TableCanoniser: Interactive Grammar-Powered Transformation of Messy, Non-Relational Tables to Canonical Tables

Kai Xiong
State Key Lab of CAD&CG
Zhejiang University
Hangzhou, Zhejiang, China
kaixiong@zju.edu.cn

Cynthia A Huang
Department of Econometrics and Business Statistics
Monash University
Melbourne, Victoria, Australia
cynthia.huang@monash.edu

Michael Wybrow*
Department of Human-Centred Computing
Monash University
Melbourne, Victoria, Australia
michael.wybrow@monash.edu

Yingcai Wu
State Key Lab of CAD&CG
Zhejiang University
Hangzhou, Zhejiang, China
ycwu@zju.edu.cn

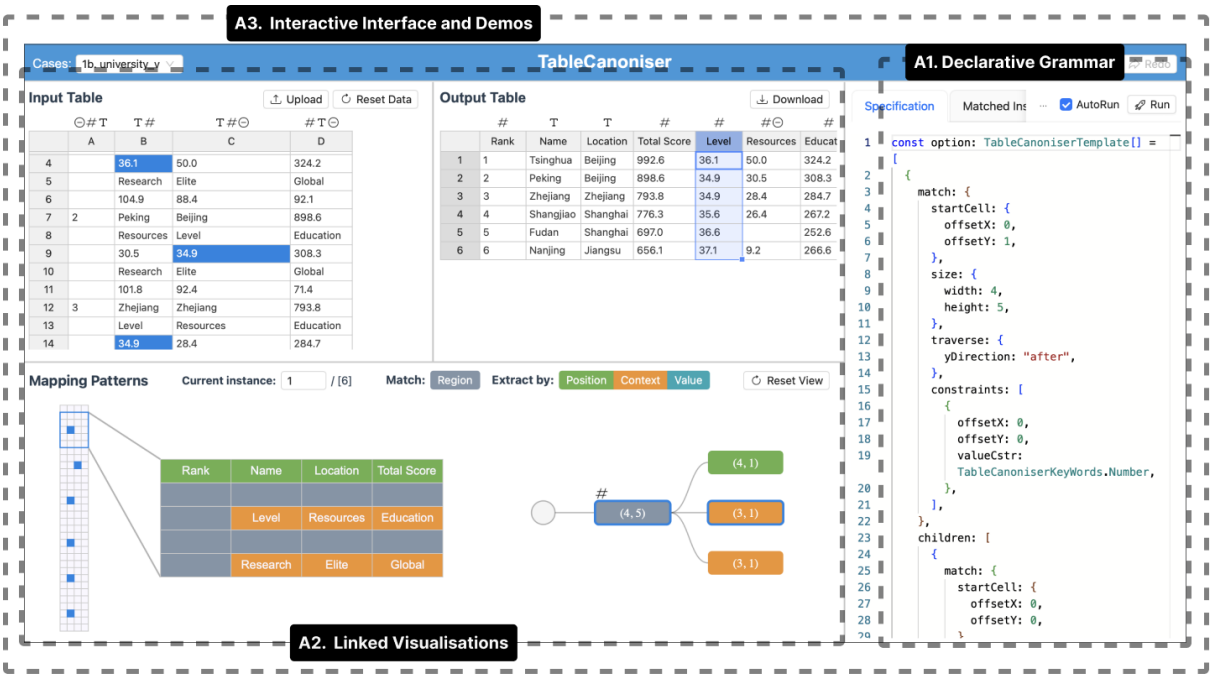


Figure 1: The TableCanoniser system provides integrated support for obtaining relational tables from messy tabular inputs using [A1] a declarative grammar for specifying patterns to match and extract values, [A2] linked visualisations which allow users to construct the specification, then examine and verify transformation inputs and outputs, wrapped in [A3], an interactive application with preloaded examples to reduce the learning curve for new users.

Abstract

TableCanoniser is a declarative grammar and interactive system for constructing relational tables from messy tabular inputs such as spreadsheets. We propose the concept of *axis alignment* to categorise input types and characterise the expanded scope of our system relative to existing tools. The declarative grammar consists of match conditions, which specify repeating patterns of input cells, and extract operations, which specify how matched values map to the output table. In the interactive interface, users can specify match and extract patterns by interacting with an input table, or author more advanced specifications in the coding panel. To refine and verify specifications, users interact with grammar-based provenance visualisations such as linked highlighting of input and

*Michael Wybrow is the corresponding author.

CHI '25, Yokohama, Japan

ACM ISBN 979-8-4007-1394-1/25/04
https://doi.org/10.1145/3706598.3714321

output values, tree-based visualisation of matching patterns, and a mini-map overview of matched instances of patterns with annotations showing where cells are extracted to. We motivate and illustrate our work with real-world usage scenarios and workflows.

CCS Concepts

• **Human-centered computing** → **Graphical user interfaces**; **Information visualization**; • **Information systems** → **Information integration**; • **Applied computing** → *Spreadsheets.*

Keywords

data transformation, data provenance, table canonicalisation, table understanding, declarative grammar, interactive visualisation

ACM Reference Format:
Kai Xiong, Cynthia A Huang, Michael Wybrow, and Yingcai Wu. 2025. TableCanoniser: Interactive Grammar-Powered Transformation of Messy, Non-Relational Tables to Canonical Tables. In *CHI Conference on Human Factors in Computing Systems (CHI '25), April 26–May 01, 2025, Yokohama, Japan.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3706598.3714321

1 Introduction

Spreadsheets are ubiquitous sources of valuable data. However, they often require substantial processing and wrangling before the embedded data can be used in downstream data analysis tasks. Most data science workflows require inputs that follow structural rules of relational tables: entities or observations as rows, attributes or variables as columns, and a single type of entity or observation across all rows. However, spreadsheets often violate these rules, conforming only to being semi-structured representation of values on a grid. These semi-structured grids may contain table content in the form of values, context and metadata such as titles and captions, as well as formatting and style information.

The task of retrieving relational data from grids of values, known as *Table Canonicalisation* [35], is a well-studied task, but existing solutions are limited in both task scope and verifiability. We propose a new principle for categorising patterns of content in messy tables to characterise limitations of existing tools. We define *axis-aligned* content as patterns of entity and/or attribute values that are aligned along column or row axes. Alignment corresponds to ease of indexing by column or row numbers. By definition, this includes relational tables, but also includes a subset of messy table patterns such as columns containing multiple variables separated by a delimiter, variables in column headings, or multiple entity types in the same table. *Non-aligned* content is any other pattern of an observation), dictionary style presentation (i.e., name-value pairs) or even variable content size within known boundary patterns (e.g., financial statements).

Existing interactive data wrangling tools often focus on handling inputs with easy-to-index aligned content, with limited support for non-aligned patterns. Easy-to-index transformations include applying the same transformation function across all rows in a single column, or restructuring tables with aligned content using operations indexed by column or key-value pairs [e.g., 21, 23, 45]. Row-wise operations are commonly referred to as "row-to-row" transformations, while table restructuring is also known as "table-to-table" transformation [27]. Support in existing tools for transforming non-aligned content is often the by-product of attempting to handle specific patterns from tables "in the wild", rather than intentional support for wrangling non-aligned tables [e.g., 17, 27, 36]. Moreover, tools that are sufficiently flexible to handle arbitrary non-alignment often—implicitly or explicitly [e.g., 11]—treat table cell values as separate atoms of content to be rearranged into canonical tables. Unfortunately, this atomisation often involves loss of context provided by the table structure, and creates issues with provenance and verification where the only record of transformation are scripts or interactive logs. Such artefacts are often are difficult to comprehend and audit.

The complexity and variety of "table-to-table" transformations, and mismatching dimensions of input and output tables makes verifying the "correctness" of table canonicalisation programs significantly more difficult than confirming "row-to-row" transformations. For "row-to-row" transformations, comparison of row-wise inputs and outputs may be sufficient to assure a user that the program or system is completing their desired data wrangling operations. However, depending on the structure and content of messy input tables, and the users' end goals, it might be possible to produce multiple acceptable canonical tables with different attributes or entities from a given source. The possibility of multiple acceptable outputs further complicates verification.

