Wrangler: Interactive Visual Specification of Data Transformation Scripts

Sean Kandel*, **Andreas Paepcke***, **Joseph Hellerstein**† **and Jeffrey Heer***
* Stanford University † University of California, Berkeley
skandel, paepcke, jheer@cs.stanford.edu hellerstein@cs.berkeley.edu

ABSTRACT

Though data analysis tools continue to improve, analysts still expend an inordinate amount of time and effort manipulating data and assessing data quality issues. Such "data wrangling" regularly involves reformatting data values or layout, correcting erroneous or missing values, and integrating multiple data sources. These transforms are often difficult to specify and difficult to reuse across analysis tasks, teams, and tools. In response, we introduce *Wrangler*, an interactive system for creating data transformations. Wrangler combines direct manipulation of visualized data with automatic inference of relevant transforms, enabling analysts to iteratively explore the space of applicable operations and preview their effects. Wrangler leverages semantic data types (e.g., geographic locations, dates, classification codes) to aid validation and type conversion. Interactive histories support review, refinement, and annotation of transformation scripts. User study results show that Wrangler significantly reduces specification time and promotes the use of robust, auditable transforms instead of manual editing.

Author Keywords

Data cleaning, transformation, validation, visualization, programming by demonstration, mixed-initiative interfaces.

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: UI

INTRODUCTION

Despite significant advances in technologies for data management and analysis, it remains time-consuming to inspect a data set and mold it to a form that allows meaningful analysis to begin. Analysts must regularly restructure data to make it palatable to databases, statistics packages, and visualization tools. To improve data quality, analysts must also identify and address issues such as misspellings, missing data, unresolved duplicates, and outliers. Our own informal interviews with data analysts have found that these types of transforms constitute the most tedious component of their analytic process. Others estimate that data cleaning is responsible for up to 80% of the development time and cost in data warehousing projects [4]. Such "data wrangling" often requires writing idiosyncratic scripts in programming languages such as Python and Perl, or extensive manual editing using interactive tools such as Microsoft Excel. Moreover, this hurdle discourages many people from working with data in the first place. Sadly, when it comes to the practice of data analysis, "the tedium is the message."

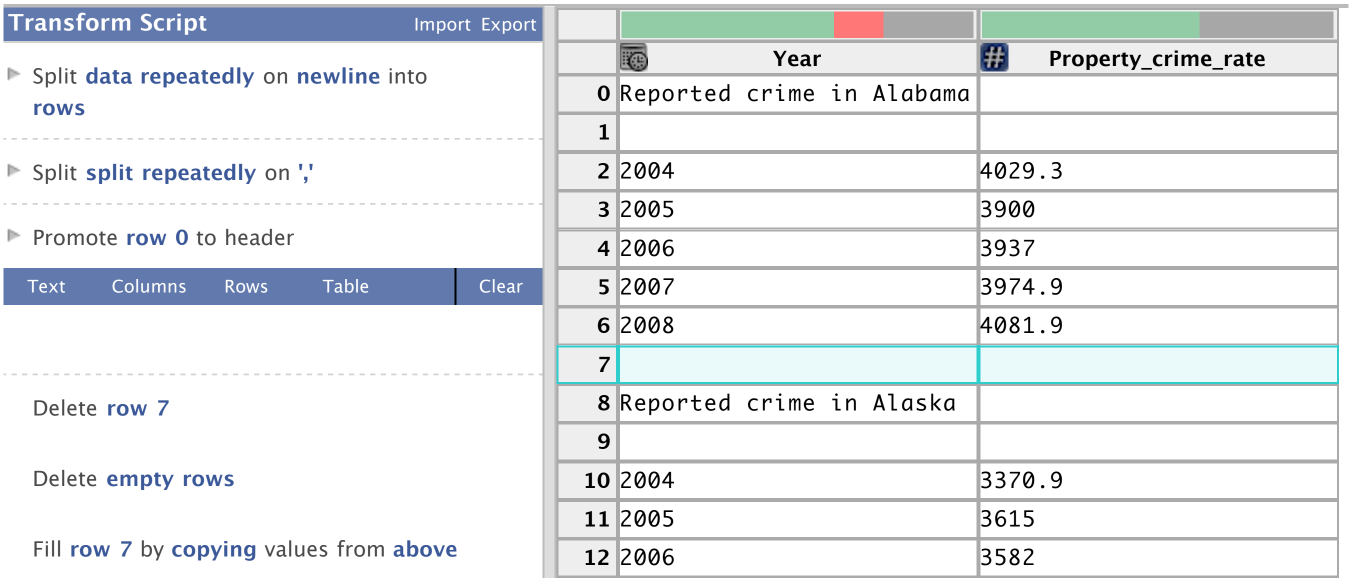
Part of the problem is that reformatting and validating data requires transforms that can be difficult to specify and evaluate. For instance, analysts often split data into meaningful records and attributes—or validate fields such as dates and addresses—using complex regular expressions that are error-prone and tedious to interpret [2, 24]. Converting coded values, such as mapping FIPS codes to U.S. state names, requires integrating data from one or more external tables. The effects of transforms that aggregate data or rearrange data layout can be particularly hard to conceptualize ahead of time. As data sets grow in size and complexity, discovering data quality issues may be as difficult as correcting them.

Of course, transforming and cleaning a data set is only one step in the larger data lifecycle. Data updates and evolving schemas often necessitate the reuse and revision of transformations. Multiple analysts might use transformed data and wish to review or refine the transformations that were previously applied; the importance of capturing data provenance is magnified when data and scripts are shared. As a result, we contend that the proper output of data wrangling is not just transformed data, but an editable and auditable description of the data transformations applied.

This paper presents the design of *Wrangler*, a system for interactive data transformation. We designed Wrangler to help analysts author expressive transformations while simplifying specification and minimizing manual repetition. To do so, Wrangler couples a *mixed-initiative user interface* with an underlying *declarative transformation language*.

With Wrangler, analysts specify transformations by building up a sequence of basic transforms. As users select data, Wrangler *suggests applicable transforms* based on the current context of interaction. Programming-by-demonstration techniques help analysts specify complex criteria such as regular expressions. To ensure relevance, Wrangler enumerates and rank-orders possible transforms using a model that incorporates user input with the frequency, diversity, and specification difficulty of applicable transform types. To convey the effects of data transforms, Wrangler provides

CHI 2011, May 7–12, 2011, Vancouver, BC, Canada.
Copyright 2011 ACM 978-1-4503-0267-8/11/05...$5.00.



▸ Split **data repeatedly** on **newline** into **rows**

▸ Split **split repeatedly** on **','**

▸ Promote **row 0** to header

Delete **row 7**

Delete **empty rows**

Fill **row 7** by **copying** values from **above**

	Year	Property_crime_rate
0	Reported crime in Alabama	
1		
2	2004	4029.3
3	2005	3900
4	2006	3937
5	2007	3974.9
6	2008	4081.9
7		
8	Reported crime in Alaska	
9		
10	2004	3370.9
11	2005	3615
12	2006	3582

Figure 1. The Wrangler Interface. The left panel contains (from top-to-bottom) a history of transforms, a transform selection menu, and automatically suggested transforms based on the current selection. Bold text within the transform descriptions indicate parameters that can be clicked and revised. The right panel contains an interactive data table; above each column is a data quality meter.

short *natural language descriptions*—which users can refine via interactive parameters—and *visual previews* of transform results. These techniques enable analysts to rapidly navigate and assess the space of viable transforms.

