Transform-Data-by-Example (TDE): Extensible Data Transformation in Excel

Yeye He[1], Kris Ganjam[1], Kukjin Lee[1], Yue Wang[1], Vivek Narasayya[1], Surajit Chaudhuri[1], Xu Chu[2], Yudian Zheng[3]
[1]Microsoft Research, Redmond, USA
[2]Georgia Institute of Technology, Atlanta, USA
[3]Twitter Inc., San Francisco, USA
{yeyehe,krisgan,kulee,wanyue,viveknar,surajitc}@microsoft.com
xu.chu@cc.gatech.edu,yudianz@twitter.com

ABSTRACT

Business analysts and data scientists today increasingly need to clean, standardize and transform diverse data sets, such as name, address, date time, phone number, etc., before they can perform analysis. These ad-hoc transformation problems are typically solved by one-off scripts, which is both difficult and time-consuming.

Our observation is that these domain-specific transformation problems have long been solved by developers with code libraries, which are often shared in places like GitHub. We thus develop an extensible data transformation system called *Transform-Data-by-Example* (*TDE*) that can leverage rich transformation logic in source code, DLLs, web services and mapping tables, so that end-users only need to provide a few (typically 3) input/output examples, and *TDE* can synthesize desired programs using relevant transformation logic from these sources. The beta version of *TDE* was released in Office Store for Excel.

ACM Reference Format:
Yeye He[1], Kris Ganjam[1], Kukjin Lee[1], Yue Wang[1], Vivek Narasayya[1], Surajit Chaudhuri[1], Xu Chu[2], Yudian Zheng[3] . 2018. Transform-Data-by-Example (TDE): Extensible Data Transformation in Excel. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183713.3193539

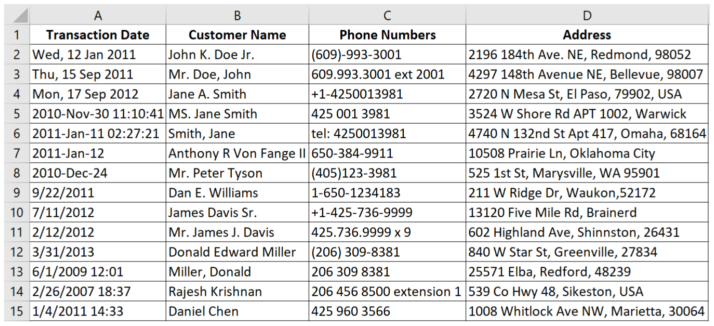
1 INTRODUCTION

Users such as business analysts and data scientists today regularly perform ad-hoc analysis using diverse data sets, which however often need to be *prepared* (a multi-step process that typically involves clean, transform, and join, among other things), before analysis can be performed. This is difficult and time-consuming for end-users – studies suggest that users spend up to 80% of time on data preparation [8].

There is increasing momentum in the industry towards *self-service data preparation* [9], where the key objective is to build intelligent systems that enable business analysts and data scientists to prepare ad-hoc data sets themselves without needing help from IT staff. This, if realized, holds the potential to democratize data analytics for a wide spectrum of users who often lack technical skills like scripting. Gartner reckons this fast growing market to be worth over $1 billion by 2019 [9]. In this work we focus on *self-service data transformation*, which is a major component in data preparation [9].

[2, 3]: Work done at Microsoft Research.

