As with mapping, ideally one would like to generate format transformation rules automatically from examples. However, in the absence of a tool for doing so, a Python developer has to provide the appropriate rules.

Assume that a transformation rule (t-rule, for short) is a pair, whose left-hand side (LHS) is a regular expression and whose right-hand side (RHS) is a replacement string. In the context of Figs. 1, 5 gives examples of t-rules that, resp., delete a class of substrings (e.g., a prefix number in *street* values), abbreviate certain terms (e.g., the word `Avenue` in *street* values) and remove punctuation (e.g., in *price* values). Given a set of t-rules for a given table and a given attribute, the `apply_t_rules` function formats the values of the attribute in the table so that they comply to the t-rules.

The example t-rules in Fig. 5 on *street* values could be used to make the ones in *Agency1* and in *Agency2* consistent with the implicit expectations for them in the *Target*.

Data Repair

In the program-based approach, repairs are, again, handcrafted. Assume the following (admittedly blunt) repair strategy. Let $A \rightarrow B$ be a postulated functional dependency, such as might be obtained by data profiling. Then, assume that an FD is checked on a data set and violations are detected. For example, it is observed that for an element in the domain (i.e., values in the extent of A) one may find more than one element in B. Then, if there is a sufficiently predominant value, we assume it is the correct value and we propagate it to all tuples. By predominant, we might mean the most frequently occurring value of B occurs with a frequency above a certain threshold (say, 0.5). The snippet in Fig. 6 codes this strategy in Python.

With respect to Fig. 1, there seems to be a functional dependency *agency_PC* $\rightarrow$ *agency* in the *Agency2* table. If so, the repair strategy in the code snippet above could be used to repair a faulty value for *agency*. Note, for example, that the value for *agency* in the tuple whose *street* value is *10 Canton Street* might be repaired using this strategy.

Workflow Based

In workflow-based approaches, programs are also manually crafted, but typically using a visual programming language in which different components are connected together on a canvas that describes how the data flows through the program. This is a well-established approach among commercial data preparation tools, which are often referred to as *Extract-Transform-Load* (ETL) systems [79]. Commercial products tend to be supplied with large libraries of components that support access to different types of data source, provide a wide range of data manipulation and storage capabilities, and in some cases carry out a variety of analyses. Examples of commercial ETL vendors include Talend,[7] KNIME,[8] Infogix,[9] Informatica,[10] CloverETL,[11] Alteryx[12] and Matillion.[13] Typically, in workflow-based tools, a palette of data processing components is made available to the user. Users can typically drag and drop components onto a canvas, to handcraft a data processing pipeline.

Typically, the user interface comprises:

- A set of data sources. These data sources can contain static files or API calls that return data in real time.
- A palette of data processing components. Data inputs and outputs to these components are typically data flows, allowing the user to create a specified target data set from a set of initial sources.

[7] www.talend.com.
[8] www.knime.com.
[9] www.infogix.com.
[10] www.informatica.com.
[11] www.cloverdx.com.
[12] www.alteryx.com.
[13] www.matillion.com.

Table 1 Support for the functionalities within each paradigm

Functionality	Approach	Description
Profiling	Program	Profiling informs the coding of data preparation steps. Languages such as Python have extensive profiling libraries for their internal data structures
	Workflow	Profiling informs the connecting of data sets and configuration of some components. Systems may provide profiling in catalogs, or within individual components
	Dataset	Profiling makes explicit the properties of data sets and identifies data that may require attention, and can inform the expression of proposed actions
	Automation	Profiling data provides evidence for automated components, and may be central to explaining why specific actions have been taken
Matching	Program	There is no explicit target; target attributes are created as an output from the preparation process
	Workflow	There is no explicit target, but components for merging data sources may use matches to inform data derivation
	Dataset	There is often no explicit target, but where there is, matches can be derived to align source attributes to the target
	Automation	Matches are often central, aligning sources with the target; other automated steps build on these matches
Mapping	Program	Mappings are handcrafted, either expressed directly in the code, or using query libraries
	Workflow	Mappings are expressed using components that combine data sets, and dependencies are expressed as workflow edges
	Dataset	Most manipulations involve individual data sets, but systems may support merge operations or links to workflows
	Automation	Mappings can be generated based upon metadata or profiling data
Transformation	Program	Transformations are manually coded, typically using pattern matching libraries
	Workflow	Transformations are manually encoded, using specialized languages or through programming language interfaces
	Dataset	Transformations are usually expressed using custom languages, informed by profiling, sometimes with suggestions
	Automation	Transformations are synthesized from examples
Repair	Program	Repairs are handcrafted, typically detecting conditions and carrying out actions
	Workflow	Repairs are handcrafted, often using components that support custom condition-action rule languages
	Dataset	Repairs may be supported by rule languages, for detecting and/or responding to detected conditions
	Automation	Certain repairs can be automated with reference to dependable data sets, such as master data

- A canvas on which the user can handcraft the workflow. This is done in practice by adding data sources and processing components, and offers a visual representation of the flow, as shown in Fig. 7.

Configuring and running a workflow becomes essentially a visual programming task, requiring technical skill and familiarity with the data. While dragging and dropping components on a canvas seems intuitive and straightforward, individual components may be both powerful and complex. This complexity is illustrated in Fig. 8, in which the user manually specifies a mapping. Automation in component configuration is sometimes present, aiming at simplifying the process. As an example, the *Auto map!* link in Fig. 8 will match attributes with the same name.

In this section, we illustrate how the different data preparation functionalities are supported in a representative ETL system, namely *Talend Open Studio for Data Integration*. Figure 7 illustrates part of the canvas of a data flow in Talend Open Studio, in which the user makes use of a component to merge the contents of two sources into one. Specifically, the *tUnite* component unions the contents of deprivation data for Oxford and Manchester into a file containing deprivation data for both regions.

Data Profiling

Data profiling can provide users with information about data sources and intermediate results that may inform their use in practice, and thus their place within a data preparation job. When authoring ETL flows, users take fine-grained control over which components are used, how they are configured and how they are connected, so profiling mostly provides information about the available data and associated relationships to inform decision-making.