The prioritisation of verification support distinguishes our work from existing attempts at complete end-to-end extraction of relational information from multiple table sources at scale into knowledge graph structures [i.e., *Table Understanding* as defined by 35]. Such work has related but distinct goals compared to correctly extracting relational data for specific research or analysis tasks. In the former case, the need for automation is more acute, and there may be a greater tolerance for small inconsistencies in the extraction process. However, in the latter case, documentation and verification of transformations operations is integral to the provenance and valid analysis of extracted data. The need for transparency and provenance is clear regardless of how the transformation operations are implemented, and especially important when using automated systems or generative models to author and implement transformations. Furthermore, existing automated table processing tools also have limited affordances for interactive authoring, refinement and verification of table manipulation operations.

To provide support for non-axis aligned inputs and extend the verifiability of table canonicalisation processes, we propose a new declarative grammar and interactive system, **TableCanoniser**. The system facilitates the construction of canonical tables from messy tabular content frequently found in spreadsheets and similar grid structures. Our system provides an easy to use and highly transparent interface for users to author and verify transformation specifications based on patterns in the messy tables. We directly support transparency and verification needs by tightly coupling our proposed "match-and- extract" transformation grammar with interface choices and visualisations to support the verification of transformation semantics.

We proceed by reviewing key background concepts and definitions from existing literature on messy tables in section 2. In particular, we review task description and abstractions for messy

table processing, and definitions and taxonomies of table types relevant for understanding the contribution of our work. Building upon this background, we begin section 3 by formalising the concept of content alignment for categorising messy data, and using it to characterise the scope of TableCanoniser relative to existing data wrangling tools. We then discuss some demo cases and formative experiments which informed and motivated our work, and provide an overview of the system. In sections 4–6 we describe the declarative grammar, visual encoding and interactive interface components of the TableCanoniser system, and walk through selected workflows and usage scenarios. Finally, section 7 offers critical reflections on the strengths, limitations and implications of this work.

2 Background

2.1 Spreadsheet Data Wrangling and Table Authoring

The task of constructing relational tables from non-relational inputs is addressed to various extents by both data wrangling tools, and table understanding approaches. In general, data wrangling encompasses a broad range of inputs beyond structured tables and grids including unstructured (e.g., text) and semi-structured formats (e.g., hierarchical structures and graphs). However, most existing tools for grid-like inputs often focus on manipulations of "table content", which generally refers to the more narrow scope of cell values and their arrangement along row/column axes, ignoring the style channels.

Existing tools include interactive platforms such as Tableau [40], Trifacta [41], and OpenRefine [19]; programmatic libraries including Pandas [32], Tidyr and other Tidyverse packages [47, 48]; and programming-by-example approaches which support partial specification of transformations by pattern (e.g., Wrangler [23] and Potters Wheel [33]), or inference of transformation rules based on input-output pairs (e.g., [13], Foofah [21] and FlashRelate [1]). These existing solutions generally rely on index-based operations (e.g., split, sort, merge) for data transformation, reformatting, and cleaning. However, as further discussed in subsection 3.3, meaningful extraction patterns are not always parameterisable with row or column indices.

Table authoring tools deal with the inverse task of creating presentation ready tables from well-formed input tables [e.g., 16, 20, 44]. The distinction between table authoring and data wrangling tools is not always clear. Both types of tools facilitate layout or structure transformations (cross-tabulation, merging row/column headings etc.), though data wrangling tools tend not to offer style (font, colour, spacing, etc.) transformations and generally limit output to relational tables. Interactive "table-to-table" wrangling tools tend to blur the distinction between authoring and wrangling more than "row-to-row" tools, by supporting the transformation of various relational and non-relational tables into other tabular, but often non-relational, formats [e.g., 7, 26].

2.2 Table Understanding and Canonicalisation

Our work covers multiple aspects of Table Understanding as described by Shigarov [35]. Our proposed distinction between aligned and non-aligned table content and declarative transformation grammar both contribute efforts in to *Table Extraction* and *Table Canonicalisation*. This includes efforts to analyse, describe and categorise types of tables and operations on tables [e.g., 21, 25, 26, 44, and references therein]. Table Canonicalisation aims at "inferring the *canonical form* of a table" [35], which satisfies the conditions of a *relational database table*. We deviate from the assumption of a singular canonical form in our conceptualisation to support the construction of multiple canonical tables from the same input source. In this manner, our work shares similarities with *Spreadsheet Table Extraction* approaches, which address the understanding and extraction of relational data from grid-like inputs, using a variety of rule-based and inferential methods [e.g., 1, 36].

In contrast to the interactive and/or declarative tools discussed in the previous section, *Table Understanding* often attempts to leverage file and table semantics, syntax and occasionally style, to extract relational information in end-to-end automated processes with less focus on interactive affordances [35]. In their extensive overview of existing attempts to extract relational data from grid-like sources, Bonfitto et al. [3] outline a number of existing rule-based and automated approaches to spreadsheet table extraction. Notably, TranSheet [17] and TabbyXL [36] also introduce domain specific languages for specifying the construction of a relational table from cells or regions in an existing Microsoft Excel spreadsheets. TranSheet implements a formula based syntax for extracting relational data from `.xlsx` files with an Excel plugin providing an interactive specification interface; while TabbyXL supports the Cells Rule Language (CRL) proposed in earlier work by the authors [37]. However, neither approaches support the declarative specification or interactive provenance exploration provided by our system.

2.3 Workflow Provenance Visualisations and AI-assisted Data Analysis

As noted in various discussions on reproducibility and replicability [e.g., 39, 52], the ability to validate, comprehend and vary data preparation decisions and steps is vital for understanding the stability and generalisability of analysis outcomes, as well as assessing the suitability of a dataset for specific analyses. Our work contributes to attempts to encourage and support the documentation and visualisation of provenance information from data analysis workflows, which was most recently surveyed by Xu et al. [51]. This includes direct visualisation of data analysis operations as graph structures [e.g., 6], table comparison tools, [e.g., 31, 43], or annotation and examination of data wrangling scripts [e.g., 28, 49, 50]. Unlike our integrated grammar-driven approach, most of these existing tools visualise the code and/or inputs and outputs of predefined data wrangling operations offered by separately developed libraries. However, some similar attempts to model, implement and then capture semantically relevant provenance artefacts for visualisation and verification do exist [e.g., 4, 15].

Our work also shares similarities with recent attempts from Human Centred Explainable AI (HCXAI) to understand and support verification needs in AI-assistants [10, 12, 24]. Gu et al. [12] design a interactive probe with a chat window, generated code, original, intermediate and final execution outputs (i.e., data tables) to understand the verification workflows of analysts conducting AI-assisted

data analysis. They observe that analysts attempted to verify correctness by direct examination of AI-generated explanations and code, and also by evaluating generated outputs against their expectations. TableCanoniser simplifies the direct examination of code by carefully aligning the specification syntax to the semantics of table canonicalisation operations and providing accompanying interactive visualisations of specifications. We also support data-oriented verification through linked highlighting between inputs, outputs and the specification.

2.4 Definitions and Taxonomies of Tables

The inputs to table manipulation tools are often referred to as "tables". However, depending on context, *table* can refer to a variety of grid-like structures from arbitrary spreadsheets to well-formatted presentation tables. In the context of interactive and code-based table authoring tools, *table* can refer to both standardised structures such as relational table inputs, as well as highly structured and formatted grids of content (i.e., presentation tables), which form the outputs of such systems.