As analysts transform data, their steps are recorded in a script to facilitate reuse and provide documentation of data provenance. Wrangler's *interactive history viewer* supports review, refinement, and annotation of these scripts. Wrangler's high-level language supports a variety of runtime platforms: Wrangler scripts can be run in a web browser using JavaScript or translated into MapReduce or Python code.

We also present a controlled user study comparing Wrangler and Excel across a set of data wrangling tasks. We find that Wrangler significantly reduces specification time and promotes the use of robust transforms rather than manual editing. Wrangler is one piece of a larger effort to address bottlenecks in the data lifecycle by integrating insights and methods from the HCI and database communities.

RELATED WORK

The database and machine learning communities have contributed a number of algorithmic techniques for aiding data cleaning and integration. These techniques include methods for detecting erroneous values [10, 11], information extraction [1, 25], entity resolution [6], type inference [7], and schema matching [9, 21]. In the Wrangler interface we seek to surface such techniques in an accessible manner.

A number of commercial and research systems provide graphical interfaces leveraging the above methods. Many of these tools provide interfaces for schema matching or entity resolution [3, 9, 16, 23]. Toped^{++} [24] is an interface for creating *Topes*, objects that validate and transform data. Topes support transformations such as text formatting and lookups, but provide little support for filtering, reshaping, or aggregation. Bellman [5] helps users understand the structure and quality of a database, but does not enable transformations.

Many data cleaning applications apply direct manipulation and programming-by-demonstration (PBD) methods to specific cleaning tasks. Users of SWYN [2] build regular expressions by providing example text selections and can evaluate their effect in visual previews. Potluck [14] applies simultaneous text editing [19] to merge data sources. Karma [26] infers text extractors and transformations for web data from examples entered in a table. Vegemite [18] applies PBD to integrate web data, automates the use of web services, and generates shareable scripts. Other interfaces [15] apply PBD to data integration via copy and paste actions.

Wrangler applies a number of these techniques: it infers regular expressions from example selections [2] and supports mass editing [14, 19]. Wrangler uses semantic roles akin to Topes [24] and provides natural language descriptions of transforms [18]. However, Wrangler differs in important ways. PBD data tools support text extraction or data integration, but lack operations such as reshaping, aggregation, and missing value imputation. Prior tools (except for Vegemite [18]) also do not generate scripts to document provenance.

Most closely related to Wrangler is prior work on interactive data cleaning. Potter's Wheel [22] provides a transformation language for data formatting and outlier detection. Wrangler extends the Potter's Wheel language with key differences discussed later. Ajax [8] also provides an interface to specify transforms, with advanced facilities for entity resolution. Neither tool provides much support for direct manipulation: interaction is largely restricted to menu-based commands or entering programming statements. Google Refine [13] (formerly Freebase GridWorks) leverages Freebase to enable entity resolution and discrepancy detection. It provides summarization and filtering support through faceted histograms. Though users can specify some commands graphically, others must be written in a command language. Moreover, the system assumes that input data arrives in a proper tabular format, limiting the forms of data to which it can be applied.

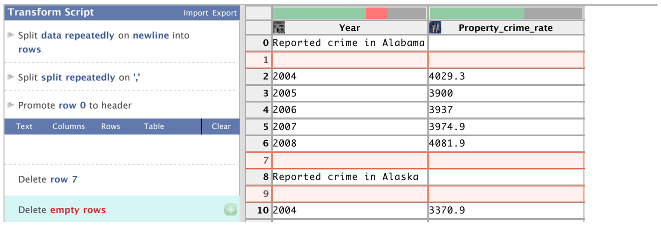
Wrangler builds on this prior work to contribute novel techniques for specifying data transforms. Wrangler provides an *inference engine* that generates and rank-orders suggested transforms in response to direct manipulation of a data table. Analysts can navigate the space of transforms by directly selecting data, indicating a desired transform via menus, or by modifying a related transform; each of these actions leads Wrangler to further refine the set of suggestions. To help analysts understand the effects of an operation before they commit to it, Wrangler's natural language transform descriptions are augmented by novel *transform previews* that visualize transform results. In concert, these techniques help analysts iteratively hone in on a desired transformation.

USAGE SCENARIO

Consider an example wrangling task, using housing crime data from the U.S. Bureau of Justice Statistics. The data were downloaded as a CSV (comma-separated values) file, but are not immediately usable by other tools: the data contains empty lines, U.S. states are organized in disjoint matrices, and the state names are embedded in other text. We describe how an analyst can use Wrangler to transform the data into more usable formats (Figures 1–7).

The analyst begins by pasting the text of the file into an input box; alternatively, she could upload the file. The interface now shows a data table (Fig. 1). To the left of the table is a panel containing an interactive history, a transform menu, and a transform editor. The history already contains three transforms, as Wrangler inferred that the data was in CSV format and so split the text into rows on newline characters, split the rows into columns on commas, and promoted the first row to be the table header. Note that the analyst could undo any transform by clicking the red undo button (which appears upon mouse-over of a transform), or could modify transform parameters in place. In this case, she has no need.

The analyst then begins wrangling the file into a usable form. The analyst could specify transforms explicitly by selecting a transform type from the menu and then assigning values to parameters; however, she instead opts to use direct manipulation along with Wrangler's suggestion engine. First, she *clicks* a row header for an empty row (7) to select it; the transformation editor suggests possible operations in response (Fig. 1). The first suggestion is to delete just the selected row. The analyst can navigate the suggestions using the keyboard *up* and *down* arrows or by mousing over the description in the editor pane. As she navigates the suggestions, Wrangler previews the effects of the transforms in the data table. For deletions, the preview highlights the candidate deleted rows in red (Fig. 2). The analyst mouses over the suggestion to delete all empty rows in the table and clicks the green *add* button to execute the transform. The system then adds the deletion operation to the history view.



Split data repeatedly on newline into rows

Split split repeatedly on ','

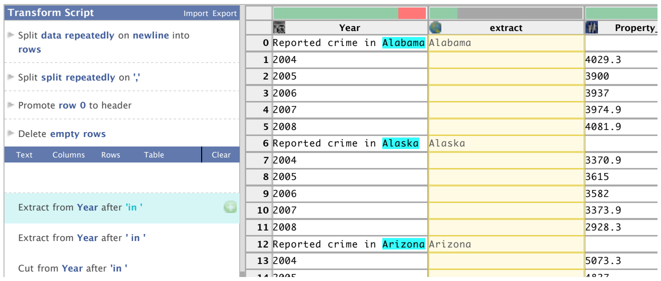
Promote row 0 to header

Delete row 7

Delete empty rows

	Year	Property_crime_rate
0	Reported crime in Alabama	
1		
2	2004	4029.3
3	2005	3900
4	2006	3937
5	2007	3974.9
6	2008	4081.9
7		
8	Reported crime in Alaska	
9		
10	2004	3370.9

Figure 2. Row deletion. The analyst selects an empty row and chooses a *delete* transform. Red highlights preview which rows will be deleted.