Talend Open Data Studio offers statistics about the sources. Furthermore, to derive profiling data, among the plethora of available components, users can design workflows to, e.g., extract unique values from a column, or even create bar charts and graphs to illustrate the value distributions.

```
import pandas as pd
# Use the pandas shape method to get the
# dimensions of the dataframe as a (rows,columns)
# pair. Scan the nulls dataframe, by attribute
# and row, computing nulls_count and null_ratio.
# Construct a dictionary with the attribute name
# as key and the pair # (null_count,null_ratio)
# as value.
def get_null_info(dataframe):
  nulls = dataframe.isnull()
  # isnull generates a dataframe where each cell is
  # True only if the corresponding value cell is null
  null_info = {}
  for attr in nulls.columns:
    null_count = 0
    for val in nulls[attr]:
      if val: null_count += 1
    null_info[attr] = (null_count,
      null_count/float(nulls.shape[0]))
  return (null_info)
```

Fig. 2 Using Python to derive profiling data on nulls

Matching

Data matching identifies relationships between the attributes of data collections, in particular identifying pairs of attributes that may be related. When writing ETL flows, the relationships between tables can indicate alternative sources for the same type of data. However, when writing data preparation flows, there is typically no explicit target up front, so identifying tables that may be relevant to a future flow may involve search or discovery tasks over a data catalog. However, within the Map Editor component in Fig. 8, relationships between input and output attributes are specified, with automated matching based upon names.

Mapping

Mapping is the process whereby data sets are combined to create new data sets, for example in the case study through joining property data with associated crime rankings. Furthermore, data sets may be unioned to bring together different sources to provide a more complete picture. In ETL systems, users make explicit which join and union operations are applied to which data sets, moving incrementally from the sources towards the intended target.

In Talend Open Data Studio, a *tJoin* component can be used to perform an inner or outer join between the main data flow and a lookup flow. Similarly, a *tUnite* component, as illustrated in Fig. 7 can be used to union source contents. A graphical interface for configuring mappings is also available in the *tMap* component, offering the user the capability to manually configure mappings from the sources.

```
import jellyfish as jf
def similarity_score(x,y):
  # compute normalized scores per distance-metric
  l = jf.levenshtein_distance(x,y) /
    float(max(len(x),len(y)))
  j = 1-jf.jaro_distance(x,y)
  w = 1-jf.jaro_winkler(x,y)
  # take the mean of the scores
  s_s = ((l + j + w) / 3)
  return s_s
```

Fig. 3 Using Python to compute similarity scores for matching

Format Transformation

Format transformation is the process whereby the representation of attribute values is revised to make them more consistent. In ETL systems, such transformations are handcrafted, and components are provided that allow the construction of expressions that reformat data values and derive new values from existing ones (e.g., by adding the local tax rate to a net price). These expressions may be written in custom transformation languages, or using expressions written directly in an underlying programming language.

Talend Open Data Studio offers a custom transformation language. Using an Expression Builder, the user can modify data flows by applying transformation functions. Consider for instance the following expression, applied to a column `bedroom_number`:

```
= StringHandling.CHANGE
  (row13.bedroom_number, }}[0 – 9]″, }}″).
```

The result of applying the expression to the column will be replacing non-numeric values in column bedroom_number of the Main data flow with the empty string, thus keeping only numeric values for bedrooms. This means that the input value `"2 bedroom(s)"` will be transformed to `"2"`. A variety of system functions allow the user to express transformations on a column, or on a combination of columns.

Data Repair

Data repair is the process of changing or completing data sets, based on supplementary evidence. In some ETL systems there are specialized components for enforcing business rules, or they may build on generic data transformation components.

In Talend Open Data Studio, filling in missing values can be done using the tMap component. Consider for instance applying the following expression to `row13.street_address_raw`, while assuming that `row13` is the main data flow and `row14` is the lookup data flow in a *tMap* component that uses a postcode column as a key:

```
  StringHandling.LEN(row14.street_name)>
0
        ? row14.street_name
        : row13.street_address_raw
```

As a result of applying this expression, values of the main data flow will be replaced with the values of the lookup flow, if they exist in the latter, or they will otherwise be left intact.

Dataset Based

The dataset-based approach to processing data, using a spreadsheet-like interface, is one of the most well-established ways to interact with data, and spreadsheet software remains ubiquitous among data users [74]. Dataset based data preparation capabilities are designed to resemble these ubiquitous solutions. Many data preparation platforms, such as Altair Monarch,[14] Trifacta Wrangler,[15] OpenRefine[16] and Tableau Prep,[17] include dataset-based interactions. It is frequently the case that a dataset based interface is part of a wider software ecosystem, allowing different types of interaction with the data. In this section, the focus is on the spreadsheet-like interface capabilities, not the wider ecosystem that may embed these capabilities.

In the dataset-based approach to data preparation, the user has a full overview of the dataset columns and their properties. As a result, obtaining basic statistics, and applying simple statistical functions tends to be straightforward. Performing actions on the data, however, may be more complex, as the user is usually expected to write formulas and apply functions to process the data.

To highlight the properties of a column, it is common for dataset based interfaces to display a histogram of value distributions and to identify patterns in the data values in the form of regular expressions, with a view to identifying where data cleaning or transformations may be required [42, 69, 76]. Detection of patterns and regularities in the data also eases feature engineering in data science projects.

In Fig. 9, an example of a dataset-based interface is shown, as available in Trifacta Wrangler. The interface, initially based on Potter's Wheel [69], has since evolved to enable more direct manipulation interactions, as well as predictive interactions [46]. While users can have a clear focus on processing specific columns from a source, they can still union or join columns that originate from different sources, or assign a data target, to help automation in Trifacta. We note that it is becoming increasingly common for dataset-based tools to offer the ability to specify workflows,

```
import pandas as pd

# define relational-algebraic operators
def Rename(input, paired_list):
  return input.rename(index=str, columns=paired_list)
def Project(input, proj_list):
  return input[proj_list]
def Union(left, right):
  output = pd.DataFrame(columns=list(left.columns))
  return output.append(left).append(right)
def Natural_join(left, right):
  output = pd.DataFrame(columns=list(left.columns)
    + list(right.columns))
  output = pd.merge(left,right)
  return output

# given source-to-target matches
Agcy1 = {'street':'st','city':'ci',
  'postcode':'po','price':'pr','agency':'ag'}
Agcy2 = {'street':'st','city':'ci',
  'postcode':'po','price':'pr','agency':'ag'}
Agcy3 = {'street':'st','city':'ci','postcode':'po'}
Depr = {'postcode':'po','crimedecile':'cr'}
PCodes = {'postcode':'po','ward':'wa'}
Schools = {'ward':'ward'}

# define the mappings as relational-algebraic plans
N4 = Agcy1
N3 = Rename(Project(N4,Agcy1.keys()),Agcy1)
N6 = Agcy2
N5 = Rename(Project(N6,Agcy2.keys()),Agcy2)
N2 = Union(N3,N5)
N9 = PCodes
N8 = Rename(Project(N9,PCodes.keys()),PCodes)
N11 = Depr
N10 = Rename(Project(N11,Depr.keys()),Stats)
N7 = Natural_join(N8,N10)
N1 = Natural_join(N2,N7)
```

Fig. 4 An approach to expressing mappings in Python

and vice-versa, thus combining the capabilities of the two categories.