Kostyleva et al. [25] and Shigarov [35] discuss in detail existing attempts to describe the possible space of tables using taxonomies of table types, along with grammars for decomposing tables into functional parts. These attempts include categorisations based on patterns in existing corpora of table-like structures "from the wild", as well as groupings based on layout and style features. *Tidy*, *relational* and *genuine* are just some of the terms [c.f. 25, 46] used to refer to canonical table formats expected or required in data science and information management workflows. Such tables are defined directly by their properties. whilst table-like structures that do not conform to these properties are variously referred to as *messy*, *non-relational*, *non-genuine* or *arbitrary*. Applying this distinction to the more limited scope of table content as defined above, we refer to tables that do not meet the definition of relational tables as ***messy tables***. We propose the concept of aligned and non-aligned table content to further disambiguate types of messy tables and articulate the scope of TableCanoniser relative to existing table manipulation tools.

3 System Motivation and Conceptualisation

3.1 Aligned and Non-Aligned Content in Messy Tables

The defining principle of axis-alignment is whether entities and attributes can be extracted from complete groupings or patterns of cells along row or column axes. If all values for a given entity or attribute can be extracted from a single row or column of cells, then those entities or attributes are axis-aligned. Axis-alignment is useful for characterising the complexity of operations required to extract relational data from messy tables. Figures 2–4 illustrate the concept of axis-alignment using selected stylised example tables. These examples are based on figures in Shigarov [35] and the demo cases described below in subsection 3.2.

Relational tables are axis-aligned by definition, with each entity corresponding to a row, and each attribute to a single column. By the same configuration, attributes and entities in Figure 2(c) and (d) are also axis-aligned despite the missing attribute names and values. Figure 2(b) and (e) show how non-relational tables can still exhibit axis-alignment. Figure 2(b) is a transposition of (a), which flips the axis-alignment, entities corresponding to columns, and attributes to rows. In Figure 2(e), attributes **A** and **B** are stored in the same column of cells, violating the structural rules of relational tables. However, values for the two attributes are aligned vertically down the column axis, and can be separated along the comma delimiter using a simple row-to-row operation. Tables with axis-alignment across both entities and attributes are either canonical or can be transformed into canonical tables with relatively simple row-to-row wrangling operations which preserve the number of entities/rows or modify them in a predictable manner (e.g., one column split into two).

Figure 3 shows non-canonical tables with non-alignment across entities and/or attributes but have easy to index patterns that are supported by most existing automated table understanding and data wrangling tools. These tables generally requires slightly more complex operations on implicitly e.g., (b), or explicitly e.g., (d) indexed repeating groups of rows or columns. Such operations often also modify the table dimensions along both directions (i.e., rows and columns). Figure 3(a) shows non-alignment of both entities and attributes, with entities 1, 4 and 7 in the first row, and attributes A, B and C split across three columns each. However, every three columns forms a set of axis-aligned tables, which can easily be extracted by column index and then stacked on top of each other to create a canonical table. Similarly, Figure 3(b) shows 100 entities with three attributes each stacked on top of each other, which can be transformed into a canonical table by extracting, rotating and re-stacking every three rows. Both of these examples exhibit easy to index "lego-like" patterns, whereby attributes for entities or sets of entities are contained in a series of cells with a common length that repeat in a periodic pattern. Figure 3(c) and (d) illustrate cases where attributes for entities are spread across disjoint cells, but can still be retrieved via standard parameterised table reshaping operations (e.g., pivot longer and wider [48]) using row and column indexes. Figure 3(c) shows a cross-tabulation matrix where repetition in attributes A and B is used to index values of C for 9 entities. In order to retrieve a canonical table from the matrix, we would apply a "pivot longer" operation [48] that "melts" [45] attribute A and C into new columns, repeating the a_* and b_* values three times each. The values of attributes A and B serve to index the c_* values, making the transformation possible. Figure 3(d) can be transformed with the inverse "pivot wider" [48] or "cast" [45] operations which can turn the second column of attribute names into new columns, populate them with the paired values in the third column and collapse the first index column into one row per indexed entity. Note that a canonical table cannot be constructed without the index column as it would be unclear whether the value c_2 belonged to i_1 or i_2.

Finally, Figure 4 shows a selection of messy tables that exhibit more complex instances of non-alignment that are not well-supported by current interactive data wrangling tools, but are supported by TableCanoniser. Figure 4(a) and (b) have attributes that are spread across multiple disjoint cells, and entities which take up identically sized blocks (i.e., every two rows is an entity). Figure 4(a) has a periodic pattern where every second cell belongs to the same attribute, allow cells to be extracted by a sequence of cells position alone. However, attribute-value pairs in (b) are shuffled between

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2
a_3	b_3	c_3	d_3

(a)

A	a_1	a_2	a_3
B	b_1	b_2	b_3
C	c_1	c_2	c_3
D	d_1	d_2	d_3

(b)

a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	c_4

(c)

A	B	C
	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	

(d)

A,B	C
a_1, b_1	c_{11}
a_1, b_2	c_{12}
a_2, b_1	c_{21}
...	...
a_5, b_2	c_{52}

(e)

Figure 2: *Tables with axis-aligned entities and attributes.* (a) Relational Table. (b) Transposed Relational Table. (c) Missing Attribute Names. (d) Missing Values. (e) Multiple Attributes in a Shared Column.

A	B	C	A	B	C	A	B	C
a_1	b_1	c_1	a_4	b_4	c_4	a_7	b_7	c_7
a_2	b_2	c_2	a_5	b_5	c_5	a_8	b_8	c_8
a_3	b_3	c_3	a_6	b_6	c_6	a_9	b_9	c_9

(a)

a_1
b_1
c_1
a_2
b_2
c_2
...
a_{100}
b_{100}
c_{100}

(b)

	a_1	a_2	a_3
b_1	c_{11}	c_{12}	c_{13}
b_2	c_{21}	c_{22}	c_{23}
b_3	c_{31}	c_{32}	c_{33}

(c)

i_1	A	a_1
i_1	B	b_1
i_2	C	c_2
i_2	A	a_2
i_2	B	b_2
i_2	D	d_2

(d)

Figure 3: *Tables with mixed-alignment patterns* with indexing that is accessible and parameterisable for transformation by existing tools. (a) Concatenated relational tables. (b) Stacked entities. (c) Cross-Tabulated (matrix). (d) Entity indexed name-value pairs with implicit missing values

A	B
C	D
a_1	b_1
c_1	d_1
a_2	b_2
c_2	d_2

(a)

i_1	A	a_1	B	b_1
	C	c_1	D	d_1
i_2	D	d_2	C	c_2
	A	a_2	B	b_2
i_3	B	b_3	A	a_3
	C	c_3	D	d_3

(b)

	C
A-a_1 **B**-b_1	c_1
A-a_2 **D**-d_2	c_2

(c)

C	a_1		a_2	
	b_1	b_2	b_1	b_2
c_1	d_{111}	d_{112}*	d_{121}	d_{122}
c_2**	d_{211}	d_{212}	d_{221}	d_{222}
c_3	d_{311}	d_{313}	d_{321}	d_{322}
*n_1				
**n_2				
n_3				

(d)

Figure 4: *Tables with non-aligned content* in difficult to index patterns. (a) Identically sized and structured repeating blocks. (b) Identically sized but irregularly structured repeating blocks indexed by merged cells. (c) Nested name-value pairs indexed by merged cells (d) Hierarchical spanning headers with Cell and Table Notes

entity blocks, making extraction using single cell positions impossible and necessitating the use of the attribute cells A, B, C and D as context when extracting values. Figure 4(b) and (c) both have indexing within merged cells, i.e., i_* and c_*, which would require forward filling of the index values after unmerging the cells before canonicalisation using existing tools. Figure 4(d) shows a number of issues commonly found in presentation tables included nested column headers and notes applied to attribute values in both C and D as well as the entire table (i.e., n_3).