Split data repeatedly on newline into rows

Split split repeatedly on ','

Promote row 0 to header

Delete empty rows

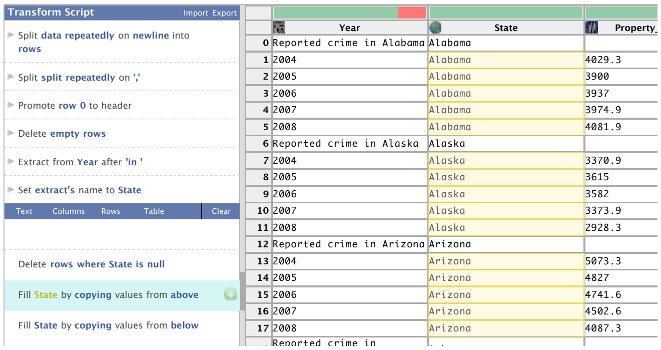
Extract from Year after 'in '

Extract from Year after ' in '

Cut from Year after 'in '

	Year	extract	Property_
0	Reported crime in Alabama	Alabama	
1	2004		4029.3
2	2005		3900
3	2006		3937
4	2007		3974.9
5	2008		4081.9
6	Reported crime in Alaska	Alaska	
7	2004		3370.9
8	2005		3615
9	2006		3582
10	2007		3373.9
11	2008		2928.3
12	Reported crime in Arizona	Arizona	
13	2004		5073.3

Figure 3. Text extraction. The analyst selects state names to *extract* them into a new column. Yellow highlights show a preview of the result.



Split data repeatedly on newline into rows

Split split repeatedly on ','

Promote row 0 to header

Delete empty rows

Extract from Year after 'in '

Set extract's name to State

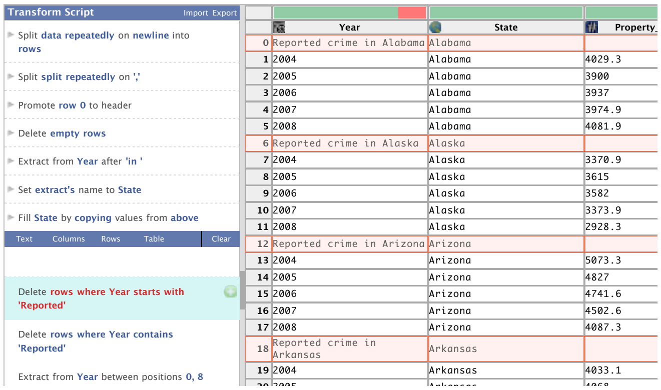
Delete rows where State is null

Fill State by copying values from above

Fill State by copying values from below

	Year	State	Property_
0	Reported crime in Alabama	Alabama	
1	2004	Alabama	4029.3
2	2005	Alabama	3900
3	2006	Alabama	3937
4	2007	Alabama	3974.9
5	2008	Alabama	4081.9
6	Reported crime in Alaska	Alaska	
7	2004	Alaska	3370.9
8	2005	Alaska	3615
9	2006	Alaska	3582
10	2007	Alaska	3373.9
11	2008	Alaska	2928.3
12	Reported crime in Arizona	Arizona	
13	2004	Arizona	5073.3
14	2005	Arizona	4827
15	2006	Arizona	4741.6
16	2007	Arizona	4502.6
17	2008	Arizona	4087.3

Figure 4. Filling missing values. The analyst populates empty cells by clicking the gray bar in the data quality meter above the "State" column, and then selecting a *fill* transform.



Split data repeatedly on newline into rows

Split split repeatedly on ','

Promote row 0 to header

Delete empty rows

Extract from Year after 'in '

Set extract's name to State

Fill State by copying values from above

Delete rows where Year starts with 'Reported'

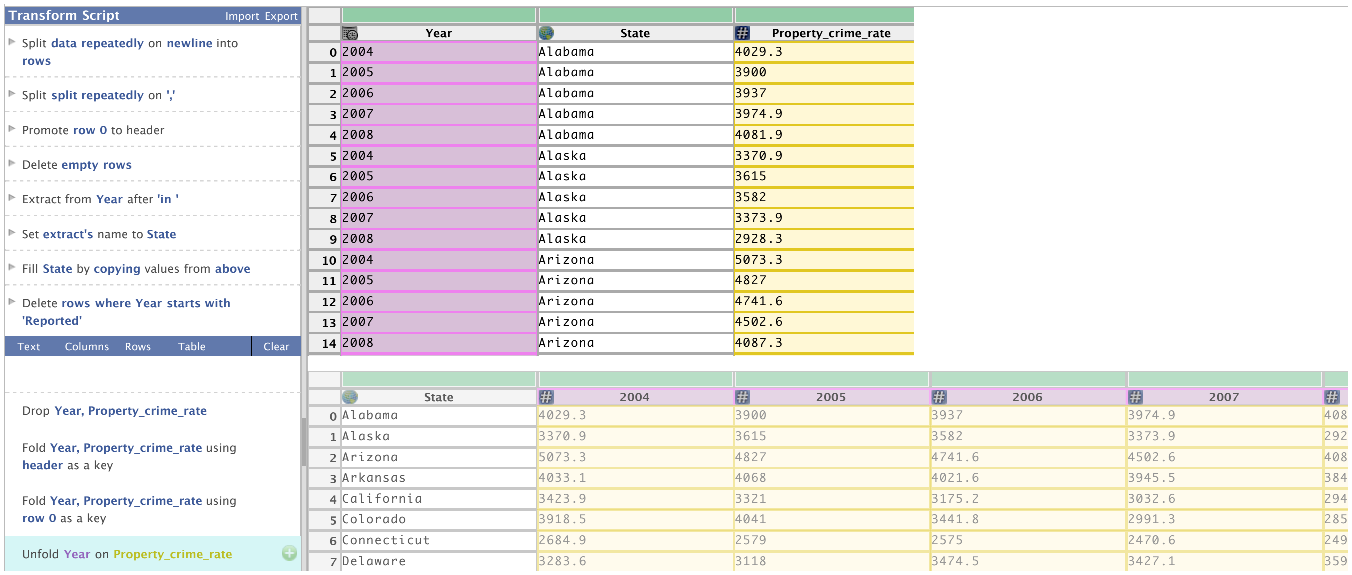
Delete rows where Year contains 'Reported'

Extract from Year between positions 0, 8

	Year	State	Property_
0	Reported crime in Alabama	Alabama	
1	2004	Alabama	4029.3
2	2005	Alabama	3900
3	2006	Alabama	3937
4	2007	Alabama	3974.9
5	2008	Alabama	4081.9
6	Reported crime in Alaska	Alaska	
7	2004	Alaska	3370.9
8	2005	Alaska	3615
9	2006	Alaska	3582
10	2007	Alaska	3373.9
11	2008	Alaska	2928.3
12	Reported crime in Arizona	Arizona	
13	2004	Arizona	5073.3
14	2005	Arizona	4827
15	2006	Arizona	4741.6
16	2007	Arizona	4502.6
17	2008	Arizona	4087.3
18	Reported crime in Arkansas	Arkansas	
19	2004	Arkansas	4033.1

Figure 5. Deleting rows. The analyst selects text in an unwanted row and selects a *delete* operation within the "Rows" menu. Red highlighting previews which rows will be deleted.