Data Profiling

Information about value distributions is useful in order to detect patterns, or detect missing values, outliers or anomalies in the data. In Trifacta, illustrations of value distributions are provided with every column, which is helpful for coming to terms with the data at hand, considering one column at a time. Trifacta Wrangler incorporates visual profiles of the available data, which are, in essence, automatically generated summary visualizations of the data. These are primarily aimed at increasing human understanding of the available data sources, but they also trigger transformation suggestions, as illustrated in Fig. 9.

[14] www.altair.com.

[15] www.trifacta.com.

[16] https://www.openrefine.org.

[17] www.tableau.com.

Table 2 Example target

Target						
Street	City	Postcode	Price	Agency	Crime	Ward
Biscayne avenue	London		E1W 1AD	A1 Branch2	3	St Katharines

```
import re
# given a table, an attribute and a rule set, apply
# every rule in the set to every value of the attribute
# in the table
def apply_t_rules(table, attr, rules):
  cardinality = table.shape[0]
  # for each value in the column
  for i in range(0, cardinality):
    # provided it is not a NaN
    if table[attr].iloc[i] == table[attr].iloc[i]:
      # apply each rule given, destructively assigning
      for rule in rules:
        table[attr].iloc[i] =
          re.sub(rule[0], rule[1], table[attr].iloc[i],
            0, re.MULTILINE)
  return True

# transformation rules for street values
Street_t_rules = []
# if the value has a numeric prefix, remove it
LHS = r"[0-9]+[A-Za-z,. ]"
RHS = ""
Street_t_rules.append((LHS,RHS))
# abbreviate 'Avenue' to 'Ave'
LHS = r"Avenue"
RHS = "Ave"
Street_t_rules.append((LHS,RHS))
```

Fig. 5 Format transformation in Python

```
import pandas as pd
from scipy.stats import mode
# FD A → B is represented as ['A', 'B']
def apply_fd_based_repair(df, fd_list, threshold):
  for fd in fd_list:
  # find all unique values of the determinant
  u_vals = df[fd[0]].unique()
  for u_val in u_vals:
    # given a unique value of the determinant
    # find all values for its dependent
    corr_vals = list(df.loc[df[fd[0]] == u_val, fd[1]])
    # find the most frequent dependent value
    mode_val = mode(corr_vals)
    pop_size = len(corr_vals)
    # compute the frequency of the mode
    mode_freq = mode_val[1][0]/float(pop_size)
    # if the mode frequency is less than one and
    # above the given threshold, repair
    if mode_freq < 1 and mode_freq > threshold:
      df.loc[df[fd[0]] == u_val, fd[1]] = mode_val[0][0]
  return df
```

Fig. 6 An approach to data repair in Python

Matching

Matching is important when the task at hand involves populating a target with data from several sources. The task of discovering relationships between attributes belonging to different sources is not generally central to dataset-based interfaces, that focus primarily on one table at a time.

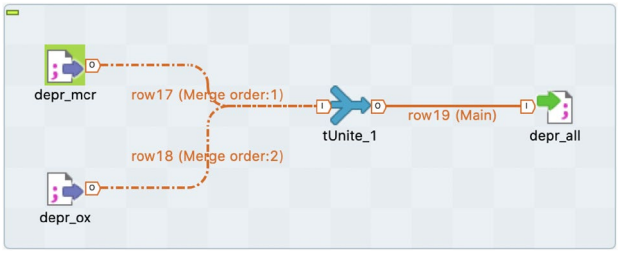

Fig. 7 Visual programming interface to a workflow-based system

However, Trifacta Wrangler allows data engineers to define a target schema to populate. Targets can then be attached to sources. This provides automated guidance for how diverse data sources may be structured, formatted and renamed in order to match to the target. Moreover, it provides the ability for users to manually define matches, based on suggestions that take into account either the column name alone (column name match), or the instance data as well (fuzzy match). As shown in Fig. 10, the interface can help the user when processing data to populate a target; columns *postcode* and *price* are identified as potential matches to the target, based on their column names.

Mapping

Joins and Unions between sources can be manually defined in recipes, which are essentially scripts that contain sequences of actions. For example, from a data set view, it is possible to select a union operation that can be used to add rows from another source to the current one. Having identified the data sets to be unioned, the principal interaction involves aligning columns across the two data sets.

Format Transformation

A wide range of transformation functions is provided in Trifacta. Besides the transformations that users can author, Trifacta can detect patterns in the data and recommend plausible transformations.

As shown in the right-hand side of Fig. 9, the software can suggest meaningful actions to take on the data. These suggestions are based on the observation that for a range of simple interactions (e.g., selecting a column or text), only a few transformations make sense [46]. This led to the *predictive interaction* approach adopted in Trifacta

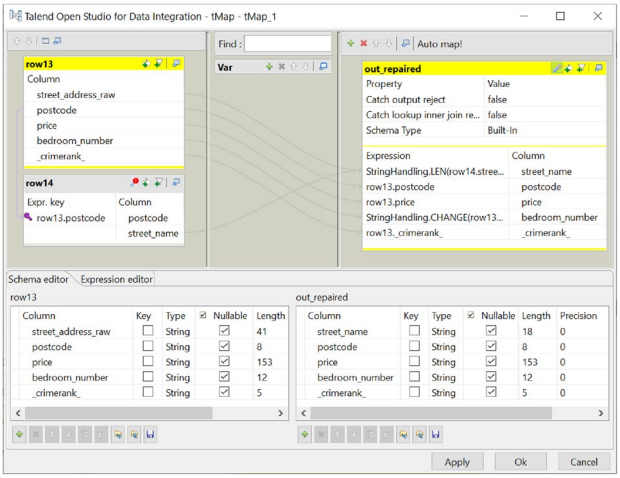

Fig. 8 Map Editor component in Talend Open Studio for Data Integration

Wrangler, which can be considered to be an analogous to auto-completion.