Whether existing wrangling tools can canonicalise this example depends on which attributes need to be extracted. For example, if attribute A or the notes n_* are not required, those cells can be ignored and the task simplifies to the cross-tabulated matrix shown in Figure 3(c). However, if we also wanted to extract attribute A, the task becomes more complex and would require additional steps to flatten the hierarchical headers. Notice that a single source table might feasibly be transformed into multiple canonical tables. Structural rules specify how the data which are extracted need to be arranged, but does not prescribe which entities or attributes must be extracted. The suitability of particular table manipulation tools and operations depends on what information is required in the canonical table, and whether that information exists in easily indexed axis-aligned patterns.

3.2 Demo Cases

As part of our exploration of table canonicalisation problems, we collected a number of examples of messy inputs exhibiting a variety of alignment patterns. We include selected examples based on the following data sources as demo cases within the TableCanoniser application: (1) the 2024 Best Chinese Universities Ranking data presented in a web blog,[1] (2) model comparison evaluation results from a machine learning research paper [38], (3) specifications of various smartphones collected from multiple online sources, aimed at comparing and selecting the most cost-effective option, (4) financial transaction records from a bank's sub-accounts, (5) employee payroll data from a proprietary company, and (6) demographic and clinical characteristics of a population in York, provided by TableTidier.[2]

The demo cases were constructed to illustrate various features of the system using subsets or modifications of the original data, with sensitive information removed or anonymised to ensure privacy. For example, the university ranking data was used to construct two related demo cases (1a and 1b), which correspond closely to the non-axis-aligned patterns shown in Figure 4(a) and (b) respectively. Various examples and figures throughout the paper use these cases.

3.3 Comparison of Existing Approaches

To transform non-aligned tables into canonical formats suitable for downstream tasks like data analysis and visualisation, data workers have multiple available approaches: (1) **Manual editing with spreadsheet software** like Excel is intuitive but can be highly repetitive and error-prone. Moreover, these edits often lack traceability, making it difficult to reproduce or audit results. (2) **Writing data wrangling scripts or specifications** with existing programmatic libraries discussed in subsection 2.1 offers flexibility but requires significant programming expertise. (3) **Utilising interactive data wrangling tools and automated solutions** described in sections 2.1 and 2.2 can be more accessible than writing code directly. However, regardless of interface, specifying logic for handling complex messy tables often requires additional abstractions and operations not available in current tools. (4) **Leveraging Large Language Models (LLMs)**, such as ChatGPT, has a similar appeal to interactive tools, but poses additional challenges in terms of prompt engineering and output validation as identified in subsection 3.4.

Existing solutions support most simple table canonicalisation scenarios such as the axis-aligned patterns shown in Figure 5 (a) and (b). However, non-aligned tables often feature entities or attributes spanning multiple rows or columns, making it difficult to directly access their values using index-based techniques. For example, the merged cells and irregular order of attribute-value pairs in Figure 5 (c) are difficult to extract into a canonical form using existing tools despite following identifiable patterns. Moreover, existing tools also lack support for traceability and explainability, which are essential for fostering trust and understanding in data workflows [2, 22]. In their qualitative study of how data workers interact with tabular data, Bartram et al. [2] found that many participants "were willing to perform extremely labor-intensive processes rather than rely on automatic operations about which they lacked trust or confidence", and suggest that interactive experimentation is particularly important for verification tasks. These shortcomings of existing solutions, and the desire to explicitly support interactive verification heavily motivated and informed the design and development of TableCanoniser.

The TableCanoniser system enables declarative and interactive transformation across a broader range of tabular content patterns compared to previous tools, with explicit support for the following subtasks of table canonicalisation:

- matching regions and repeated instances of regions or cells in the input grid containing relational data,
- definition of a schema for the target canonical table following the established structural rules of relational tables,
- extraction of values from the input to the target table, and where necessary,
- recycling of values (e.g., repeating values from header regions across multiple rows) and creation of sentinel values (e.g., `NA` or `NULL` values).

Our grammar is also expressive enough to implement *Table Normalisation* tasks, which involves manipulation of canonical tables into normalised forms such as Codd's 3rd Normal form [9]. However, we note that using table manipulation operations from more limited table wrangling grammars may produce more parsimonious and abstracted code (e.g., melt/cast [45]; fold/unfold [23, 33]; or pivot wider/longer [48]).

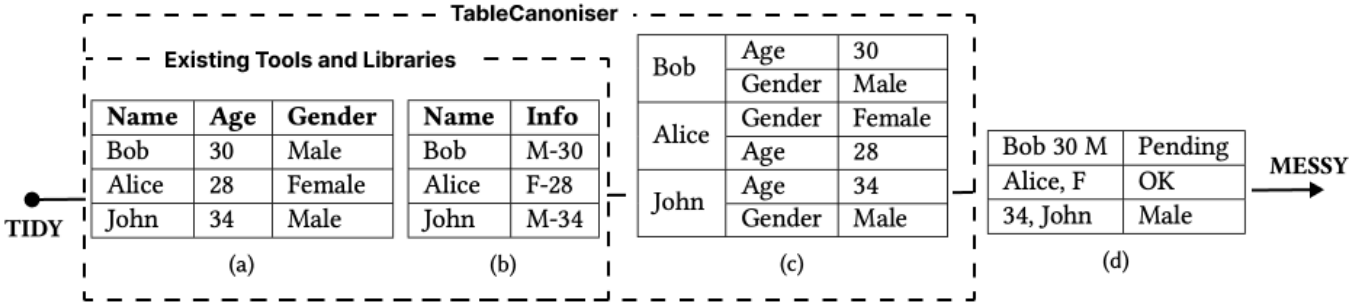
The TableCanoniser system takes as input any loosely-structured grids of values. A single grid could contain multiple tables, of different types, and encode relational information in multiple channels including cell values, table headings and other metadata, or formatting (e.g., font, cell colour, borders). We follow existing data wrangling tools in focusing on transforming table content based on cell values and layout, without reference to style and formatting features. Although formatting channels may contain desirable information, extracting or transforming such data in an declarative and transparent manner is beyond the scope of this work.

3.4 Formative Experiments

Given the growing uptake of general-purpose AI technology like LLMs, an obvious question is whether such technologies can be used for the canonicalisation problems described above. Prior to the design and development of the TableCanoniser system, we experimented with using GPT-3.5-Turbo and GPT-4o to construct canonical tables from the messy tables shown in Figure 6. We experimented with multiple approaches including using GPT to generate and explain python code for the transformations, as well as directly requesting transformed output based on an exemplar relational table, with comparable results from both models. The repeating patterns in Figure 6(a) were handled easily by both models. We provided the messy table and the prompt: "Please write code to first clean `df_messy` and then transform it into a relational table.", and the generated python code produced a canonical table with the expected variables: *Rank, Name, Location, Total Score, Level, Resources, Education, Research, Elite and Global.*

However, we quickly ran into issues of seemingly correct but silently erroneous output, and inaccurate self-explanation by GPT

[1] https://mp.weixin.qq.com/s/tjVUR3wGxR2kAmvt4qeOQg
[2] https://tabletidier.org/table?tid=12115



Name	Age	Gender
Bob	30	Male
Alice	28	Female
John	34	Male

Name	Info
Bob	M-30
Alice	F-28
John	M-34

Bob	Age	30
	Gender	Male
Alice	Gender	Female
	Age	28
John	Age	34
	Gender	Male

Bob 30 M	Pending
Alice, F	OK
34, John	Male

Figure 5: Example tabular inputs containing name, age and gender information for three individuals arranged from tidy to messy. The scope of existing data wrangling tools include (a) *axis-aligned normalised relational tables* and (b) *axis-aligned non-canonical tables*. TableCanoniser further supports (c) *non-aligned tables with merged index cells*, but has limited utility for (d) *non-aligned tabular cells* without discernible patterns.

when switching from the messy table in Figure 6(a) to the modified variation in Figure 6(b).[3] Using a similar prompt as before and providing the same modified messy table data, GPT generated code that produced the output shown in Figure 7. At first glance it appears that the canonicalisation was successful, as the code does in fact produce table satisfying the requirements of a relational table. However, there are multiple errors, including incorrect swapping of values between `Level`, `Resources` and `Education` highlighted on lines 2 and 6. Although we were able to iteratively prompt GPT to generate alternative scripts that rectified these issues, identifying and then describing the issues to GPT was non-trivial.