The analyst would like to compare data across states, so she now needs to extract the state names and add them to each row of the data. She selects the text 'Alaska' in row 6 of the "Year" column. Wrangler initially interprets this as selecting text at positions 18-24. The analyst updates Wrangler's inference by selecting 'Arizona' in the cell six rows below. Wrangler now suggests extracting text occurring after the string "in " (Fig. 3). The analyst executes this transform and renames the resulting column "State". She notices that the column is sparsely populated. These missing values are in-



Split data repeatedly on newline into rows

Split split repeatedly on ','

Promote row 0 to header

Delete empty rows

Extract from Year after 'in '

Set extract's name to State

Fill State by copying values from above

Delete rows where Year starts with 'Reported'

Drop Year, Property_crime_rate

Fold Year, Property_crime_rate using header as a key

Fold Year, Property_crime_rate using row 0 as a key

Unfold Year on Property_crime_rate

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

	State	2004	2005	2006	2007	
0	Alabama	4029.3	3900	3937	3974.9	408
1	Alaska	3370.9	3615	3582	3373.9	292
2	Arizona	5073.3	4827	4741.6	4502.6	408
3	Arkansas	4033.1	4068	4021.6	3945.5	384
4	California	3423.9	3321	3175.2	3032.6	294
5	Colorado	3918.5	4041	3441.8	2991.3	285
6	Connecticut	2684.9	2579	2575	2470.6	249
7	Delaware	3283.6	3118	3474.5	3427.1	359

Figure 6. Table reshaping. The analyst selects two columns, and then elects to *unfold* them to create a cross-tabulation. A ghosted table overlay previews the result. Color highlights show the correspondence of data between the start and end states.

dicated by the gray bar in the *data quality meter* above the column. The analyst clicks the gray bar and Wrangler suggests transforms for missing values. The analyst chooses to fill empty cells with the value from above (Fig. 4).

Looking at the "Year" column, the analyst notices a red bar in the data quality meter indicating inconsistent data types. Wrangler has inferred that the column primarily contains numbers, and so has flagged non-numeric values as potential errors. She decides to remove the rows containing the text 'Reported'. She selects the text 'Reported' in row 0. Wrangler suggests *split*, *extract*, and *cut* transforms, but no delete operations. In response, the analyst selects the *Delete* command from the *Rows* menu in the transform editor. This action reorders the suggestions so that *delete* commands have higher ranking. She finds the suggestion that deletes the unwanted rows (Fig. 5) and executes the transform.

At this point the analyst has wrangled the data into a proper relational format, sufficient for export to database and visualization tools. But now suppose she would like to create a cross-tabulation of crime rates by state and year for subsequent graphing in Excel. She selects the "Year" and "Property_crime_rate" columns, previews the suggested *unfold* operation (Fig. 6), then executes it to create the desired cross-tab. The *unfold* operation creates new columns for each unique value found in the "Year" column, and reorganizes the "Property_crime_rate" values by placing each in the appropriate cell in the resulting matrix.

The analyst's process results in a transformation script written in a declarative transformation language. The script provides an auditable description of the transformation enabling later inspection, reuse, and modification. The analyst can also annotate these transformations with her rationale. By clicking the *Export* button above the transformation history, the analyst can either save the transformed data or generate runnable code implementing the transformation (Fig. 7).

```
split('data').on(NEWLINE).max_splits(NO_MAX)
split('split').on(COMMA).max_splits(NO_MAX)
columnName().row(0)
delete(isEmpty())
extract('Year').on(/.*/).after(/in /)
columnName('extract').to('State')
fill('State').method(COPY).direction(DOWN)
delete('Year starts with "Reported"')
unfold('Year').above('Property_crime_rate')
```

Figure 7. The result of a data wrangling session is a declarative data cleaning script, shown here as generated JavaScript code. The script encodes a step-by-step description of how to operate on input data; a Wrangler runtime evaluates the script to produce transformed data.

DESIGN PROCESS

We based Wrangler on a transformation language with a handful of operators. Originally we thought that each of these operators might correspond to a single interaction with example data in a table view. However, after considering different mappings and evaluating their implications, we were unable to devise an intuitive and unambiguous mapping between simple gestures and the full expressiveness of the language. A given interaction could imply multiple transforms and multiple interactions might imply the same transform.

Although this many-to-many relationship between the language and interaction might complicate our interface, we found the relationship to be relatively sparse in practice: the number of likely transforms for a given gesture is small. As a result, we adopted a *mixed-initiative* approach [12]; instead of mapping an interaction to a single transform, we surface likely transforms as an ordered list of suggestions. We then focused on rapid means for users to navigate—prune, refine, and evaluate—these suggestions to find a desired transform.

Wrangler is a browser-based web application, written in JavaScript. In the next section we describe the Wrangler transformation language. We then present the Wrangler interface and its techniques for navigating suggestion space. Next, we describe Wrangler's mechanisms for verification. We go on to discuss the technical details of our inference engine.

THE WRANGLER TRANSFORMATION LANGUAGE

Underlying the Wrangler interface is a declarative data transformation language. Both prior work [8, 17, 22] and empirical data guided the language design. As our starting point we used the Potter's Wheel transformation language [22] (which in turn draws from SchemaSQL [17]). Informed by a corpus of data sets gathered from varied sources (e.g., data.gov, NGOs, log files, web APIs), we then extended the language with additional operators for common data cleaning tasks. These include features such as positional operators, aggregation, semantic roles, and complex reshaping operators (e.g., using multiple key rows for cross-tabs). We also introduced conditional mapping operators (e.g., update country to "U.S." where state="California"). Language statements manipulate *data tables* with numbered rows and named columns of data. Wrangler treats raw text as a "degenerate" table containing one row and one column. The language consists of eight classes of transforms, described below.

Map transforms map one input data row to zero, one, or multiple output rows. *Delete* transforms (one-to-zero) accept predicates determining which rows to remove. One-to-one transforms include *extracting*, *cutting*, and *splitting* values into multiple columns; reformatting; simple arithmetic; and value *updates*. One-to-many transforms include operations for splitting data into multiple rows, such as splitting a text file on newlines or unnesting arrays and sets.

Lookups and joins incorporate data from external tables. Wrangler includes extensible lookup tables to support common types of transformations, such as mapping zip codes to state names for aggregation across states. Currently Wrangler supports two types of joins: equi-joins and approximate joins using string edit distance. These joins are useful for lookups and for correcting typos for known data types.

Reshape transforms manipulate table structure and schema. Wrangler provides two reshaping operators: *fold* and *unfold*. *Fold* collapses multiple columns to two or more columns containing key-value sets, while an *unfold* creates new column headers from data values; see [22] for an extended discussion. Reshaping enables higher-order data restructuring and is common in tools such as R and Excel Pivot Tables.

Positional transforms include *fill* and *lag* operations. *Fill* operations generate values based on neighboring values in a row or column and so depend on the sort order of the table. For example, an analyst might fill empty cells with preceding non-empty values. The *lag* operator shifts the values of a column up or down by a specified number of rows.

The language also includes functions for **sorting**, **aggregation** (e.g., sum, min, max, mean, standard deviation), and **key generation** (a.k.a., *skolemization*). Finally, the language contains **schema** transforms to set column names, specify column data types, and assign semantic roles.

To aid data validation and transformation, Wrangler supports standard *data types* (e.g., integers, numbers, strings) and higher-level *semantic roles* (e.g., geographic location, classification codes, currencies). Data types comprise standard primitives and associated parsing functions. Semantic roles consist of additional functions for parsing and formatting values, plus zero or more transformation functions that map between related roles. As an example, consider a semantic role defining a *zip code*. The zip code role can check that a zip code parses correctly (i.e., is a 5 digit number) and that it is a valid zip code (checking against an external dictionary of known zipcodes). The zip code role can also register mapping functions, e.g., to return the containing state or a central lat-lon coordinate. Wrangler leverages types and roles for parsing, validation, and transform suggestion. The Wrangler semantic role system is extensible, but currently supports a limited set of common roles such as geographic locations, government codes, currencies, and dates.