However, software suggestions will not always include what the user has in mind, so the software allows definition of custom calculation formulae. In Fig. 11, a simple case conversion is defined. The formula `SUBSTITUTE(PROPER(town_raw), ',', "")` in Fig. 11 can be saved as a macro and executed on other columns as well, as converting to title case and removing commas may be generally helpful in the data wrangling task at hand.

The creation of reusable scripts is a common feature in dataset-based tools. For instance, in OpenRefine, while simple transformations such as changing the text case, and trimming trailing spaces, are straightforward to perform, an expression language allows for more complex transformations. In OpenRefine, the expression `value.toTitlecase().replace(',', "")` will have the same effect as the Trifacta formula in Fig. 11.

Data Repair

In a way similar to transforming data, custom formulas can also be used to repair data in Trifacta, e.g., by expressing logic that fills in empty values with the values from another column. For instance, the following single-row formula will fill in missing values from column12 with the values from column3: `IF(ISMISSING([column12]),column3,column12)`.

Therefore, data can be repaired using formulas such as the above, which are essentially conditional data transformations. These formulas add the ability to repair data, however not as a first class citizen. Simple repairs, related to missing or mismatching data, are easier to define in Trifacta, as it is easy to replace problematic values with, e.g., the last valid value, the rolling average, etc.

In terms of helping ensure data quality, Trifacta also supports dataset-specific rules that allow the user to apply tests to the data and assess data quality with respect to these rules. For instance, the following simple rule will test whether the column Price contains positive numbers: `Price> 0`. These rules, however, are merely there to help to the user assess data quality, and do not trigger actions on the data.

Automation Based

As handcrafting of the data preparation steps from Section "Data Preparation Functionalities" is labor-intensive, there has been growing interest in the use of automation to support data preparation.[18] Automation tends to mean that, given a problem description and some evidence, software takes responsibility for carrying out the task that would normally be hand-coded. Automation has been applied to individual data preparation steps and to more complete data preparation processes.

Note that automated systems are rarely black boxes; it is not realistic to expect users to trust an automated data preparation component or system without evidence that the trust is justified. Thus, automated systems tend to show users the results of the automated steps, and often also show how the automation was carried out, thereby maintaining provenance. In the light of such information, changes can be made that affect how automation is carried out. For example, this may mean changing the evidence (such as training data) used to inform the automation of a step or process.

In the research community, most of the focus has been on individual steps. Some data preparation steps are typically carried out automatically; for example, it is less than obvious why there would be manual involvement in data profiling [1], and candidate matches have long been produced using schema and instance evidence [68]. For example, in the well-established COMA++ schema matching systems [5], there are several primitive matchers that compare intensional data (e.g., column names) and extensional data (e.g., column values), and similarity scores from the primitive matchers are then aggregated by composite matchers to identify candidate matches. However, other tasks are often carried out manually, even though techniques have been

[18] There has been work to model the cost of data preparation [58], but the true cost of specific tasks, and the most suitable metrics with which to characterize a data preparation problem could likely benefit from further investigation.

developed for automating the generation of format transformations [38], schema mappings [29] and data repair [30].

In commercial products, there are also examples of automation or semi-automation of individual steps; for example, format transformations are synthesized from examples in Excel using FlashFill [38] and in MagicFill from Talend,[19] and automatic extraction of tables from reports is supported by AutoDefine from Altair Monarch.[20]

There are fewer examples of more end-to-end automation. In Tamr [75], sources are aligned with a target representation, and duplicates are identified, informed by evidence collected from users familiar with the application domain. The associated commercial offering is used to consolidate different data sets inside enterprises relating to customers, suppliers, clinical trials, etc.[21] In Auto-Pipeline [83], given a description of a target result, a search is made for Python code fragments that can populate the target from a given starting point.

Here, we describe automation in data preparation using Data Preparer, a descendant of VADA [57]. In Data Preparer, the problem description is: given a target, some sources and some optional instances that are related to the target, populate the target with data from the sources. Several steps within Data Preparer use evidence in the form of instances that are associated with attributes in the target, which is referred to as the *data context* [55].

For the real estate scenario, the target takes the form of a table definition, as illustrated in Fig. 1, and the data context consists of reference data on addresses, associated with the *street*, *city* and *postcode* attributes in the target. Given the sources and the data context, as illustrated in Fig. 1, Data Preparer has enough information to directly populate the target from the sources, as discussed for the different data preparation steps below.

Data Profiling

Data profiling is used in Data Preparer to provide evidence to inform other steps, in particular *mapping* and *data repair*. Specifically, for each source, *candidate keys* are inferred, and between the sources *inclusion dependencies* are derived, as described in Section "Data Profiling". For the example, in Fig. 1, *postcode* is a candidate key for *DeprStats*, and the values in the *postcode* attributes in the *Agency* tables are fully included in *postcode* on *DeprStats*.

[19] https://www.talend.com.

[20] https://www.altair.com/monarch/.

[21] https://www.tamr.com.

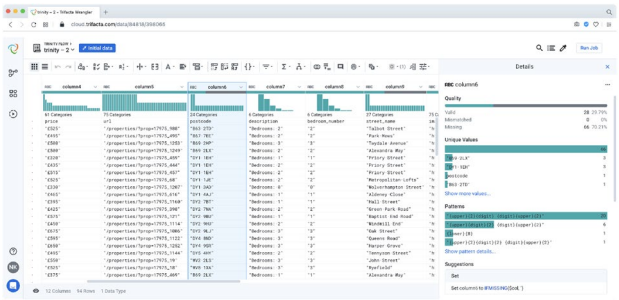

Fig. 9 A dataset-based system front-end

Matching

Matching is used in Data Preparer to identify which source attributes are candidates to populate which target attributes; a match between a source and a target attribute means that the source attribute is a candidate to populate the target attribute. Specifically, two attributes are compared considering the string similarity of their names and the level of overlap between the values in their extents.

