Towards More Transparent, Reproducible, and Reusable Data Cleaning with OpenRefine*

Lan Li[1], Bertram Ludäscher[1], and Qian Zhang[2]

[1] School of Information Sciences, University of Illinois at Urbana-Champaign, USA
{lanl2,ludaesch}@illinois.edu
[2] University of Waterloo, 200 University Ave W, Waterloo ON N2L 3G1, Canada
q394zhan@uwaterloo.ca

Abstract. We study provenance features of OpenRefine, a popular data cleaning tool. In OpenRefine, provenance is available through *operation histories* and *recipes*. The former provide users with an undo/redo capability; the latter represent histories in JSON, so recipes can be reused. The model implicit in histories and recipes exhibits both *prospective* and *retrospective* provenance features, but is incomplete in at least two ways: (i) functions resulting in mass edits, and (ii) single cell edits are not captured, thus missing important prospective and retrospective provenance information, respectively. We propose to complete the missing information by capturing names and parameters of user-invoked functions, and by exposing retrospective provenance hidden in internal project files. The feasibility of the approach is demonstrated with an early prototype.

Keywords: OpenRefine · data cleaning · provenance · transparency · reproducibility · reusability

1 Introduction

OpenRefine [2] is a popular data cleaning tool that lets users explore, profile, and edit datasets in a browser-based, spreadsheet-like GUI. In particular, users can execute various transformations on columns, e.g., trimming whitespaces, changing case, converting formats and data types, etc. To create canonical names or values in the presence of variant spellings and representations, OpenRefine employs powerful clustering methods (*key collision* and *nearest neighbor*), parameterized by *keying* or *distance functions*.

Consider an output dataset $D' = W(D)$ resulting from the application of a user-defined data cleaning workflow W on an input dataset D (often a CSV file). The goal of W is that the output D' has better *data quality* (is "cleaner") than the input D and that D' is "fit for use" [5,4].

While working on a data cleaning project, a user's linear *operation history* H provides a high-level, human-readable overview of the sequence of steps performed and can be used to undo and redo operations. In addition, the user can

* Work supported in part by the National Science Foundation, award OAC #1541450.

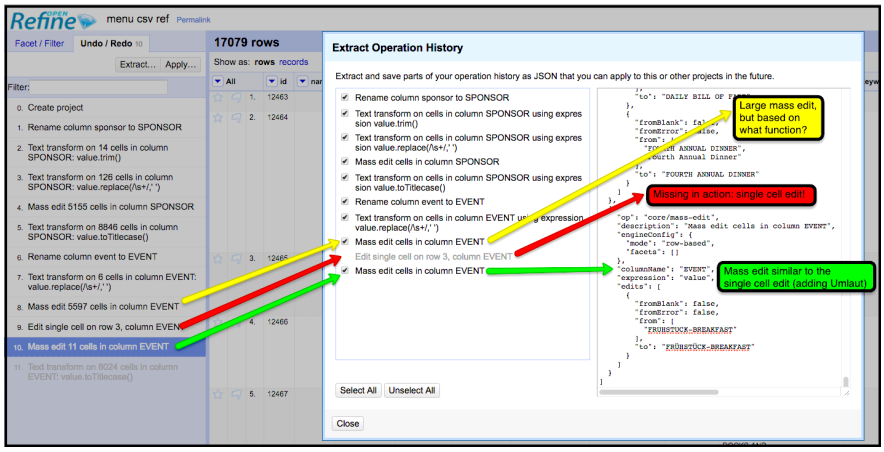

Fig. 1. From Steps 1–10 of the history H (background, left) the user can extract a machine-readable JSON recipe R (bright foreground, right). The *mass edit* in Step 8 is captured via a set of rules $\{\texttt{from}{:}old \rightarrow \texttt{to}{:}new\}$, but it is *not* recorded which function was used to create these rules. The *single cell edit* in Step 9 is greyed out and cannot be selected by the user for inclusion in R. Neither H nor R indicate what this edit was. In contrast, the mass edit in Step 10 is included in the recipe R: it is fully described by the rule `from:FRUHSTUCK-BREAKFAST` $\rightarrow$ `FRÜHSTÜCK-BREAKFAST` that has been (and can be, prospectively) applied to all values in the EVENT column.

extract all or parts of the operation history to obtain a *recipe* R in machine-readable JSON format. This JSON recipe R can then be *reused* on datasets similar to D. The history H and its companion recipe R thus provide valuable *provenance* information about W. The following are important requirements or desiderata for workflows and provenance artifacts:

- *Transparency.* A skeptical data cleaning customer should be able to explain how D' was obtained from D, i.e., what operations were performed on what parts of D and in what order. This is the classical domain of provenance.
- *Reproducibility.* Given D and a description of W, a skeptic should be able to devise their own implementation I_W such that $I_W(D)$ also yields D'.
- *Reusability.* It should be "easy" to apply I_W to other datasets E that are "similar enough" to D to obtain a "cleaner" E'. This notion is necessarily vague, given that the underlying notions have to be prescribed first.