The Wrangler language design co-evolved with the interface described in subsequent sections. We sought a consistent mapping between the transforms shown in the interface and statements in the language. Disconnects between the two might cause confusion [20], particularly when analysts try to interpret code-generated scripts. As a result, we chose to introduce redundancy in the language by adding operators for high-level actions that are commonly needed but have unintuitive lower-level realizations (e.g., *positional* operators can be realized using *key* transforms, self-joins, and scalar functions). The result is a clear one-to-one mapping between transforms presented in the interface and statements in output scripts. Prior work [17, 22] proves that our basic set of transforms is sufficient to handle all one-to-one and one-to-many transforms. Through both our own practice and discussions with analysts, we believe our extended language is sufficient to handle a large variety of data wrangling tasks.

THE WRANGLER INTERFACE DESIGN

The goal of the Wrangler interface is to enable analysts to author expressive transformations with minimal difficulty and tedium. To this aim, our interface combines direct manipulation, automatic suggestion, menu-based transform selection, and manual editing of transform parameters. This synthesis of techniques enables analysts to navigate the space of transforms using the means they find most convenient.

Both novices and experts can find it difficult to specify transform parameters such as regular expressions. While direct manipulation selections can help, inference is required to suggest transforms without programming. To reduce this gulf of execution [20], Wrangler uses an *inference engine* that suggests data transformations based on user input, data type or semantic role, and a number of empirically-derived heuristics. These suggestions are intended to facilitate the discovery and application of more complicated transforms.

However, suggested transforms (and their consequences) may be difficult to understand. To reduce this gulf of evaluation [20], Wrangler provides *natural language descriptions* and *visual transform previews*. Natural language descriptions are intended to enhance analysts' ability to review and refine transformation steps. Textual annotations enable communication of analyst intent. Wrangler also couples verification (run in the background as data is transformed) with visualization to help users discover data quality issues.

Basic Interactions

The Wrangler interface supports six basic interactions within the data table. Users can select rows, select columns, click bars in the data quality meter, select text within a cell, edit data values within the table (for mass editing [14, 19]), and assign column names, data types or semantic roles. Users can also choose transforms from the menu or refine suggestions by editing transform descriptions as described below.

Automated Transformation Suggestions

As a user interacts with data, Wrangler generates a list of *suggested transforms*. In some cases the set of possible suggestions is large (in the hundreds), but we wish to show only a relevant handful to avoid overload. Instead of enumerating the entire suggestion space, users can prune and reorder the space in three ways. First, users can provide more examples to disambiguate input to the inference engine. Providing examples is especially effective for text selections needed for splitting, extraction, and reformatting; two or three well-chosen examples typically suffice. Second, users can filter the space of transforms by selecting an operator from the transform menu. Third, users can edit a transform by altering the parameters of a transform to a desired state.

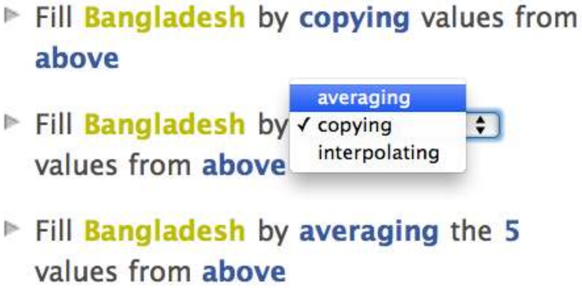
Wrangler does not immediately execute a selected suggestion. Instead, Wrangler makes it the *current working transform*. The user can edit this transform directly; as a user edits parameters, the suggestion space updates to reflect these edits. Also, a user can instead interact with the table to generate new suggestions that use the working transform as context.

Natural Language Descriptions

To aid apprehension of suggested transforms, Wrangler generates short *natural language descriptions* of the transform type and parameters. These descriptions are editable, with parameters presented as bold hyperlinks (Fig. 8). Clicking a link reveals an in-place editor appropriate to the parameter (Fig. 8b). Enumerable variables (such as the direction of a fill) are mapped to drop-down menus while free-form text parameters are mapped to text editors with autocomplete.

We designed these descriptions to be concise; default parameters that are not critical to understanding may be omitted. For example, the *unless between* parameter for split operations indicates regions of text to ignore while splitting. In most cases, this parameter is left undefined and including it would bloat the description. To edit hidden parameters, users can click the expansion arrow to the left of the description, revealing an editor with entries for all possible parameters.

We also sought to make parameters within descriptions readable by non-experts. For instance, we translate regular expressions into natural language via pattern substitution (e.g., (\d+) to 'number'). This translation can make some descriptions less concise but increases readability. Translation is only performed with regular expressions generated by the Wrangler inference engine. If a user types in a custom expression, Wrangler will reflect their input.



▸ Fill **Bangladesh** by **copying** values from **above**

▸ Fill **Bangladesh** by **copying** values from **above**

▸ Fill **Bangladesh** by **averaging** the **5** values from **above**

Figure 8. Editable Natural Language Descriptions. (a) An example of an editable description; highlighted text indicates editable parameters. (b) Clicking on a parameter reveals an in-place editor. (c) After editing, the description may update to include new parameters. In this case, a new window size parameter is displayed for the moving average.

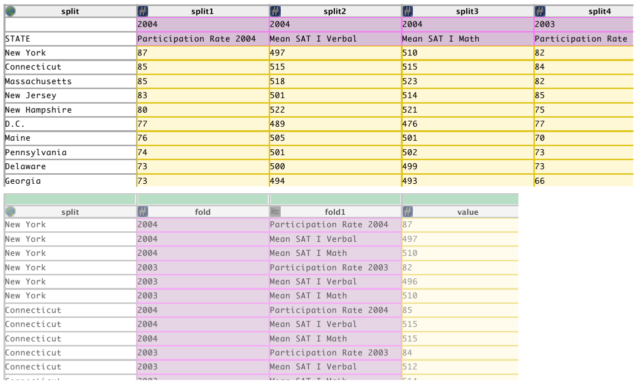
Visual Transformation Previews

Wrangler uses *visual previews* to enable users to quickly evaluate the effect of a transform. For most transforms, Wrangler displays these previews in the source data, and not as a separate visualization (e.g., side-by-side before and after views). In-place previews provide a visual economy that serves a number of goals. First, displaying two versions of a table inherently forces both versions to be small, which is particularly frustrating when the differences are sparse. Second, presenting in-place modifications draws user attention to the effect of the transformation in its original context, without requiring a shift in focus across multiple tables. As we discuss next, in-place previews better afford direct manipulation for users to revise the current transform.

Wrangler maps transforms to at least one of five preview classes: selection, deletion, update, column and table. In defining these mappings, we attempted to convey a transform's effect with minimum displacement of the original data. This stability allows users to continue interacting with the original data, e.g., to provide new selection examples.

Selection previews highlight relevant regions of text in all affected cells (Fig. 3). Deletion previews color to-be-deleted cells in red (Fig. 2). Update previews overwrite values in a column and indicate differences with yellow highlights (Fig. 4). Column previews display new derived columns, e.g., as results from an *extract* operation (Fig. 3). We show a side-by-side display of versions when previewing *fold* and *unfold* transforms. These alter the structure of the table to such an extent that the best preview is to show another table (Fig. 6, 9). These table previews use color highlights to match input data to their new locations in the output table. Some transforms map to multiple classes; e.g., *extract* transforms use both selection and column previews.