ACM ISBN 978-1-4503-4703-7/18/06...$15.00
https://doi.org/10.1145/3183713.3193539



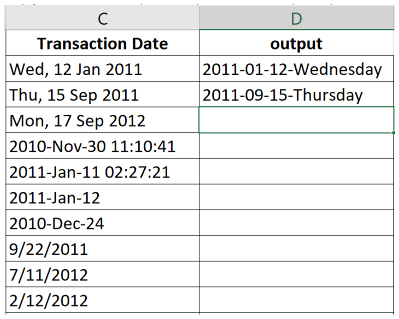
	A	B	C	D
1	**Transaction Date**	**Customer Name**	**Phone Numbers**	**Address**
2	Wed, 12 Jan 2011	John K. Doe Jr.	(609)-993-3001	2196 184th Ave. NE, Redmond, 98052
3	Thu, 15 Sep 2011	Mr. Doe, John	609.993.3001 ext 2001	4297 148th Avenue NE, Bellevue, 98007
4	Mon, 17 Sep 2012	Jane A. Smith	+1-4250013981	2720 N Mesa St, El Paso, 79902, USA
5	2010-Nov-30 11:10:41	MS. Jane Smith	425 001 3981	3524 W Shore Rd APT 1002, Warwick
6	2011-Jan-11 02:27:21	Smith, Jane	tel: 4250013981	4740 N 132nd St Apt 417, Omaha, 68164
7	2011-Jan-12	Anthony R Von Fange II	650-384-9911	10508 Prairie Ln, Oklahoma City
8	2010-Dec-24	Mr. Peter Tyson	(405)123-3981	525 1st St, Marysville, WA 95901
9	9/22/2011	Dan E. Williams	1-650-1234183	211 W Ridge Dr, Waukon,52172
10	7/11/2012	James Davis Sr.	+1-425-736-9999	13120 Five Mile Rd, Brainerd
11	2/12/2012	Mr. James J. Davis	425.736.9999 x 9	602 Highland Ave, Shinnston, 26431
12	3/31/2013	Donald Edward Miller	(206) 309-8381	840 W Star St, Greenville, 27834
13	6/1/2009 12:01	Miller, Donald	206 309 8381	25571 Elba, Redford, 48239
14	2/26/2007 18:37	Rajesh Krishnan	206 456 8500 extension 1	539 Co Hwy 48, Sikeston, USA
15	1/4/2011 14:33	Daniel Chen	425 960 3566	1008 Whitlock Ave NW, Marietta, 30064

Figure 1: A sales data set with heterogeneous data values.

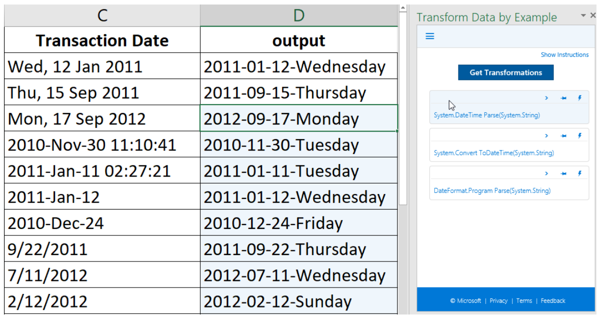
Figure 1 gives a concrete example for data transformation. This sales data set has information such as transaction dates, customer names, their phone numbers and addresses, etc. However, values in same columns are highly heterogeneous, which can often happen when data is collected from different sources, or when values are manually entered. In this example, date values in the first column have many different formats. In the second column, some customer names are first-name followed by last-name, while others are last-name followed by comma and first-name, with various optional salutations (Mr., Dr., etc.) and suffixes (III, Jr., etc.). Similarly, phone number and address columns are also highly inconsistent.

This data set is obviously not ready for analysis yet – an analyst wanting to figure out which day-of-the-week (Mon, Tue, etc.) has the most sales, for instance, cannot find that out by executing a SQL query or a pivot table using this data, as day-of-the-week is missing from the input. However, deriving day-of-the-week from date strings is non-trivial even for programmers, and the heterogeneity of date values only adds to the complexity. Similarly, the analyst may want to analyze sales with a group-by on area code from phone-numbers, or zip-code from addresses, both of which again require difficult data transformations.