We also experimented with prompting GPT for explanations of the generated code and linked maps of input-output cells. We found that although GPT was able to offer plain language explanations of specific pandas functions and conditional statements in the generated python code, such sequential narrative did not express the semantics of the overall table canonicalisation task. Furthermore, when prompted produce a cell level mapping between input and output tables based on transformation scripts generated in the same session, GPT returned the same mapping for two distinct pairs of input and output, an example of which is shown in Figure 8. This unreliable self-narration motivates the need for our declarative approach to minimise the risk of silent value-swapping errors in the table canonicalisation process. By contrast, accurate cell-to-cell mappings are trivial to accurately produce in our grammar-driven approach and power the linked highlighting in our interactive system.

As discussed in sections 2.1 and 3.3, existing data wrangling libraries and tools tend to focus on manipulations of relational data and well-formed tables. This limits their ability to express more complex table canonicalisation specifications, which can manifest as exceedingly long or difficult to interpret transformation scripts. This difficulty clearly extends to code from generative models such as GPT using existing specifications of data wrangling operations such as the pandas library in Python. Moreover, our experimentation with GPT showed that it is often non-trivial to evaluate the "correctness" of output tables, especially if a ground-truth does not exist, or multiple canonical tables could be extracted from a source table. Detecting the swapped values in Figure 7 would require close cross-checking of values, or prior knowledge of the irregularities described in the caption for Figure 6(b), neither of which are well-supported by current table processing or data wrangling tools.

The difficulty of detecting erroneous output also suggests we should not rely solely on binary success metrics and percentages when evaluating automated table understanding systems. Many existing systems are evaluated based on their success rate in exactly reproducing ground-truth tables produced by other means [e.g. 27]. This binary approach to evaluation is common in research on automated table understanding [3, 35]. However, this binary success metric obscures the important question of "how wrong" or different the generated tables are compared to the desired output. For physical processes, we can often define an small acceptable tolerance for misalignment or loss that is insignificant to the naked eye or performance. Conversely, such small and difficult to detect anomalies in the output may in fact correspond to systematic errors in the data extraction and table transformation process, as illustrated by our GPT experiments. As such, we might consider extending evaluation of automated tools to include how easy it is to detect incorrect outputs.

The issues we encountered illustrate how fully automated and AI-powered approaches to generating table transformation code are limited by the semantics of available libraries and grammars, and can easily corrupt data in difficult to detect ways. These issues and the resulting insights strongly guided the development of TableCanoniser, and motivated our attempt to not only provide functional support for constructing canonical tables from any messy table that can be represented as a grid of values, but also to leverage the declarative grammar to produce interactive provenance visualisations to support verification workflows.

4 TableCanoniser System Overview

TableCanoniser is an entirely open-source system,[4] comprising of the three components labelled in Figure 1 and detailed below:

[3] Cell highlighting implemented using the R package *emphatic* [8]

[4] https://github.com/TableCanoniser/TableCanoniser.github.io

(a)

1	Rank	Name	Location	Total Score
2	1	Tsinghua	Beijing	992.6
3	NA	Level	Resources	Education
4	NA	36.1	50.0	324.2
5	NA	Research	Elite	Global
6	NA	104.9	88.4	92.1
7	2	Peking	Beijing	898.6
8	NA	Resources	Level	Education
9	NA	34.9	30.5	308.3
10	NA	Research	Elite	Global
11	NA	101.8	92.4	71.4
12	3	Zhejiang	Zhejiang	793.8
13	NA	Level	Resources	Education
14	NA	34.9	28.4	284.7
15	NA	Research	NA	Global
16	NA	92.6	NA	53.7
17	NA	Notation	The above ... rankings	NA
18	4	Shangjiao	Shanghai	776.3
19	NA	Level	Resources	Education
20	NA	35.6	26.4	267.2
21	NA	Research	Elite	Global
22	NA	106.5	64.6	39.6
23	5	Fudan	Shanghai	697.0
24	NA	Level	Resources	Education
25	NA	36.6	21.3	252.6
26	NA	Research	Elite	Global
27	NA	79.3	62.6	44.1
28	6	Nanjing	Jiangsu	656.1
29	NA	Level	Education	Resources
30	NA	37.1	266.6	9.2
31	NA	Research	Elite	Global
32	NA	70.4	58.3	29.2

(b)

Figure 6: (a) [Demo Case 1a] A cropped image of a table showing university rankings for 5 of 6 universities. The image was converted into a messy table and translated into English to form Demo Case 1a in the TableCanoniser system. The entities and attributes of this messy table are non-aligned, but attribute values for each entity (i.e., instances) are presented in the same repeating pattern, allowing for position-based extraction. (b) [Modified Demo Case 1a*] A modification of Demo Case 1a which replaces the repeating pattern with the following irregularities: (1) different positions for attribute name-value pairs (e.g., Resources, Level, Education are shuffled on lines 8–9 compared to lines 29–30); (2) contextual notes on line 17; and (3) missing values on lines 15–16.

The declarative grammar, and code for parsing it, is defined using TypeScript through the *TableCanoniserTemplate* interface, which establishes the structure of the grammar. The *transformTable* function, along with other auxiliary functions, is responsible for parsing the given specification that adheres to the *TableCanoniserTemplate*, enabling the transformation of messy, two-dimensional data into canonical tables. We have packaged these capabilities into an NPM package named *table-canoniser*.[5] Developers familiar with coding can easily integrate it into their projects to generate target tables by directly invoking the API.

Linked visualisations of the grammar and transformations support easier verification and comprehension of the table canonicalisation. Tree-based visualisation of the *TableCanoniserTemplate* supports learning and comprehension of the grammar specification, whilst the inputs and outputs to the *transformTable* function are used to visually match messy input cell values in with corresponding cells in the output tables.

[5]https://www.npmjs.com/package/table-canoniser

	Rank	Name	Location	Total Score	Level	Resources	Education	Research	Elite	Global
1	1	Tsinghua	Beijing	992.6	36.1	50.0	324.2	104.9	88.4	92.1
2	2	Peking	Beijing	898.6	34.9	30.5	308.3	101.8	92.4	71.4
3	3	Zhejiang	Zhejiang	793.8	34.9	28.4	284.7	92.6	NA	53.7
4	4	Shangjiao	Shanghai	776.3	35.6	26.4	267.2	106.5	64.6	39.6
5	5	Fudan	Shanghai	697.0	36.6	21.3	252.6	79.3	62.6	44.1
6	6	Nanjing	Jiangsu	656.1	37.1	266.6	9.2	70.4	58.3	29.2

Figure 7: [Modified Demo Case 1a*] Canonical table produced by executing code generated by GPT-3.5-Turbo using the ChatGPT interface on the messy table shown in Figure 6(b). The swapped values highlighted on lines 2 and 6 are due to the irregularity (1) described in the caption for Figure 6(b).

	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43	44	45	46	47	48	49
Original	(0, 0)	(0, 1)	(0, 2)	(0, 3)	(2, 1)	(2, 2)	(2, 3)	(4, 1)	(4, 2)	(4, 3)	...	(21, 0)	(21, 1)	(21, 2)	(21, 3)	(23, 1)	(23, 2)	(23, 3)	(25, 1)	(25, 2)	(25, 3)
New	(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)	(0, 6)	(0, 7)	(0, 8)	(0, 9)	...	(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	(4, 6)	(4, 7)	(4, 8)	(4, 9)

Figure 8: [Modified Demo Case 1a*] Cell-to-cell mapping showing where original input cells mapped to in the canonical output table. The same mapping was generated by GPT-3.5-Turbo for two different transformation scripts on the same input with different execution outputs.