When possible, previews also indicate where the user can modify the transform through either direct manipulation or description refinement. Highlighting selected text or cells works well for certain transformations. For example, by



split	split1	split2	split3	split4
	2004	2004	2004	2003
STATE	Participation Rate 2004	Mean SAT I Verbal	Mean SAT I Math	Participation Rate
New York	87	497	510	82
Connecticut	85	515	515	84
Massachusetts	85	518	523	82
New Jersey	83	501	514	85
New Hampshire	80	522	521	75
D.C.	77	489	476	77
Maine	76	505	501	70
Pennsylvania	74	501	502	73
Delaware	73	500	499	73
Georgia	73	494	493	66

split	fold	fold1	value
New York	2004	Participation Rate 2004	87
New York	2004	Mean SAT I Verbal	497
New York	2004	Mean SAT I Math	510
New York	2003	Participation Rate 2003	82
New York	2003	Mean SAT I Verbal	496
New York	2003	Mean SAT I Math	510
Connecticut	2004	Participation Rate 2004	85
Connecticut	2004	Mean SAT I Verbal	515
Connecticut	2004	Mean SAT I Math	515
Connecticut	2003	Participation Rate 2003	84
Connecticut	2003	Mean SAT I Verbal	512

Figure 9. Visual preview of a *fold* operation. For transforms that rearrange table layout, Wrangler previews the output table and uses color highlights to show the correspondence of values across table states.

highlighting the text selected by a regular expression in each cell, users can determine which examples they need to fix. For reshape transforms, Wrangler highlights the input data in the same color as the corresponding output in the secondary table. For instance, in a *fold* operation, data values that will become keys are colored to match the keys in the output table (Fig. 9). Wrangler also highlights the parameters in the transform description using the same colors as those generated in previews (Fig. 3–6). The consistent use of colors allows users to associate clauses in a description with their effects in the table.

Transformation Histories and Export

As successive transforms are applied, Wrangler adds their descriptions to an interactive *transformation history viewer*. Users can edit individual transform descriptions and selectively enable and disable prior transforms. Upon changes, Wrangler runs the edited script and updates the data table. Toggling or editing a transform may result in downstream errors; Wrangler highlights broken transforms in red and provides an error message to aid debugging.

Wrangler scripts also support lightweight text annotations. Analysts can use annotations to document their rationale for a particular transform and may help future users better understand data provenance. To annotate a transform, users can click the *edit* icon next to the desired transform and write their annotation in the resulting text editor. Users can view an annotation by mousing over the same *edit* icon. These annotations appear as comments in code-generated scripts. Users can export both generated scripts and transformed data; clicking the *Export* button in the transform history invokes export options. Analysts can later run saved or exported scripts on new data sources, modifying the script as needed.

TYPES, ROLES, AND VERIFICATION

It is often difficult to discover data quality issues and therefore difficult to address them by constructing the appropriate transform. Wrangler aids discovery of data quality issues through the use of data types and semantic roles.

As users transform data, Wrangler attempts to infer the data type and semantic role for each column. Wrangler applies validation functions to a sample of a column's data to infer these types, assigning the type that validates for over half of the non-missing values. When multiple types satisfy this criteria, Wrangler assigns the more specific one (e.g., *integer* is more specific than *double*). Wrangler infers semantic roles analogously. An icon in the column header indicates the semantic role of the column, or the underlying data type if no role has been assigned. Clicking the icon reveals a menu with which users can manually assign a type or role.

Above each column is a *data quality meter*: a divided bar chart that indicates the proportion of values in the column that verify completely. Values that parse successfully are indicated in green; values that match the type but do not match the role (e.g., a 6 digit zip code) are shown in yellow; those that do not match the type (e.g., 'One' does not parse as an integer) are shown in red; and missing data are shown in gray. Clicking a bar generates suggested transforms for that category. For instance, clicking the *missing values* bar will suggest transforms to fill in missing values or delete those rows. Clicking the *fails role* bar will suggest transforms such as a similarity join on misspelled country names.

THE WRANGLER INFERENCE ENGINE

We now present the design of the Wrangler inference engine, which is responsible for generating a ranked list of suggested transforms. Inputs to the engine consist of user interactions; the current working transform; data descriptions such as column data types, semantic roles, and summary statistics; and a corpus of historical usage statistics. Transform suggestion proceeds in three phases: inferring transform parameters from user interactions, generating candidate transforms from inferred parameters, and finally ranking the results.

Usage Corpus and Transform Equivalence

To generate and rank transforms, Wrangler's inference engine relies on a corpus of usage statistics. The corpus consists of frequency counts of transform descriptors and initiating interactions. We built our initial corpus by wrangling our collection of gathered data sets. The corpus updates over time as more analysts use Wrangler.

For any given transform, we are unlikely to find an exact match in the corpus. For instance, an analyst may perform a *fold* operation over a combination of columns and rows that does not appear in the corpus. In order to get useful transform frequencies, we define a relaxed matching routine: two transforms are considered *equivalent* in our corpus if (a) they have an identical transform type (e.g., *extract* or *fold*) and (b) they have equivalent parameters as defined below.

Wrangler transforms accept four basic types of parameters: row, column or text selections and enumerables. We treat two row selections as equivalent if they both (a) contain filtering conditions (either index- or predicate-based) or (b) match all rows in a table. Column selections are equivalent if they refer to columns with the same data type or semantic role. We based this rule on the observation that transforms that operate on identical data types are more likely to be similar. Text selections are equivalent if both (a) are index-based selections or (b) contain regular expressions. We consider enumerable parameters equivalent only if they match exactly. We chose these equivalency classes based on exploratory analysis of our corpus and they seem to work well in practice. As our corpus of transforms grows with more use, we plan to explore more principled approaches (such as clustering) to refine our matching routines.

Inferring Parameter Sets from User Interaction

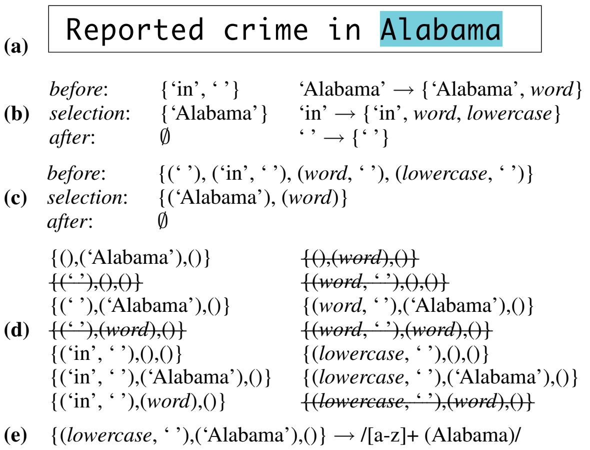
In response to user interaction, Wrangler attempts to infer three types of transform parameters: row, column, or text selections. For each type we enumerate possible parameter values, resulting in a collection of inferred parameter sets. We infer a parameter's values independent of the other parameters. For example, we infer regular expressions for text selection based solely on the selected text, a process otherwise independent of which rows or columns are selected.

We infer row selections based on row indices and predicate matching. We list predicates of the form "row is empty" and "column [equals | starts with | ends with | contains] selected-value", then emit the selections that match the rows and text currently selected in the interface. For column selections we simply return the columns that users have interacted with.

Emitted text selections are either simple index ranges (based directly on selections in the interface) or inferred regular expressions. To generate regular expressions, we tokenize the text within a cell and extract both the selected text and any surrounding text within a 5 token window. We annotate tokens with one or more labels of the form *number*, *word*, *uppercase* word, *lowercase* word, or *whitespace*. We then enumerate label sequences that match the text before, within, and after the selection range (see Fig. 10); sequences can contain either an annotation label or the exact token text. Next we emit all possible combinations of before, within, and after sequences that match all current text selection examples in the interface. It is then straightforward to translate matching label sequences into regular expressions.