Our observation is that these domain-specific transformation problems like date-time parsing and address standardization have



C	D
Transaction Date	output
Wed, 12 Jan 2011	2011-01-12-Wednesday
Thu, 15 Sep 2011	2011-09-15-Thursday
Mon, 17 Sep 2012	
2010-Nov-30 11:10:41	
2011-Jan-11 02:27:21	
2011-Jan-12	
2010-Dec-24	
9/22/2011	
7/11/2012	
2/12/2012	

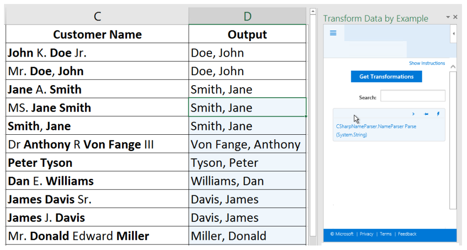


C	D
Transaction Date	output
Wed, 12 Jan 2011	2011-01-12-Wednesday
Thu, 15 Sep 2011	2011-09-15-Thursday
Mon, 17 Sep 2012	2012-09-17-Monday
2010-Nov-30 11:10:41	2010-11-30-Tuesday
2011-Jan-11 02:27:21	2011-01-11-Tuesday
2011-Jan-12	2011-01-12-Wednesday
2010-Dec-24	2010-12-24-Friday
9/22/2011	2011-09-22-Thursday
7/11/2012	2012-07-11-Wednesday
2/12/2012	2012-02-12-Sunday

System.DateTime Parse(System.String)
System.Convert ToDateTime(System.String)
DateFormat.Program Parse(System.String)

Transform Data by Example
Show Instructions
Get Transformations
© Microsoft | Privacy | Terms | Feedback

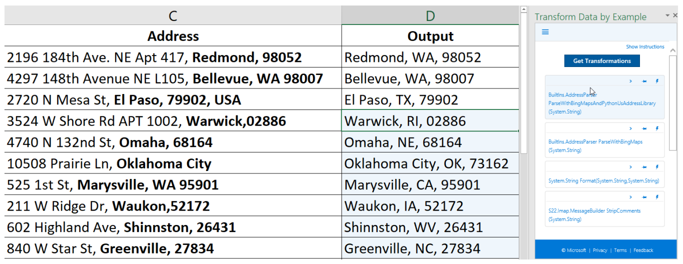
Figure 2: *TDE* transformation for date-time. (Left): input data is in column-C, user provides two desired output examples in column-D. (Right): After clicking on the "Get Transformation" button, *TDE* searches over thousands of functions to compose new programs whose output are consistent with the given examples. Within a few seconds, a ranked list of programs are returned in the right pane. Hovering over the first program (using System.DateTime.Parse from .Net) gives a preview of all results (shaded in green).



C	D
Customer Name	Output
John K. **Doe** Jr.	Doe, John
Mr. **Doe, John**	Doe, John
Jane A. **Smith**	Smith, Jane
MS. **Jane Smith**	Smith, Jane
Smith, Jane	Smith, Jane
Dr **Anthony** R **Von Fange** III	Von Fange, Anthony
Peter Tyson	Tyson, Peter
Dan E. **Williams**	Williams, Dan
James Davis Sr.	Davis, James
James J. **Davis**	Davis, James
Mr. **Donald** Edward **Miller**	Miller, Donald

CSharpNameParser.NameParser Parse (System.String)

Transform Data by Example
Show Instructions
Get Transformations
Search:
© Microsoft | Privacy | Terms | Feedback



C	D
Address	Output
2196 184th Ave. NE Apt 417, **Redmond, 98052**	Redmond, WA, 98052
4297 148th Avenue NE L105, **Bellevue, WA 98007**	Bellevue, WA, 98007
2720 N Mesa St, **El Paso, 79902, USA**	El Paso, TX, 79902
3524 W Shore Rd APT 1002, **Warwick,02886**	Warwick, RI, 02886
4740 N 132nd St, **Omaha, 68164**	Omaha, NE, 68164
10508 Prairie Ln, **Oklahoma City**	Oklahoma City, OK, 73162
525 1st St, **Marysville, WA 95901**	Marysville, CA, 95901
211 W Ridge Dr, **Waukon,52172**	Waukon, IA, 52172
602 Highland Ave, **Shinnston, 26431**	Shinnston, WV, 26431
840 W Star St, **Greenville, 27834**	Greenville, NC, 27834