The interactive application[6] provides a unified interface for specifying and visualising the TableCanoniser grammar and transformations. The system empowers users to iteratively author, verify and refine transformation mappings with the support of bi-directional highlighting between messy input grids and canonicalised output tables, and node-based specification of the template through interaction with the specification tree visualisation. It is implemented as a client-side, single-page web application, which can be accessed directly through a web browser. TableCanoniser is implemented using Vue.js 3.0 [42] as the frontend framework, with Handsontable [14] for table rendering, Monaco Editor [29] for code display, D3.js [5] for visualisation, and the *table-canoniser* package for parsing the grammar.

4.1 Table Canonicalisation Grammar

The TableCanoniser grammar serves several purposes. Firstly, it provides a declarative language to specify table manipulation operations using the semantics and logic of higher-level table canonicalisation process. Secondly, the structured representation of that high-level logic as mapping patterns can be encoded visually for verification of the table transformations. Finally, it provides a semantic interface for interactive iterative authoring and refinement of table canonicalisation specifications, as described in subsection 4.2.

A TableCanoniser mapping specification template is a hierarchical tree consisting of mapping pattern nodes defined in a `TableCanoniserTemplate` object. Mapping pattern nodes define `match` conditions on regions of the messy input table, and can be optionally extended with `extract` operations on matched instances. Match conditions describes patterns of position, size, direction and cell type or value constraints that instances of extractable data can be found in. These conditions can be nested by specifying another pattern node under the `children` property of a parent node. For example, as illustrated in the Usage Scenario in subsection 6.1, all the cells relating to a single university in Figure 6(b) could form instances of a parent pattern, while sub-pattern matching specific attribute-value pairs can be defined to extract those data to the output table.

The following sub-sections describe the JavaScript-based `TableCanoniserTemplate` object for specifying mapping patterns in more detail.

4.1.1 Hierarchy. The `children` property of a mapping pattern node specifies an array of mapping pattern objects (matches or extractions, as described below), which are applied in turn each time the parent pattern is successfully matched. A typical use of this hierarchy feature would be to define a match pattern that selects regions based on a particular constraint where there are constraints that say the region must be preceded by a blank row but have a value in its top-left cell. Multiple child extraction patterns can then be defined to select a subset of the region cells for output into the target table, each with their own constraints if necessary. When the parent pattern is matched, each of the child extraction patterns is applied to matched instances to extract cells of interest.

4.1.2 Match. Creating a mapping pattern node begins with specifying match conditions, often based on a specific reference region from the input table. This region can be used to match and extract data from similar instances, or form parent instances for data extraction using children patterns. Match conditions are described through the required `match` property of a mapping pattern, which has two required components, and two optional ones:

- The `startCell` property requires an object describing a position for the top-left position of the region. Its properties, `offsetX` and `offsetY`, describe an offset (either positive or negative) from a reference location `offsetFrom` (one of `"topLeft"`, `"topRight"`, `"bottomLeft"`, `"bottomRight"`),

[6] https://tablecanoniser.github.io

defaulting to top-left relative to the `offsetLayer` property (one of `"current"`, `"parent"`, `"root"`, defaulting to `"current"`). For the top-level match, the offset layer will be the root, for sub-matches it typically will be the parent, and current will usually be used by the match constraint.
- The `size` property requires an object describing the `width` and `height` of the matched region. Both are either a positive integer or can be `null`, in which case the width or height is determined by specified `constraints`.
- An optional `traverse` property requires an object describing whether to repeatedly apply the match within the parent region relative to the start cell in the `xDirection` and/or `yDirection`, both can be either `"after"`, `"before"`, `"beforeAndAfter"`, or `null` (the default).
- An optional `constraints` property requires an array of objects each describing a particular cell that must satisfy a given constraint for the match to occur. It describes the same properties as `startCell` (i.e., `offsetX`, `offsetY`, `offsetFrom` and `offsetLayer` where layer would typically be the current layer). It also specifies `valueCstr` (value constraint), which must be one of the `TableCanoniserKeyWords` constants; `Number`, `String`, `None`, `NotNone` or a specific value, which respectively denote that the cell at this position must a number, a string, have no value, be non-empty, or must have a specific value.

4.1.3 Extract. A mapping pattern can be extended with an extract operation by defining the optional `extract` property. An extract operation can be position-based, context-based or value-based. The `extract` object must define exactly one of the optional properties `byPositionToTargetCols`, `byContext`, `byValue`, each of which are described below:

- `byPositionToTargetCols` extraction is the simplest to specify. If each instance of regions satisfying the match conditions contains n cells, then the `byPositionToTargetCols` property can define an array of strings of length n, where each cell in the matched region will be transformed to data in column with a name given by the string in the corresponding position in the array, or will not be transformed if `null` is given instead of a column name.
- `byContext` extraction is an object describing a `position` property (one of `"left"`, `"right"`, `"above"` or `"below"`) for a reference cell relative to the current cell for each cell within the region. The `toTargetCol` property provides an optional mapping function that takes the reference cell position and value and returns a column name. If not set, the content of the reference cell is returned as the target column name.
- `byValue` extraction is the most flexible option. It specifies a function which gets passed the entire matched region as a 2D table and returns an array of column name strings with length equal to the number of cells in the 2D region.

It may be the case that the output table would have different lengths after extracting values from the matched region. The optional property `fill` can be used to specify how to resolve discrepancies. It can be set to one of the `TableCanoniserKeyWords` constants, `Auto` (the default), which gives an empty string as the value, `Forward`, where the last extracted value is recycled to the end of the column, the value `null`, where no filling is done and the column lengths are mismatched, or some specific value (e.g., a string or number).

The provided extract operations offer different trade-offs between power and ease of specification. As the name suggests, `byPositionToTargetCols` succinctly supports matching and extracting values from repeating positions in identical blocks like in Figure 4(a). In order to support irregular blocks like Figure 4(b), `byContext` allows the use of context cells in the extraction logic (i.e., utilising the both cells in attribute-value pairs). Finally, the ability to pass custom functions to `byValue` provides support for arbitrarily complex messy tables, including the hierarchical row and column labels and cross-tabulated values addressed in Demo Case 6.

4.1.4 Expressivity and Extensibility. The TableCanoniser grammar provides additional expressive power through the option of specifying functions (i.e., JavaScript code) rather than specific values. In addition to custom extraction functions in `byValue`, the `valueCstr` property of the constraint object was previously described as requiring a type (e.g., string), a state (e.g., non-nil) or a specific value (e.g., "Sales"), but it can also take a function which accepts the constraint cell value as an argument and returns a boolean. This allows more complicated logic like a constraint to match one of three possible values. Similarly the `position` property of `extract-byContext` can take a function that is passed the cell being extracted (which has offsets and values) and returns a cell position object specifying and x- and y-offset and an offset layer for the context cell. Recognising that not all users will be able to implement custom functions, we support implementation of additional helpers, reducing the need for users to manually write custom functions. For example, users can use a predefined `pairSort` operator in `byValue` to reorder and extract pairs of cells according to specified criteria. These advanced features of the grammar are described in the TableCanoniser documentation available in the code panel, and are illustrated further in the provided examples.

4.2 TableCanoniser Visualisations and Interactive Application

Although the TableCanoniser grammar can be used as a stand-alone programmatic library, we also provide an interactive application with provenance visualisations to support users with minimal or no coding experience to author simple extractions and inspect complete specifications.