For example, Fig. 12 illustrates some of the matches detected by Data Preparer in the real estate case study. For example, we can see that *agent.postcode*, *agent.price* and *agent.street_address_raw* have been matched with *postcode*, *price* and *street_name*, respectively, in the target. The confidence in the match is lowest for the match of *agent.street_address_raw* with *street_name*, as the attribute names are somewhat different, and the values in *agent.street_address_raw* are not identical to those in the data context for *street_name*.

Mapping

In Data Preparer, mappings are queries that describe how the target can be populated with data from one or more sources. Mappings are generated; given the results of matching (that indicate which source attributes can be used to populate which target attributes) and the results of data profiling (that indicate how tables can be joined), Data Preparer searches through a space of candidate mappings, preferring those that promise to populate as many target attributes as possible, without introducing lots of null values [61].

Specifically, a dynamic programming algorithm starts by considering which pairs of sources may be candidates for combining using *union* or *join*, to give a collection of candidate mappings of size *2* (i.e., that combine two sources). These mappings of size *2* are then combined with other sources, to give mappings of size *3*, and with each other to give mappings of size *4*. Technical challenges to be addressed include: (i) the derivation of profiling data for the mappings from the profiling data of their operands; and (ii)

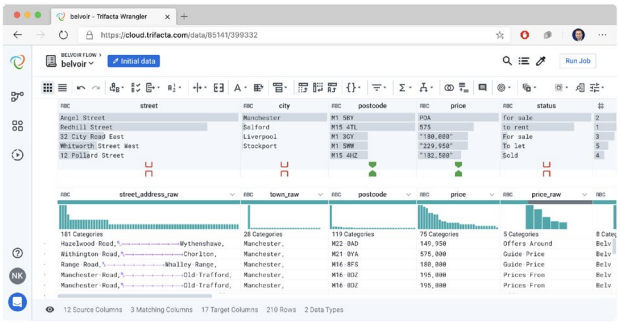

Fig. 10 Matching a source to a target in Trifacta

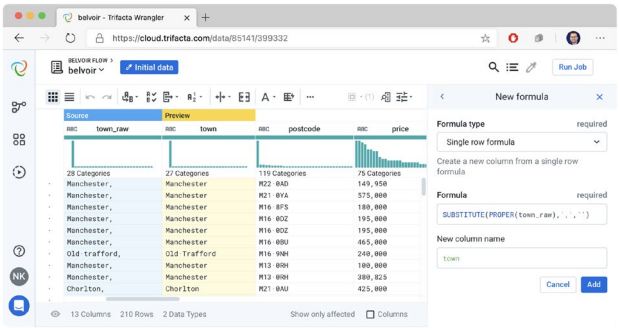

Fig. 11 Defining a custom transformation formula in Trifacta Wrangler

pruning the search space, so that only promising mappings are retained [61].

Figure 13 shows how an agency table has been combined with a *deprivation* table by joining on *postcode* to populate the target. This is a valid join that has been correctly identified. However, in the scenario, some inappropriate joins were identified, for example joining identifier columns from agencies with columns in deprivation tables on the deciles of different areas. The *id* columns in agencies were identified by profiling as candidate keys, and the *decile* columns had inclusion dependencies with the *id* columns, causing Data Preparer to conclude that the tables could be joined. However, semantically such a join makes no sense. This shows the importance of automated systems allowing the user to observe the decisions made by the automated system, and the provision of ways to influence or override inappropriate decisions. In this case, in Data Preparer, it is possible to exclude selected inclusion dependencies from consideration in mapping generation.

Format Transformation

The role of format transformation in Data Preparer is to encourage consistent representation of attribute values that are included in the result. Whenever a source attribute matches a target attribute for which data context data are available, Data Preparer will search for transformations that can make the representation of the source more consistent with that in the target. This search for transformations involves looking for data examples where the source attribute values contain tokens that are similar to those in target attribute values [12], and then trying to generate transformation programs that can translate source values to the format in the target.

The automated generation of format transformation programs tends to build on programming by example [38]. For example, given the examples *Alvaro A.A. Fernandes* → *A. Fernandes* and *Martin Koehler* → *M. Koehler*, we might hope to be able to learn a program that produces shortened variants of people's names. In Data Preparer, an approach called *SynthEdit* is used that seeks to provide near state-of-the-art program synthesis while also scaling for use with substantial training sets that are discovered rather than provided manually [11]. Specifically, SynthEdit provides an approach based on edit distance, in which program synthesis searches for the smallest number of edit operation that can translate a source string pattern into a target string pattern.

In the example scenario, one of the web data extracted agency sources has a *town_raw* attribute containing values such as *Ordsall Lane, Salford, M5 4TD*; as such, the *town* attribute contains not only the name of the town (*Salford*) but also the name of the street and the postcode. However, by considering instance matching, this attribute can match with the *city* attribute in the target, and it is then possible to infer a transformation that extracts the town name from the address.

The automatic inference of such transformations depends on the provision of suitable training examples (in this case extracted from the data context) and fairly regular representations from which to transform the data.

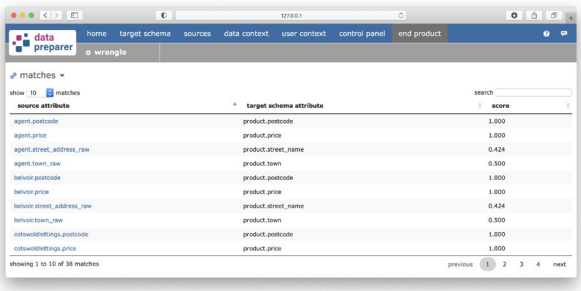

Fig. 12 Example matches for the real estate scenario in Data Preparer

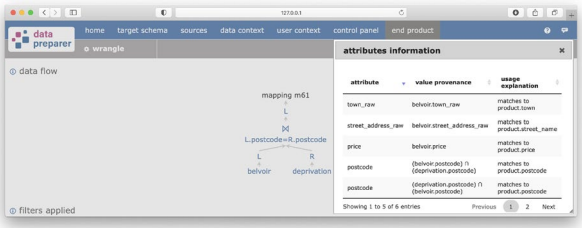

Fig. 13 An example mapping for the real estate scenario in Data Preparer. The interface shows the query plan, and how attribute values in the target are populated from the sources

Data Repair

In general, data repair may implement business rules or take steps to make source data more consistent with established data sets, such as master data. In Data Preparer, repairs are based on functional dependencies detected within the data context. In the example scenario, data profiling infers that the *postcode* functionally determines the *street* (i.e., every property with the same postcode is on the same street). As a result of this, in any sources with a missing street name, if their *postcode* is contained in the data context, the associated street can be used.