BuiltIns.AddressParser ParseWithBingMapsAndPythonUsAddressLibrary (System.String)
BuiltIns.AddressParser ParseWithBingMaps (System.String)
System.String Format(System.String,System.String)
S22.Imap.MessageBuilder StripComments (System.String)

Transform Data by Example
Show Instructions
Get Transformations
© Microsoft | Privacy | Terms | Feedback

Figure 3: (Left): transformation for names. The first three values in column-D are provided as output examples. The desired first-names and last-names are marked in bold for ease of reading. A composed program using library CSharpNameParser from GitHub is returned. (Right): transformations for addresses. The first three values are provided as examples to produce city, state, and zip-code as output. Note that some of these info are missing from the input. A program invoking Bing Maps API is returned as the top result.

existed for decades – developers traditionally build custom code libraries to solve them, and share their code in places like GitHub and StackOverflow. In a recent crawl, we extracted over 1.8M functions from code libraries crawled at GitHub, and over 2M code snippets extracted from pages on StackOverflow.

We have built a production-quality data-transformation engine called *Transform-Data-by-Example* (*TDE*) that can index rich transformation logic from sources such as code, to allow users to search and reuse existing transformation logic. The front-end of *TDE* is an Excel add-in, currently in beta release at Office Store [4]. We choose Excel as the front-end to allow end-users stay in their familiar Excel environment without switching.

Unique Features. The *TDE* system has the following features that we believe are important first steps to realize the vision of self-service data transformation. (More details of the system can be found in a full research article [11]).

• *Search-by-Example. TDE* allows end-users to search transformations by examples, a paradigm known as program-by-example (PBE) [14], first used in FlashFill [10] for data transformation. Compared to existing PBE systems such as FlashFill that compose results using a small number of string primitivies, *TDE* synthesizes programs from a much larger space of arbitrary program functions and mapping tables [17]. We develop novel algorithms to make this feasible at an interactive speed, with just a few (typically three) input/output examples.

• *Program Synthesis.* Since existing functions rarely produce the exact output specified by users, *TDE* automatically synthesizes new programs, sometimes with multiple functions, to exactly match target output, all within just a few seconds. Expert-users have the option to inspect the synthesized programs to ensure correctness.

• *Head-domain Support.* We have built an instance of *TDE* that indexes over 50K functions from GitHub that can already handle many head and tail domains, such as date-time, person-name, phone-number, us-address, url, unit-conversion, etc. Many of these transformations cannot be handled by any existing system.

• *Extensibility.* Although *TDE* can already support many important domains out of the box, there will be diverse application domains where *TDE* has no built-in support as it has not encountered and crawled relevant functions from such domains. *TDE* is therefore designed to be *extensible* – users can simply point *TDE* to their domain-specific source code, DLLs, web services, and mapping tables, the transformation logic in these resources will be automatically extracted, and made immediately search-able. The way *TDE* works is just like a search engine "indexing" a new document.

2 DEMO SCENARIOS

Given the raw data set in Figure 1, a user would like to transform this data in order to perform analysis. Suppose she wants to find out which day-of-the-week (e.g., Mon, Tue, etc.) produces the most sales. Today she has to write ad-hoc scripts in order to derive such information. However, using *TDE*, she only needs to provide a few examples to specify the desired output.