Generating Suggested Transforms

After inferring parameter sets, Wrangler generates a list of transform suggestions. For each parameter set, we loop over each transform type in the language, emitting the types that can accept all parameters in the set. For example, a *split* transform can accept a parameter set containing a text selection, but an *unfold* transform can not. Wrangler instantiates each emitted transform with parameters from the parameter set. To determine values for missing parameters, we query the corpus for the top-k (default 4) parameterizations that co-occur most frequently with the provided parameter set. During this process we do not infer complex criteria such as row predicates or regular expressions; we do infer enumerable parameters, index-based row selections, and column inputs. We then filter the suggestion set to remove "degenerate" (no-op) transforms that would have no effect on the data.



(a) Reported crime in Alabama

(b)

before:	{'in', ' '}	'Alabama' → {'Alabama', *word*}
selection:	{'Alabama'}	'in' → {'in', *word*, *lowercase*}
after:	∅	' ' → {' '}

(c)

before:	{(' '), ('in', ' '), (*word*, ' '), (*lowercase*, ' ')}
selection:	{('Alabama'), (*word*)}
after:	∅

(d)

{(),('Alabama'),()}	~~{(),(*word*),()}~~
~~{(' '),(),()}~~	~~{(*word*, ' '),(),()}~~
{(' '),('Alabama'),()}	{(*word*, ' '),('Alabama'),()}
~~{(' '),(*word*),()}~~	~~{(*word*, ' '),(*word*),()}~~
{('in', ' '),(),()}	{(*lowercase*, ' '),(),()}
{('in', ' '),('Alabama'),()}	{(*lowercase*, ' '),('Alabama'),()}
{('in', ' '),(*word*),()}	~~{(*lowercase*, ' '),(*word*),()}~~

(e) {(*lowercase*, ' '),('Alabama'),()} → /[a-z]+ (Alabama)/

Figure 10. Regular Expression Inference. (a) The user selects text in a cell. (b) We tokenize selected and surrounding text. For clarity, the figure only includes two neighboring tokens. For each token, we generate a set of matching labels. (c) We enumerate all label sequences matching the text. (d) We then enumerate all candidate *before*, *selection* and *after* combinations. Patterns that do not uniquely match the selected text are filtered (indicated by strike-through). (e) Finally, we construct regular expressions for each candidate pattern.

Ranking Suggested Transforms

Wrangler then rank-orders transform suggestions according to five criteria. The first three criteria rank transforms by their type; the remaining two rank transforms within types. Ensuring that transforms of the same type are adjacent helps users compare varying parameterizations more easily.

First, we consider explicit interactions: if a user chooses a transform from the menu or selects a current working transform, we assign higher rank to transforms of that type. Second, we consider specification difficulty. We have observed that row and text selection predicates are harder to specify than other parameters. We thus label row and text selections as *hard* and all others as *easy*. We then sort transform types according to the count of *hard* parameters they can accept. Third, we rank transform types based on their corpus frequency, conditioned on their initiating user interaction (e.g., text or column selection). In the case of text selection, we also consider the length of the selected text. If a user selects three or fewer characters, *split* transforms are ranked above *extract* transforms; the opposite is true for longer selections.

We then sort transforms within type. We first sort transforms by frequency of *equivalent* transforms in the corpus. Second, we sort transforms in ascending order using a simple measure of transform *complexity*. Our goal is to preferentially rank simpler transforms because users can evaluate their descriptions more quickly. We define transform complexity as the sum of complexity scores for each parameter. The complexity of a row selection predicate is the number of clauses it contains (e.g., "a=5 and b=6" has complexity 2). The complexity of a regular expression is defined to be the number of tokens (described previously) in its description. All other parameters are given complexity scores of zero.

Finally, we attempt to surface diverse transform types in the final suggestion list. We filter the transforms so that no type accounts for more than 1/3 of the suggestions, unless the transform type matches the working transform or the filter results in fewer suggestions than can appear in the interface.

COMPARATIVE EVALUATION WITH EXCEL

As an initial evaluation of Wrangler, we conducted a comparative user study with Microsoft Excel. Subjects performed three common data cleaning tasks: value extraction, missing value imputation, and table reshaping. Our goal was to compare task completion times and observe data cleaning strategies. We chose Excel because it is the most popular data manipulation tool and provides an ecologically valid baseline for comparison: all subjects use it regularly and half self-report as experts. Excel also supports our chosen tasks. Neither Potter's Wheel [22] (no support for fill) nor Google Refine [13] (lack of reshaping) support the full set. In contrast, Excel includes specific tools for each task (text-to-columns, goto-special & pivot tables) in addition to manual editing.

Participants and Methods

We recruited 12 participants, all professional analysts or graduate students who regularly work with data. Subjects rated their prior experience with Excel on a 10-point scale (1 being basic knowledge and 10 being expert); the median score was 5. Participants had never used the Wrangler interface.

We first presented a 10 minute Wrangler tutorial describing how to create, edit, and execute transforms. We then asked subjects to complete three tasks (described below) using both Wrangler and Excel. We randomized the presentation of tasks and tools across subjects. In each task, we asked subjects to transform a data set into a new format, presented to them as a picture of the final data table.

Task 1: Extract Text. In this task, we asked users to extract the number of bedrooms and housing price from housing listings on craigslist. The original data set contained one cell for each listing, with all the information in a text string. The target data set consisted of two columns: one for the number of bedrooms and one for the housing price.

Task 2: Fill Missing Values. We gave users data containing year-by-year agricultural data for three countries. Some of the values in the data set were blank. The target data set contained the same data with all missing values replaced with the closest non-empty value from a previous year.[1]

Task 3: Reshape Table Structure. Users started with three columns of housing data: year, month, and price. The target data set contained the same data formatted as a cross-tab: the data contained one row for each year, with the 12 months as column headers and housing prices as cell values.

When using Excel, we allowed subjects to ask for references to functions they could describe concretely (e.g., we would answer *"how do I split a cell?"* but not *"how do I get the number of bedrooms out?"*). For Wrangler tasks, we did not respond to user inquiries. We permitted a maximum of 10 minutes per task. Each data set had at most 30 rows and 4 columns; complete manual manipulation in Excel was easily attainable within the time limits. Afterwards, each user completed a post-study questionnaire.

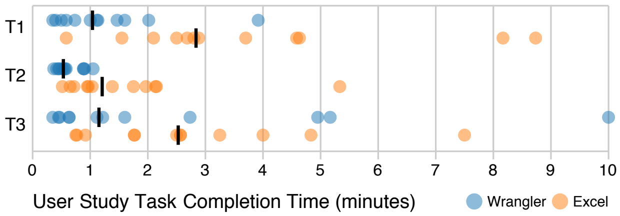


Figure 11. Task completion times. Black bars indicate median values. Median Wrangler performance is over twice as fast in all tasks.

Wrangler Accelerates Transform Specification

We performed a repeated-measures ANOVA of completion times with task, tool, and Excel novice/expert[2] as independent factors; we log-transformed responses to better approximate a normal distribution. We found a significant main effect of tool ($F_{1,54} = 23.65$, $p < 0.001$), but no main effect of task ($F_{1,54} = 0.01$, $p = 0.943$) or expertise ($F_{1,54} = 0.30$, $p = 0.596$). We found a significant interaction effect of task and expertise ($F_{1,54} = 11.10$, $p < 0.002$) driven by improved performance by experts (regardless of tool) in the reshaping task (T3). No other interactions were significant.

Across all tasks, median performance in Wrangler was over twice as fast as Excel (Fig. 11). Users completed the cleaning tasks significantly more quickly with Wrangler than with Excel, and this speed-up benefitted novice and expert Excel users alike. Moreover, the user study tasks involved small data sets amenable to manual manipulation. As data set size grows, we expect the benefits of Wrangler to come into even sharper relief. Of course, larger data sets might complicate the process of assessing transform effects and so may benefit from additional validation and visualization techniques.