2 Provenance and OpenRefine

We distinguish *prospective provenance* which is the specification of a workflow W or at least of W's dataflow dependencies, and *retrospective provenance* which records runtime observables used to instantiate W. In OpenRefine the user workflow W is partially captured in the operation history H and the associated

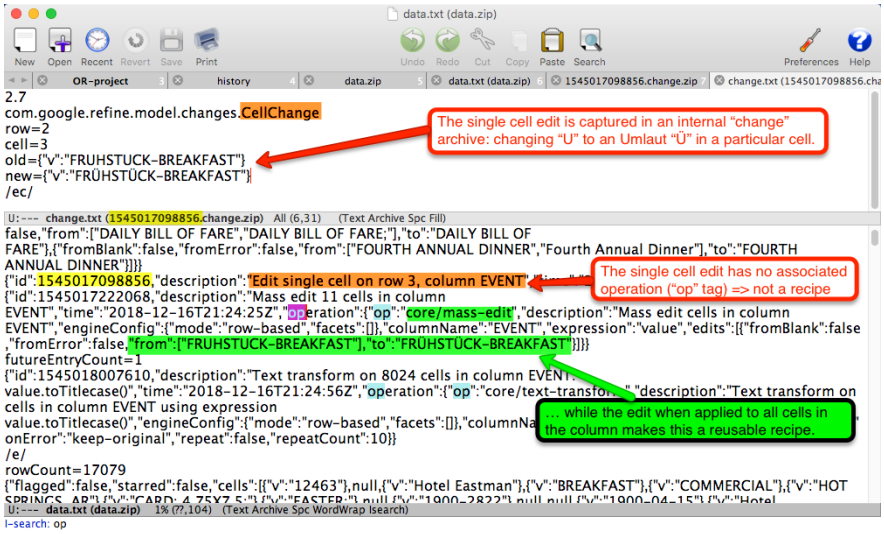

Fig. 2. OpenRefine project files reveal missing retrospective provenance: the `data` zip-archive contains a reference to a single-cell edit (center left); the individual update is recorded in a `change` archive in a history folder (top).

recipe R, which can thus serve as an implementation I_W. However, as illustrated in Figure 1, R is *incomplete*: If the user chose a *Cluster & Edit* method, e.g., *key collision* with n-gram *fingerprint* function for $n = 3$, then R only captures rules $\{\texttt{from}{:}old \rightarrow \texttt{to}{:}new\}$, but not the method, function, or parameters. Thus prospective provenance is lost and reusability diminished. If the user performs a *single cell edit*, say fixing umlauts in a particular cell, then this "one-off" is not considered a reusable operation and (understandably) omitted from the recipe R. However an important piece of retrospective provenance is lost this way, reducing transparency and preventing reproducibility of W via R.

3 Approach and Demonstration

We propose to enhance the native OpenRefine recipes as follows: when users select operations such as *Cluster & Edit*, the system should also capture *prospective* information such as the clustering method with associated functions and parameters. Missing *retrospective* information, e.g., from single cell edits should be extracted from internal OpenRefine files and made explicit (Figure 2).

To demonstrate our approach we have developed CLOPER (Command-Line OpenRefine Prototype for Enhanced Recipes) and ER3 (Enhanced Recipe Re-Runner) [3], both of which use a Python client library [1] to communicate with the OpenRefine server. CLOPER can execute OpenRefine operations via a command-line driven interface (i.e., without a GUI) and adds missing prospective information to recipes (Figure 3). ER3 can then run those enhanced recipes and

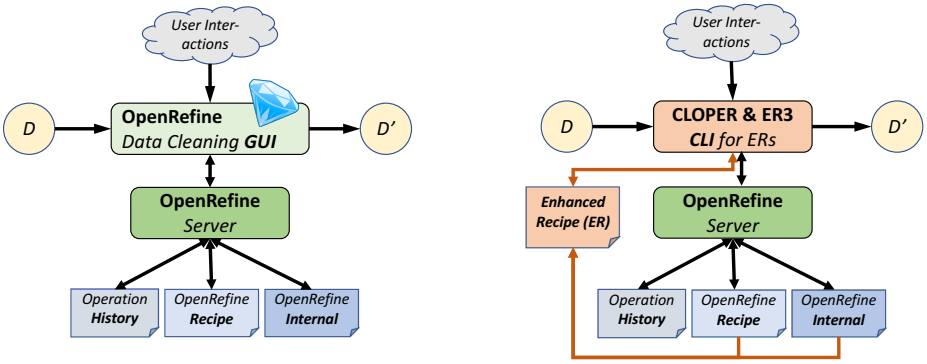

Fig. 3. Driven by user interactions in the OpenRefine (OR) GUI, input dataset D is transformed (cleaned) to obtain D'. The OR server captures provenance information in a user-readable history H, machine-readble JSON recipe R, and internal project files. CLOPER is used to capture additional provenance in an enhanced recipe (ER). The ER Re-Runner (ER3) can execute native recipes and enhanced recipes

demonstrate the improved transparency, reproducibility, and reusability. Further extensions to CLOPER and ER3 to add additional prospective and retrospective provenance are under development.

References

1. Makepeace, P., Lohmeier, F.: OpenRefine Python client library. `https://github.com/openculturecon sulting/openrefine-client` (2018)
2. OpenRefine: A free, open source, powerful tool for working with messy data. `http://openrefine.org/` (2018)
3. OpenRefine provenance tools: CLI for OpenRefine and Enhanced Recipes Re-Runner CLOPER & ER3. https://github.com/idaks/OpenRefine-Provenance-Tools (2018)
4. Sadiq, S. (ed.): Handbook of Data Quality. Springer (2013)
5. Wang, R.W., Strong, D.M.: Beyond Accuracy: What Data Quality Means to Data Consumers. Journal of Management Information Systems **12**(4), 5 (1996)