Synthesis of functions. In Figure 2(left), she types in two example output for the first two input rows, and then clicks on the "Get Suggestions" button. The Excel add-in will talk to the back-end *TDE* service running on Microsoft Azure cloud, which searches over thousands of indexed functions, to on-the-fly *compose* new programs consistent with all given examples. In Figure 2(right), a ranked list of programs synthesized from existing code are returned. The top-ranked program uses the `System.DateTime.Parse()` function from the .Net system library, which is a function specifically written to handle heterogeneous date-time string values. If we hover over the suggested program, a preview for remaining output will be shown (shaded values), which all look correct. An advanced user can further click on the suggestion to review the code automatically generated, or she can simply click on the "Apply" button to accept the suggested output.

Note that although we only show one possible transformation for this example due to the space constraint, in practice the output can be in numerous other formats such as "`2011-01-12 (Wed)`", "`Jan 12, 2011, Wednesday`", or simply "`Wed`", etc. We will demonstrate all these scenarios and show that *TDE* can dynamically generate suitable programs to precisely match these different output.

Happy with her analysis on date values, the analyst now moves on to the customer column. Here she would like to identify returning customers and big spenders. A common challenge in names collected from different sales channels (e.g. online vs. in-store) is that they often differ with minor variations. For example, customers may or may not use middle-initials, suffixes (Jr., Sr., III, etc.), and salutations (Mr., Dr., etc.). Furthermore, their names may be in different formats, e.g., last-name followed by comma, then first name; or first name followed by last name. In Figure 3(left) for instance, the first two rows can both be standardized to "Doe, John"; and for next three rows to "Smith, Jane". Suppose our analyst would like to perform this standardization to ignore optional salutations, middle-initials and suffixes, in order to identify candidate records that may belong to the same persons. This is again a complex transformation that would have required non-trivial scripting. Using *TDE*, the analyst can again search for desired transformations by just providing three output examples. *TDE* is able to synthesize a program using the `CSharpNameParser.Parse()` function from GitHub that is consistent with all input/output examples. She can accept the resulting transformation with just a click of the button.

Synthesis of web services. Names alone are often insufficient to uniquely identify customers – suppose the analyst also wants to derive city, state and zip-code information from the heterogeneous "address" column in Figure 1 to help the de-duplication. The target output is shown Figure 3. Note that this is a challenging task even for experienced programmers, as the target output (zip-code, city, etc.) may be missing from the input altogether, such that domain-specific transformation logic and reference data sets (e.g., USPS reference addresses, zip-code to city mapping) are needed to perform this transformation. *TDE* can again help this transformation with a few output examples shown in Figure 3(right). In this case *TDE* synthesizes a program that invokes Bing Maps API, which is a web service in the *TDE* index that specifically handles address data. *TDE* treats web service calls the same as local function calls – it invokes the service for each input string, and then parses the resulting JSON data to automatically synthesizes the target output.

In our planned demo, we will show additional transformation examples using *TDE*, such as standardizing phone numbers, extracting domain names from URLs, performing unit-conversions, etc., as well as lookup-based transformations that require synthesized programs to holistically combine both existing functions as well as mapping tables. We omit these cases here in the interest of space.

Extensibility. While *TDE* can already support many important domains using functions we currently index, there are many tail application domains for which *TDE* would not handle out-of-box since the relevant functions are not indexed yet. We will demonstrate a transformation on `user-agent-string` data, which is currently not supported by *TDE*. We will illustrate how the *extensibility* feature allows users to add new functions and web-services (wrapped in Azure functions) into *TDE*. The new transformation logic will be automatically extracted and made immediately search-able to solve the aforementioned problem for `user-agent-string`.

In addition to these demo scenarios, attendees will be able to interact with *TDE* themselves by trying their favorite data transformation tasks and see results right away.

3 SYSTEM ARCHITECTURE

We give a high-level overview of the *TDE* back-end service, currently deployed on Microsoft Azure. More details of the underlying technologies will be described in a research paper [11].

Figure 4 shows the overall architecture of *TDE*. There are two main phases. First, there is an *offline phase* where *TDE* collects, analyzes and index transformation logic from useful resources including (1) code libraries, (2) web service APIs, and (3) mapping tables, all of which require different indexing strategies. Next, in the *online phase*, when users submit transformation tasks in Excel, the indexes built offline are used to quickly find relevant transformation logic, from which new programs are synthesized that match given input/output examples.