4.2.1 Linked Visualisations. As shown in Figure 9, the interface offers multiple visual encoding and interaction features based on the declarative grammar to support verification and transformation subtasks in table canonicalisation:

- The ***bidirectional input-output highlighting*** support users in checking correspondences between input and output cell values in a more persistent format compared to transient checks performed when copying and pasting values from a messy input table to a target output table.
- The ***specification tree visualisation*** provides an visual representation and linked interface for the mapping specification code, allowing users to author, inspect and refine

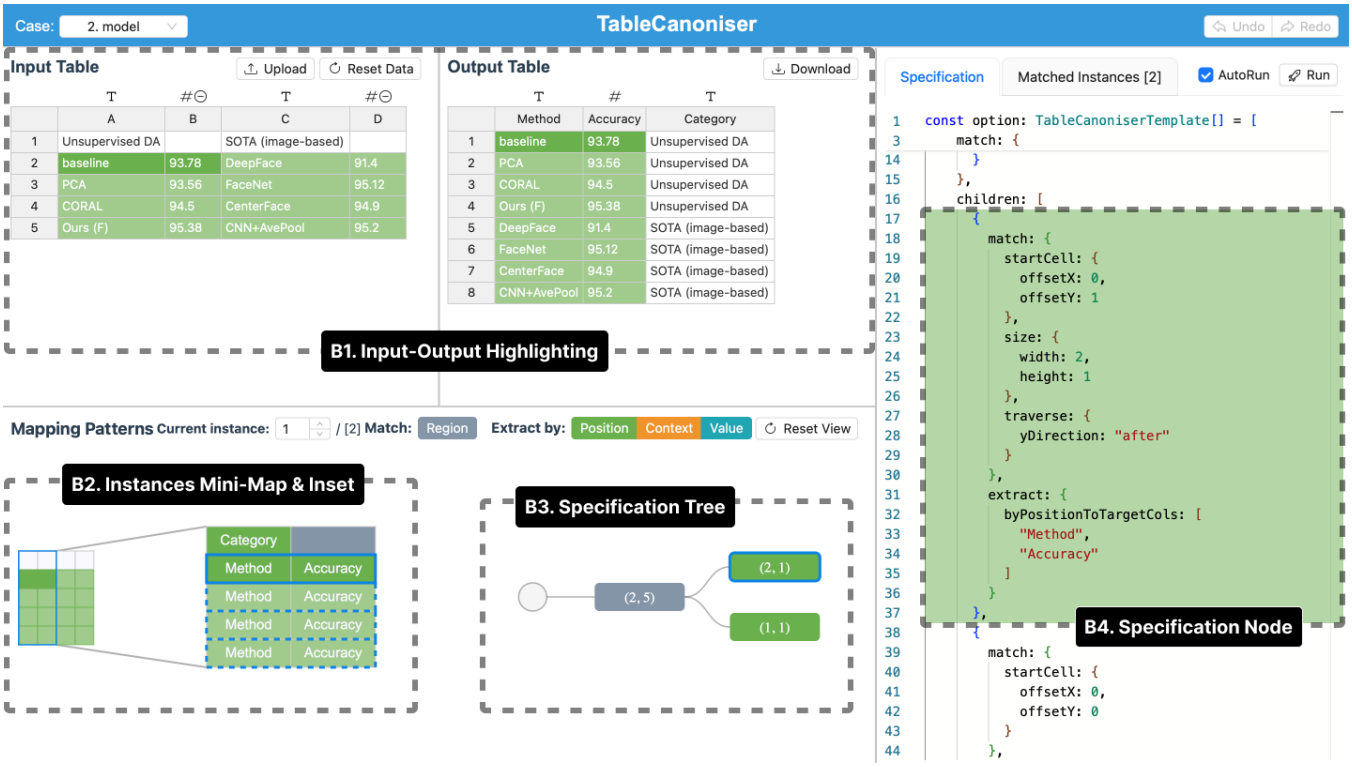


Figure 9: The TableCanoniser interface allows the user to select or upload an input table and then specify transformation to a canonical table via interactive generation of a grammar-based specification. The interface also allows the user to verify the behaviour and correctness of the transformation by interacting with linked visualisations. Users can trace corresponding input and output cells with bidirectional highlighting [B1], examine extract logic for instances of matched patterns [B2], inspect the specification in a tree representation [B3] and editable code [B4].

simple mappings via interactions with tree nodes, thereby lowering the threshold for usage by users with minimal coding expertise.

- Finally, the ***matching instances mini-map*** provides users with an overview of which areas and cells in their source spreadsheets have been matched, as well as a ***matching instance inset*** which shows specific details of the extraction logic and target columns for a selected matched instance. This allows users to check the parsed matching behaviour of the mapping specification against their own understanding of patterns in the input spreadsheet.

4.2.2 Application Interface. The TableCanoniser interface consists of four panels which are also visible in Figure 9. The first top-left panel displays an editable version of the "messy" ***input table***, the second top-middle panel displays the resulting canonical ***output table*** produced from the transformation code, along with a button to download the generated table. The third right-side ***code panel*** displays the mapping specification in an editable form, along with a tab for displaying low-level debug information. Finally, the bottom ***mapping patterns panel*** displays custom verification visualisations. The left-hand side of the bottom panel contains the *matching instances mini-map*, which shows recognised patterns for each top-level match instance, and the right-hand side contains the *specification tree visualisation* of the mapping.

The interface includes a drop-down to select and load the input data and sample mapping specification for the six different Demo Cases. Users can also upload their own flat file as input (e.g., `.csv` or a sheet from a `.xlsx`), and then interactively author a transformation specification as described in the following subsections and usage scenarios in section 6. The interface also includes standard undo and redo functionality for changes to the transformation specification.

4.2.3 Advanced Use and Debugging. The code panel provides documentation for the necessary and optional parts of the grammar while hovering over the code of objects or property names. In the event that the user writes specification code that is invalid, TableCanoniser cannot parse the transformation, and will instead display an error with the location of the problem. The output table and mapping instances visualisation will not be updated, i.e., the results from the previous working transformation code will remain. The behaviour is the same in the case of syntax errors within the mapping specification.

As well as the specification itself, the code panel also contains a tab containing low-level debug information, detailing in code for the results of each extract and match operation performed in the

transformation. This is a text-dump of the information that is used to power interactive inspection of individual elements in the input or output table and the mapping instances visualisation previously described.

5 Interactive Workflows

To illustrate the intended usage and features of the TableCanoniser application, we walk-through stylised workflows for authoring and verifying mapping transformations, followed in section 6 by two example usage scenarios based on demo cases available within the application. We also provide a narrated walk-through of the system as a video in the supplementary materials.

5.1 Interactive Authoring of Specifications

When the user first uploads some messy table data, this data will be shown in the input table panel, along with an empty specification. Since the mapping specification is empty, so too is the generated canonical output table and the two visualisations showing the structure and matching instances of the mapping.

The first step in authoring a specification is to describe conditions for matching region instances in the input table, which in many simple cases correspond to rows in the output table. To do this the user clicks the "Match Region" button in the mapping patterns panel. The interface enters the match region mode and instructs the user to select the region in the input table. Once they have done this, the system adds a new pattern node to the specification and populates the `match` property (showing this both in the code panel and the mapping pattern panel) and automatically runs the code. Once the transformation code has run, the instances mini-map visualisation shows how many instances of this match pattern were found in the input table, and the location of each. Once this specification code has been generated, the user is free to modify it manually and the system will regenerate and rerun the mapping code, updating the output table and corresponding visualisations.

When adding new pattern nodes, the system detects whether the extraction is located within an existing match pattern and makes the extract node a child of the match node. It also preserves all manual modifications to the existing pattern nodes. Alternatively the user can right-click a pattern node in the mapping patterns panel to add a child pattern. As well as editing the specification code directly, if the user ever wishes to, they can right-click on a node in the specification tree to delete this operation from the transformation specification. Conversely, if the user changes the transform specification via the code, the tree visualisation is updated to reflect this.

Users can set constraints to restrict and refine which region instances can be matched. To do this, the user right-clicks on the pattern node they would like to restrict in the specification tree visualisation. They are then asked to select the cell to which the constraint will be applied. This generates an partially populated `constraints` property for the selected pattern node in the specification. The user then enters a valid constraint condition such as restricting matching to instances where the selected cell position is a particular value or type of value. It is possible to specify multiple constraints on different cells in a mapping pattern. Regions in the input table must satisfy all constraints to be matched. Constraints can be added iteratively and checked using the instances mini-map visualisation.