Conclusions

There are ever more data to analyze, within organizations, available on the web and streaming from sensors. However, the ability to obtain value from data faces challenges with practicalities: discovering suitable data, extracting the data from its original sources, identifying connections between data sets and resolving representational inconsistencies. All this before identifying the most suitable analysis technique or learning algorithm, and configuring it to the problem. As a result, data preparation is a big deal.

In this paper, we have presented key functionalities within data preparation, and described how they surface in different products. As summarized in Table 1, the key functionalities covered (profiling, matching, mapping, format transformation and data repair) are relevant to all the prominent data preparation paradigms reviewed (program based, workflow based, dataset based and automation based), but the ways in which they surface differ significantly from paradigm to paradigm.

If there are trends in data preparation, they are perhaps towards convergence and automation. In relation to convergence, increasingly software platforms bring together broader capabilities; for example, this is represented by several systems supporting both workflow and dataset-based interfaces, the former for bringing together different data sets and the latter for manipulating individual data sets. In relation to automation, increasingly automated components are becoming available in workflow systems, and dataset-based systems provide suggestions to inform user actions. Data preparation techniques are also increasingly being integrated with functionalities for data extraction, data governance, and data analysis, to produce platforms that are highly capable, but often complex.

Reflecting the multi-faceted nature of data preparation, there is ongoing research of relevance to data preparation across a wide front. In data lakes, there is research on discovering, grouping, selecting and inter-relating data sets at scale [13, 64, 65, 77], thus identifying promising areas of focus for data preparation. In data transformation, there is research on synthesizing and discovering transformations at the structure [32, 49, 84] and value levels [2, 44, 49]. In data repair, there are developments that accommodate a variety of constraint languages [35] and on using different types of evidence to inform repair [20]. Of relevance to human-in-the-loop data preparation, there is research on synthesizing readable programs [25], explainability [9] and active learning [67]. There is also ongoing work to bring innovations in other areas into data preparation, such as deep learning [78] and embeddings [17]. As a result, although data preparation is a well-established component in business intelligence and data science architectures, building on a significant body of work, cost-effectiveness remains a challenge, and there is much to do.

Acknowledgements We are pleased to acknowledge the support of the UK Engineering and Physical Sciences Research Council, through the VADA Programme Grant (EP/M025268/1), and the H2020 I-BiDaaS project (grant agreement No. 780787).