The offline phase. In a current instantiation of the system, we collect over 11K C# projects from GitHub with over 1.8M functions, and additionally over 2M code snippets from the StackOverflow online forum. We further index all functions in the .Net system library, and 16K mapping relationships using a variant of [17]. Note that although we focus on one language (C# in this case) for ease of program analysis, the techniques here can be easily generalized to other languages like Java and Python.

Given the large repository of code, a key technical challenge is to "understand" these functions so that at run time we can quickly find ones relevant to a given transformation task. We perform extensive offline analysis of all candidate functions, using code-analysis as well as distributional-analysis of execution output, so that we can profile typical input/output of these functions.

Mapping relationships (e.g., "Washington" to "WA"; or "SFO" to "San Francisco International Airport") work like dictionary lookup and are important ingredients to complement program logic. We start with a large crawl of HTML tables from Bing's index [7] to derive mapping-relationships [17]. *TDE* can combine the resulting mappings with code to synthesize complex programs.

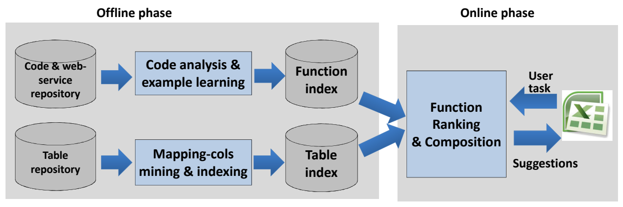

Offline phase
Online phase
Code & web-service repository
Code analysis & example learning
Function index
Table repository
Mapping-cols mining & indexing
Table index
Function Ranking & Composition
User task
Suggestions

Figure 4: Architecture of *TDE* back-end.

For web services, we use REST APIs identified from Bing's index, as well as important web service APIs manually curated that may require API keys (e.g., Bing Maps API [1]). These services are important for certain complex transformations requiring extensive external resources, for which standalone code libraries are often insufficient (e.g., the address example in Figure 3 (right)). Web service APIs are profiled and indexed by *TDE* just like functions but are invoked slightly differently (remotely over HTTP vs. locally).

The online phase. In this phase, given a user transformation task, we need to (1) leverage index structures built offline for functions, web-services and mapping-relationships, to quickly identify a small subset of relevant transformation logic for actual execution; (2) execute such relevant logic such as functions, and synthesize new programs with a combination of resources consistent with all input/output examples; and (3) if multiple candidate programs can be synthesized, then rank them based on their program "complexity" following the MDL principle. This end-to-end process needs to be highly efficient to ensure interactivity. We design novel ranking and synthesis algorithms to efficiently compose transformation logic into complex programs.

4 RELATED WORK

There are significant activities in the industry in the space of data preparation [15], producing an interesting array of solutions from both startups (e.g., Trifacta [5]), as well as established companies (e.g., Informatica Rev [2]). For data transformation, the approaches taken by existing solutions fall into five broad categories to be reviewed below. To the best of knowledge, *TDE* is the first system with all desirable features discussed in Section 1 (e.g., extensibility, by-example, composability, etc.).

Menu-based transformation. Most existing systems provide a menu of common built-in transformations, from which users are expected to browse manually. Menus typically present only a small number of simple transformations (e.g., upper/lower case, split, etc.), for otherwise menus can quickly become overwhelming and hard to navigate. As a result, they tend to be limited in functionality, and in fact no menu-based transformations in existing systems we surveyed can solve the complex tasks described in Section 2.

Language-based transformation. In addition to menus, existing systems also define their own transformation languages that users can learn to write. For example, Trifacta [5] uses a Trifacta Wrangle language, while OpenRefine [3] has its own Google Refine Expression Language (GREL). Like other domain specific languages (DSL), these tend to have steep learning curves and not easily accessible to users [13, 14]. Furthermore, like the menu-based systems, predefined languages often have a limited number of built-in operators and thus limited expressiveness.