Strategies for Navigating Suggestion Space

When working with Wrangler, users applied different navigation strategies for different tasks. These strategies were largely consistent across users. For text selection, users frequently provided multiple examples. For other operations, users performed an initial selection and then previewed each suggestion. One subject noted, *"I just look at the picture."* Users with a programming background spent time reading transform descriptions, whereas the other users relied almost entirely on the previews. When users did not find a transform among the initial suggestions, they most often filtered the suggestions by selecting a transform type from the menu. If only imperfect matches were found, users then selected the nearest transform and edited its parameters. In other words, users turned to manual parameterization only as a last resort.

Our post-study questionnaire asked users to rate automated suggestions, visual previews, and direct editing of transforms on a scale from 1 (not useful) to 5 (most useful). We performed an ANOVA and found a significant difference among the ratings ($F_{2,33} = 17.33$, $p < 0.001$). Users rated previews ($\mu = 4.8$) and suggestions ($\mu = 4.3$) significantly more useful than direct editing ($\mu = 2.5$) ($p < 0.001$ in both cases by

[1] We acknowledge that this is not an ideal cleaning solution for the data, but it nonetheless served as a useful test.

[2] We divided subjects into "novices" and "experts" according to their median self-reported expertise rating (5).

Tukey's HSD). Users' preference for suggestions and previews over direct editing provides evidence that these novel user interface features have merit.

Users' navigation strategies worked well when they understood the nature of the desired transform, even if they did not know how to specify it. However, we found that users of both tools experienced difficulty when they lacked a conceptual model of the transform. For instance, Task 3 exhibited an uneven distribution of completion times; 7 of the 10 fastest times and 3 of the 4 slowest times were in Wrangler. Wrangler does not provide the recourse of manual editing, hence users who got stuck fared slightly better in Excel. However, those familiar with pivot operations in Excel uniformly performed the task more quickly with Wrangler.

We also observed one recurring pitfall: a few users got stuck in a "cul-de-sac" of suggestion space by incorrectly filtering (e.g., by selecting a specific transform type from the menu). When this happened, some users kept searching and refining only these filtered transforms. By design, Wrangler does not afford users the same flexibility to layout data as in Excel; since users cannot perform arbitrary editing in Wrangler, the recourse is less obvious when they get stuck. This pitfall was most common in Task 3, where a user might mistakenly filter all but *fold* operations when an *unfold* operation was needed. One solution may be to suggest non-matching transforms related to the selected transform type, in effect treating filtering criteria as guidelines rather than strict rules.

CONCLUSION AND FUTURE WORK

This paper introduced Wrangler, an interface and underlying language for data transformation. The system provides a mixed-initiative interface that maps user interactions to suggested data transforms and presents natural language descriptions and visual transform previews to help assess each suggestion. With this set of techniques, we find that users can rapidly navigate to a desired transform.

Our user study demonstrates that novice Wrangler users can perform data cleaning tasks significantly faster than in Excel, an effect shared across both novice and expert Excel users. We found that users are comfortable switching navigation strategies in Wrangler to suit a specific task, but can sometimes get stuck—in either tool—if they are unfamiliar with the available transforms. Future work should help users form data cleaning strategies, perhaps through improved tutorials.

Looking forward, Wrangler addresses only a subset of the hurdles faced by data analysts. As data processing has become more sophisticated, there has been little progress on improving the tedious parts of the pipeline: data entry, data (re)formatting, data cleaning, etc. The result is that people with highly specialized skills (e.g., statistics, molecular biology, micro-economics) spend more time in tedious "wrangling" tasks than they do in exercising their specialty, while less technical audiences such as journalists are unnecessarily shut out. We believe that more research integrating methods from HCI, visualization, databases, and statistics can play a vital role in making data more accessible and informative.

ACKNOWLEDGEMENTS

The first author was supported by a Stanford Graduate Fellowship. We also thank the Boeing Company, Greenplum and Lightspeed Venture Partners for their support. This work was partially funded by NSF Grant CCF-0963922.

REFERENCES

1. A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *ACM SIGMOD*, pages 337–348, 2003.
2. A. F. Blackwell. SWYN: A visual representation for regular expressions. In *Your Wish is my Command: Programming by Example*, pages 245–270, 2001.
3. L. Chiticariu, P. G. Kolaitis, and L. Popa. Interactive generation of integrated schemas. In *ACM SIGMOD*, pages 833–846, 2008.
4. T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, Inc., New York, NY, 2003.
5. T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *ACM SIGMOD*, pages 240–251, 2002.
6. A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE TKDE*, 19(1):1–16, 2007.
7. K. Fisher and R. Gruber. Pads: a domain-specific language for processing ad hoc data. In *ACM PLDI*, pages 295–304, 2005.
8. H. Galhardas, D. Florescu, D. Shasha, and E. Simon. Ajax: an extensible data cleaning tool. In *ACM SIGMOD*, page 590, 2000.
9. L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio grows up: from research prototype to industrial tool. In *ACM SIGMOD*, pages 805–810, 2005.
10. J. M. Hellerstein. Quantitative data cleaning for large databases, 2008. White Paper, United Nations Economic Commission for Europe.
11. V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artif. Intell. Rev.*, 22(2):85–126, 2004.
12. E. Horvitz. Principles of mixed-initiative user interfaces. In *ACM CHI*, pages 159–166, 1999.
13. D. Huynh and S. Mazzocchi. Google Refine. `http://code.google.com/p/google-refine/`.
14. D. F. Huynh, R. C. Miller, and D. R. Karger. Potluck: semi-ontology alignment for casual users. In *ISWC*, pages 903–910, 2007.
15. Z. G. Ives, C. A. Knoblock, S. Minton, M. Jacob, P. Pratim, T. R. Tuchinda, J. Luis, A. Maria, and M. C. Gazen. Interactive data integration through smart copy & paste. In *CIDR*, 2009.
16. H. Kang, L. Getoor, B. Shneiderman, M. Bilgic, and L. Licamele. Interactive entity resolution in relational data: A visual analytic tool and its evaluation. *IEEE TVCG*, 14(5):999–1014, 2008.
17. L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. SchemaSQL: An extension to SQL for multidatabase interoperability. *ACM Trans. Database Syst.*, 26(4):476–519, 2001.
18. J. Lin, J. Wong, J. Nichols, A. Cypher, and T. A. Lau. End-user programming of mashups with vegemite. In *IUI*, pages 97–106, 2009.
19. R. C. Miller and B. A. Myers. Interactive simultaneous editing of multiple text regions. In *USENIX Tech. Conf.*, pages 161–174, 2001.
20. D. A. Norman. *The Design of Everyday Things*. Basic Books, 2002.
21. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350, 2001.
22. V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, pages 381–390, 2001.
23. G. G. Robertson, M. P. Czerwinski, and J. E. Churchill. Visualization of mappings between schemas. In *ACM CHI*, pages 431–439, 2005.
24. C. Scaffidi, B. Myers, and M. Shaw. Intelligently creating and recommending reusable reformatting rules. In *ACM IUI*, pages 297–306, 2009.
25. S. Soderland. Learning information extraction rules for semi-structured and free text. *Mach. Learn.*, 34(1-3):233–272, 1999.
26. R. Tuchinda, P. Szekely, and C. A. Knoblock. Building mashups by example. In *ACM IUI*, pages 139–148, 2008.