Data availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Abedjan Z, Golab L, Naumann F. Profiling relational data: a survey. VLDB J. 2015;24(4):557–81.
2. Abedjan Z, Morcos J, Ilyas IF, et al. Dataxformer: a robust transformation discovery system. In: 32nd IEEE International Conference on Data Engineering, ICDE, 2016; pp. 1134–1145, https://doi.org/10.1109/ICDE.2016.7498319
3. Ali SMF, Wrembel R. From conceptual design to performance optimization of ETL workflows: current state of research and open problems. VLDB J. 2017;26(6):777–801. https://doi.org/10.1007/s00778-017-0477-2.
4. Arenas M, Barceló P, Libkin L, et al. Foundations of data exchange. Cambridge: Cambridge University Press; 2014.
5. Aumueller D, Do HH, Massmann S, et al. Schema and Ontology Matching with COMA++. In: Proceedins of 2005 ACM SIGMOD International Conference on Management of Data. ACM, 2005; 906–8. https://doi.org/10.1145/1066157.1066283
6. Azarmi B. Talend for Big Data. Packt Publishing 2014.
7. Bahri M, Salutari F, Putina A, et al. AutoML: state of the art with a focus on anomaly detection, challenges, and research directions. Int J Data Sci Anal. 2022. https://doi.org/10.1007/s41060-022-00309-0
8. Bellahsene Z, Bonifati A, Rahm E. Schema Matching and Mapping. 2011. https://doi.org/10.1007/978-3-642-16518-4.
9. Bertossi LE, Geerts F. Data quality and explainable AI. ACM J Data Inf Qual 2020;12(2):11:1–11:9. https://doi.org/10.1145/3386687
10. Beskales G, Ilyas IF, Golab L, et al. On the relative trust between inconsistent data and inaccurate constraints. In: 29th IEEE International Conference on Data Engineering, ICDE, 2013; pp. 541–552.
11. Bogatu A, Fernandes AAA, Paton NW, et al. Synthedit: Format transformations by example using edit operations. In: 22nd International Conference on Extending Database Technology. OpenProceedings.org, 2019a:714–717. https://doi.org/10.5441/002/edbt.2019.94
12. Bogatu A, Paton NW, Fernandes AAA, et al. Towards automatic data format transformations: data wrangling at scale. Comput J. 2019;62(7):1044–60. https://doi.org/10.1093/comjnl/bxy118.
13. Bogatu A, Fernandes AAA, Paton NW, et al. Dataset discovery in data lakes. In: 36th IEEE International Conference on Data Engineering, ICDE. IEEE, 2020:709–720. https://doi.org/10.1109/ICDE48307.2020.00067
14. Bogorny V, Engel PM, Alvares LO. A reuse-based spatial data preparation framework for data mining. In: Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'2005), Taipei, Taiwan, Republic of China, July 14–16, 2005;649–652.
15. Bonfitto S, Casiraghi E, Mesiti M. Table understanding approaches for extracting knowledge from heterogeneous tables. WIREs Data Mining Knowl Discov 2021;11(4) https://doi.org/10.1002/widm.1407.
16. Bouman R, van Dongen J. Pentaho Solutions: Business Intelligence and Data Warehousing with Pentaho and MySQL. Wiley Publishing. 2009.
17. Cappuzzo R, Papotti P, Thirumuruganathan S. Creating embeddings of heterogeneous relational datasets for data integration tasks. In: Proc. 2020 International Conference on Management of Data, SIGMOD. ACM, 2020:1335–49. https://doi.org/10.1145/3318464.3389742.
18. Chapman A, Simperl E, Koesten L, et al. Dataset search: a survey. VLDB J. 2020;29(1):251–72. https://doi.org/10.1007/s00778-019-00564-x.
19. Chiang F, Miller RJ. A unified model for data and constraint repair. In: Proceedings of the 27th International Conference on Data Engineering, ICDE, 2011;446–457.
20. Chu X, Morcos J, Ilyas IF, et al. KATARA: A data cleaning system powered by knowledge bases and crowdsourcing. In: Proc. 2015 ACM SIGMOD International Conference on Management of Data. ACM, 2015;1247–61. https://doi.org/10.1145/2723372.2749431.
21. Deng D, Fernandez RC, Abedjan Z, et al. The data civilizer system. In: CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research 2017.
22. Diaz O, Kushibar K, Osuala R, et al. Data preparation for artificial intelligence in medical imaging: A comprehensive guide to open-access platforms and tools. Physica Med. 2021;83:25–37. https://doi.org/10.1016/j.ejmp.2021.02.007. https://www.sciencedirect.com/science/article/pii/S1120179721000958
23. Doan A, Domingos PM, Halevy AY. Reconciling schemas of disparate data sources: A machine-learning approach. In: Proc. ACM SIGMOD international conference on Management of data, 2001:509–520. https://doi.org/10.1145/375663.375731
24. Doan A, Halevy AY, Ives ZG. Principles of Data Integration. Morgan Kaufmann, 2012. http://research.cs.wisc.edu/dibook/
25. Drosos I, Barik T, Guo PJ, et al. Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists. In: CHI '20: CHI Conference on Human Factors in Computing Systems. ACM, 2020:1–12. https://doi.org/10.1145/3313831.3376442.
26. Elmagarmid AK, Ipeirotis PG, Verykios VS. Duplicate record detection: a survey. IEEE Trans Knowl Data Eng. 2007;19(1):1–16. https://doi.org/10.1109/TKDE.2007.250581.
27. Emani CK, Cullot N, Nicolle C. Understandable big data: a survey. Comput Sci Rev. 2015;17:70–81. https://doi.org/10.1016/j.cosrev.2015.05.002.
28. Fagin R, Kolaitis PG, Miller RJ, et al. Data exchange: semantics and query answering. TCS. 2005;336(1):89–124.
29. Fagin R, Haas LM, Hernández M, et al. Clio: Schema mapping creation and data exchange. In: Conceptual Modeling: Foundations and Applications, LNCS, vol. 5600. Berlin: Springer; 2009. p. 198–236.
30. Fan W, Geerts F. Foundations of Data Quality Management. Morgan & Claypool 2012.
31. Ferrara E, Meo PD, Fiumara G, et al. Web data extraction, applications and techniques: a survey. Knowl Based Syst. 2014;70:301–23. https://doi.org/10.1016/j.knosys.2014.07.007.
32. Fink M, Meilicke C, Stuckenschmidt H. Explaining differences between unaligned table snapshots. In: Proc. 23rd International Conference on Extending Database Technology, EDBT. OpenProceedings.org, 2020:133–144. https://doi.org/10.5441/002/edbt.2020.13
33. Furche T, Gottlob G, Libkin L, et al. Data wrangling for big data: Challenges and opportunities. In: EDBT, 2016:473–478. https://doi.org/10.5441/002/edbt.2016.44
34. Gal A. Uncertain Schema Matching. Morgan & Claypool 2011.
35. Geerts F, Mecca G, Papotti P, et al. Cleaning data with llunatic. VLDB J. 2020;29(4):867–92. https://doi.org/10.1007/s00778-019-00586-5.
36. van Gennip Y, Hunter B, Ma A, et al. Unsupervised record matching with noisy and incomplete data. Int J Data Sci Anal. 2018;6(2):109–29. https://doi.org/10.1007/s41060-018-0129-7.
37. Gulwani S. Automating string processing in spreadsheets using input-output examples. In: Proc. 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL, 2011:317–330
38. Gulwani S, Harris WR, Singh R. Spreadsheet data manipulation using examples. Commun ACM. 2012;55(8):97–105.