Suggest transformation by input. Trifacta additionally uses an interesting paradigm where it suggests possible transformations based on the portion of input strings selected by users (termed *predictive interaction* [12]). However, unlike input/output examples, input alone often cannot fully specify the desired transformation. For instance, for a given selection it is not clear if user would like to delete, insert, or update (let alone more fine-grained options like upper vs. lower vs. camel casing, etc.). As a result users would often have to deal with a large number of irrelevant suggestions.

Example-driven search using string operators. Although program-by-example (PBE) is well-known problem in the literature [14], FlashFill [10] pioneered its use for data transformation. In FlashFill and related systems [13, 16], users provide input/output examples, and the system will use simple *string operators* (e.g. split and substring) to find consistent programs. While the PBE paradigm is a big step forward in terms of ease-of-use (which *TDE* also builds upon), the expressiveness of existing systems is limited to simple string operations, and cannot handle complex transformations that require domain-specific knowledge described above.

Example-driven search with search engines. DataXFormer [6] takes an interesting approach of using search engines to find relevant web tables and web forms for transformations. Appropriate input/output column headers, however, are required to construct keyword queries in this approach (e.g., if an input column has a header "inch" and an output column has a header "cm", then a query "convert inch to cm" will be automatically constructed for search engines to find relevant web forms). Note that in practice the column headers are often missing or ill-suited as search queries (e.g., the header may be "measurement" instead of "cm"), thus causing difficulties (in comparison, *TDE* does not require headers). Furthermore, while web forms and tables are useful, they often cannot handle common tasks (including all cases described above) when the desired transformations need to be be synthesized (as opposed to simple wrapper induction).

REFERENCES

[1] Bing maps api. https://www.microsoft.com/maps/choose-your-bing-maps-API.aspx.
[2] Informatica Rev. https://www.informatica.com/products/data-quality/rev.html.
[3] Openrefine. openrefine.org.
[4] Transform Data by Example (from Microsoft Office Store). https://appsource.microsoft.com/en-us/product/office/WA104380727.
[5] Trifacta. https://www.trifacta.com/.
[6] Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker. Dataxformer: A robust transformation discovery system. In *ICDE*, 2016.
[7] K. Chakrabarti, S. Chaudhuri, Z. Chen, K. Ganjam, and Y. He. Data services leveraging bing's data assets. *IEEE Data Eng. Bull.*, 2016.
[8] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, Inc., New York, 2003.
[9] J. Hare, C. Adams, A. Woodward, and H. Swinehart. Forecast snapshot: Self-service data preparation, worldwide, 2016. *Gartner, Inc.*, February 2016.
[10] W. R. Harris and S. Gulwani. Spreadsheet table transformations from examples. In *SIGPLAN 2011*.
[11] Y. He, X. Chu, K. Ganjam, Y. Zheng, V. Narasayya, and S. Chaudhuri. Transform-Data-by-Example (TDE): Extensible Data Transformation using Functions. In *submission*.
[12] J. Heer, J. M. Hellerstein, and S. Kandel. Predictive interaction for data transformation. In *CIDR*, 2015.
[13] Z. Jin, M. R. Anderson, M. Cafarella, and H. V. Jagadish. Foofah: Transforming data by example. In *SIGMOD*, 2017.
[14] H. Lieberman, editor. *Your Wish is My Command: Programming by Example*. Morgan Kaufmann, 2001.
[15] R. L. Sallam, P. Forry, E. Zaidi, and S. Vashisth. Gartner: Market guide for self-service data preparation. 2016.
[16] R. Singh. Blinkfill: Semi-supervised programming by example for syntactic string transformations. In *Proc. VLDB*, 2016.
[17] Y. Wang and Y. He. Synthesizing mapping relationships using table corpus. In *SIGMOD*, 2017.