39. Guo C, Hedeler C, Paton NW, et al. Matchbench: Benchmarking schema matching algorithms for schematic correspondences. In: 29th British National Conference on Databases, BNCOD, 2013:92–106. https://doi.org/10.1007/978-3-642-39467-6_11
40. Halevy AY. Answering queries using views: a survey. VLDBJ. 2001;10(4):270–94. https://doi.org/10.1007/s007780100054.
41. Hameed M, Naumann F. Data preparation: a survey of commercial tools. SIGMOD Rec. 2020;49(3):18–29.
42. He J, Veltri E, Santoro D, et al. Interactive and deterministic data cleaning: A tossed stone raises a thousand ripples. Proc ACM SIGMOD International Conference on Management of Data 26-June-20 2016:893–907. https://doi.org/10.1145/2882903.2915242.
43. He X, Zhao K, Chu X. Automl: a survey of the state-of-the-art. CoRR abs/1908.00709. 2019 arXiv:1908.00709
44. He Y, Jin Z, Chaudhuri S. Auto-transform: learning-to-transform by patterns. Proc VLDB Endow. 2020;13(11):2368–2381. http://www.vldb.org/pvldb/vol13/p2368-he.pdf
45. Hellerstein JM, Heer J, Kandel S. Self-service data preparation: Research to practice. IEEE Data Eng Bull 2018a;41(2):23–34. http://sites.computer.org/debull/A18june/p23.pdf
46. Hellerstein JM, Heer J, Kandel S. Self-Service Data Preparation: Research to Practice. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 2018b:23–34
47. Ilyas IF, Chu X. Trends in cleaning relational data: consistency and deduplication. Found Trends Datab. 2015;5(4):281–393. https://doi.org/10.1561/1900000045.
48. Ioannidis YE. The history of histograms (abridged). In: VLDB. Morgan Kaufmann, 2003:19–30
49. Jin Z, Anderson MR, Cafarella MJ, et al. Foofah: Transforming data by example. In: Proc. of the 2017 ACM International Conference on Management of Data, SIGMOD. ACM, 2017:683–698, https://doi.org/10.1145/3035918.3064034
50. Kandel S, Heer J, Plaisant C, et al. Research directions in data wrangling: Visualizations and transformations for usable and credible data. Inf Vis. 2011;10(4):271–88.
51. Kandel S, Paepcke A, Hellerstein J, et al. Wrangler: Interactive visual specification of data transformation scripts. In: CHI, 2011b:3363–3372
52. Kazil J, Jarmul K. Data Wrangling with Python: Tips and Tools to Make Your Life Easier, 1st edn. O'Reilly Media, Inc. 2016.
53. Kim W, Choi I, Gala SK, et al. On resolving schematic heterogeneity in multidatabase systems. Distributed and Parallel Databases. 1993;1(3):251–79. https://doi.org/10.1007/BF01263333.
54. Kluyver T, et al. Jupyter notebooks - a publishing format for reproducible computational workflows. In: Loizides F, Schmidt B (eds) 20th International Conference on Electronic Publishing. IOS Press, 2016:87–90, https://doi.org/10.3233/978-1-61499-649-1-87
55. Koehler M, Abel E, Bogatu A, et al. Incorporating data context to cost-effectively automate end-to-end data wrangling. IEEE Trans Big Data. 2021;7(1):169–86. https://doi.org/10.1109/TBDATA.2019.2907588.
56. Konstantinou N, Koehler M, Abel E, et al. The VADA architecture for cost-effective data wrangling. In: Proc. ACM international conference on management of data, SIGMOD; 2017. p. 1599–602.
57. Konstantinou N, Abel E, Bellomarini L, et al. VADA: an architecture for end user informed data preparation. J Big Data. 2019;6:74. https://doi.org/10.1186/s40537-019-0237-9.
58. Kruse S, Papotti P, Naumann F. Estimating data integration and cleaning effort. In: Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015:61–72, https://doi.org/10.5441/002/edbt.2015.07, https://doi.org/10.5441/002/edbt.2015.07
59. Marnette B, Mecca G, Papotti P, et al. ++spicy: an opensource tool for second-generation schema mapping and data exchange. PVLDB. 2011;4(12):1438–41.
60. Maynard-Atem L. The data series - data democratisation. Impact. 2019;2019(1):10–1. https://doi.org/10.1080/2058802X.2019.1594871.
61. Mazilu L, Paton NW, Fernandes AAA, et al. Schema mapping generation in the wild. Inf Syst. 2022;104(101):904. https://doi.org/10.1016/j.is.2021.101904.
62. McKinney W. Python for Data Analysis, 2nd edn. O'Reilly Media, Inc. 2018.
63. Mecca G, Papotti P, Santoro D. A short history of schema mapping systems. In: Twentieth Italian Symposium on Advanced Database Systems, SEBD 2012, 2012:99–106, http://sebd2012.dei.unipd.it/documents/188475/efd4de94-b0b6-4979-8f60-3628f30d6f03
64. Nargesian F, Zhu E, Miller RJ, et al. Data lake management: Challenges and opportunities. Proc VLDB Endow 2019;12(12):1986–1989. https://doi.org/10.14778/3352063.3352116
65. Nargesian F, Pu KQ, Zhu E, et al. Organizing data lakes for navigation. In: Proceedings of the 2020 International Conference on Management of Data, SIGMOD. ACM, 2020; 1939–1950, https://doi.org/10.1145/3318464.3380605
66. Papenbrock T, Bergmann T, Finke M, et al. Data profiling with metanome. Proc VLDB Endow 2015;8(12):1860–1863. https://doi.org/10.14778/2824032.2824086
67. Qian K, Popa L, Sen P. Active learning for large-scale entity resolution. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM. ACM, 2017:1379–1388, https://doi.org/10.1145/3132847.3132949
68. Rahm E, Bernstein PA. A survey of approaches to automatic schema matching. VLDBJ. 2001;10(4):334–50. https://doi.org/10.1007/s007780100057.
69. Raman V, Hellerstein JM. Potter's wheel: An interactive data cleaning system. VLDB 2001 - Proceedings of 27th International Conference on Very Large Data Bases 2001:381–390
70. Rekatsinas T, Chu X, Ilyas IF, et al. Holoclean: Holistic data repairs with probabilistic inference. Proc VLDB Endow 2017;10(11):1190–1201. https://doi.org/10.14778/3137628.3137631
71. Rostin A, Albrecht O, Bauckmann J, et al. A machine learning approach to foreign key discovery. In: 12th International Workshop on the Web and Databases, WebDB 2009.
72. Santu SKK, Hassan MM, Smith MJ, et al. Automl to date and beyond: Challenges and opportunities. ACM Comput Surv 2022;54(8):175:1–175:36. https://doi.org/10.1145/3470918,
73. Singh R. Blinkfill: Semi-supervised programming by example for syntactic string transformations. PVLDB. 2016;9(10):816–27.
74. Stodder D. Improving Data Preparation for Business Analytics. Tech. rep., 2016. https://info.talend.com/rs/talend/images/WP_EN_DP_Improving_DataPrep_BusinessAnalytics.pdf
75. Stonebraker M, Bruckner D, Ilyas IF, et al. Data curation at scale: The data tamer system. In: CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research 2013.
76. Sukhobok D, Nikolov N, Roman D. Tabular Data Anomaly Patterns. Proceedings - 2017 International Conference on Big Data Innovations and Applications, Innovate-Data 2017 2018-January:25–34. 2018. https://doi.org/10.1109/Innovate-Data.2017.10

77. Terrizzano I, Schwarz PM, Roth M, et al. Data wrangling: The challenging journey from the wild to the lake. In: CIDR 2015.
78. Thirumuruganathan S, Tang N, Ouzzani M, et al. Data curation with deep learning. In: Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020. OpenProceedings.org, 2020:277–286, https://doi.org/10.5441/002/edbt.2020.25
79. Vassiliadis P. A survey of extract-transform-load technology IJDWM. 2011;5(3):1–27.
80. Verborgh R, Wilde MD. Using OpenRefine, 1st edn. Packt Publishing 2013.
81. Waller T, Korbel J, Stys M. Cloveretl designer: User's guide. Javlin: Tech. rep; 2018.
82. Wu B, Knoblock CA. An iterative approach to synthesize data transformation programs. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015:1726–1732, http://ijcai.org/Abstract/15/246
83. Yang J, He Y, Chaudhuri S. Auto-pipeline: Synthesize data pipelines by-target using reinforcement learning and search. Proc VLDB Endow 2021;14(11):2563–2575. http://www.vldb.org/pvldb/vol14/p2563-he.pdf
84. Zhu E, He Y, Chaudhuri S. Auto-join: Joining tables by leveraging transformations. Proc VLDB Endow 2017;10(10):1034–1045. https://doi.org/10.14778/3115404.3115409