To create an extraction operation for the previously created match operation the user first selects the match node in the specification tree visualisation. After this they can click the "Extract by Position" button. This prompts them to select a cell or region within the input table. Doing so populates the mapping specification with the code necessary to perform the match with offsets relative to the parent region. The user then fills in the `byPositionToTargetCols` property with array of strings which are the target column names for each of the cells in the extracted region. For example, Figure 10 shows an example of a simple extract-by-position pattern for two cells from a matched parent pattern.

The other two extraction operations ("Extract by Context" and "Extract by Value") are created similarly, with the user instead selecting the corresponding extract button for each instead of the "Position" one.

The specification tree visualisation shows each match or extract operation as a node in a tree reflecting the hierarchical structure of the patterns. Each of the four operation types are displayed with a different colour and a text label shows the size of the region matched by that operation. In addition, icons are displayed at the top-left of the Node indicating that the operation has one or more constraint applied to it. For example, the # icon above the match node in Figure 1, indicates it has a constraint with the top-left cell in the region must be a number, as shown in the code panel. Hovering over this icon highlights the specific cell in the region to which this constraint applies.

5.2 Interactive Verification

Whenever the specification code changes and is valid, TableCanoniser will automatically run the transformation, displaying the results. There is a checkbox to disable auto-run. After successful changes to the specification, the system stores an undo point so the user can undo and redo changes during authoring. Interactive linked exploration of the input and output tables, and the mapping patterns visualisation can be used both to verify the correctness of an existing complete transformation, as well as to assist the user during iterative creation of the transformation specification in examining each step of the hierarchical construction.

The interface shows several linked views. When the user selects an item in one of these views, the corresponding items in the other views are highlighted. A simple use for this is tracing the source of a value in the output table. As shown in Figure 11, when a user selects the cell in row 6 of the Education column in the output table, the corresponding source cell is highlighted in the input data, the matching instances mini-map, as well as the relevant nodes in the specification tree visualisation. Similarly, if users want to trace the source cells for an entity or attribute, they can select entire rows or columns in the output table to highlight the corresponding input cells and the relevant mapping nodes, as shown with the selection of the *Level* column in Figure 1. They can also select a region of cells in the input table to see whether and where those cells were extracted to.

Alternatively, if the user wishes to explore the transformation from the mapping pattern perspective, they can click on individual

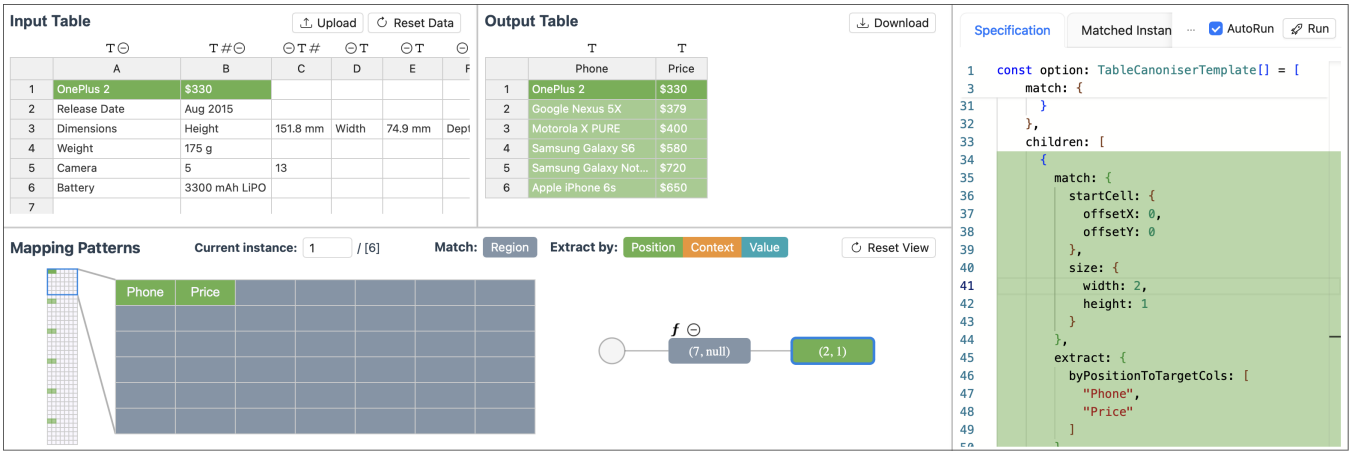


Figure 10: [Demo Case 3] A 7 x *n* matched parent region (with height is determined by two Constraints) where the first two cells on the top row are extracted and given the column label "Phone" and "Price" respectively. This results in an output table with those two columns and a row corresponding to each of the 6 matched regions, extracting the row values from the left topmost two cells in each matched parent region.

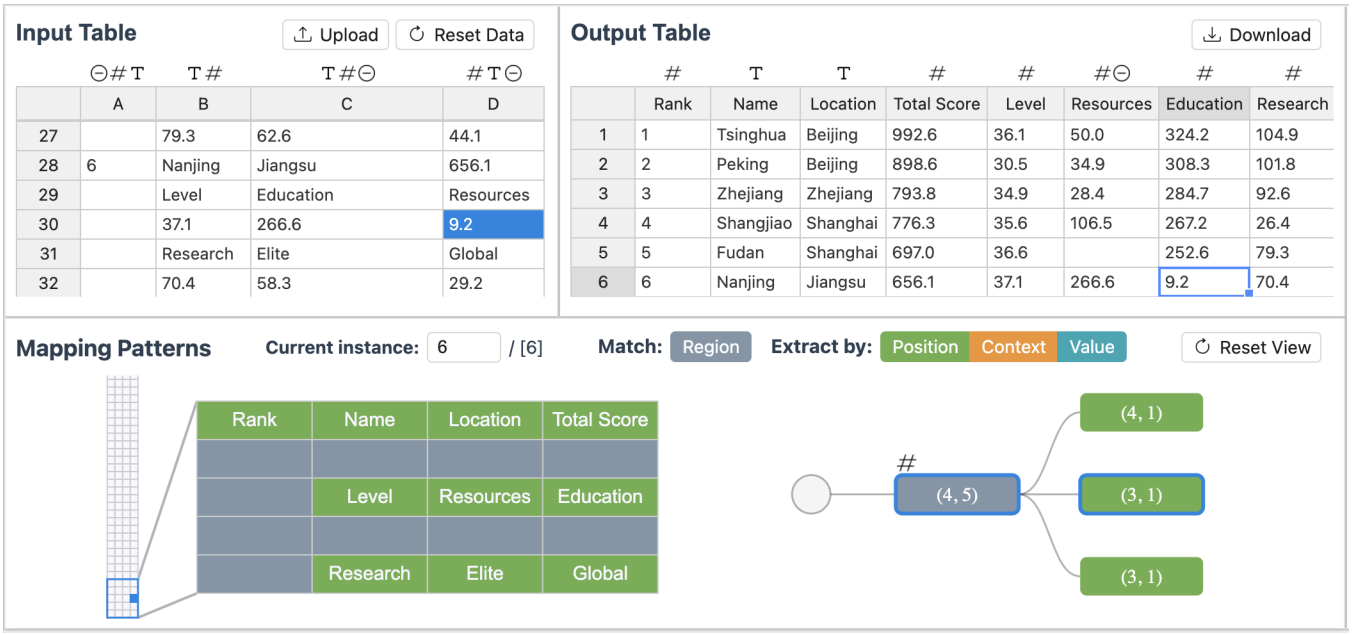


Figure 11: [Demo Case 1b: Step 4'-2] When the user selects a cell of interest in the output table, the corresponding cell in the input table is highlighted, along with the pattern(s) that matched and/or extracted that cell, as well as the location in the matching instance visualisation. Note the specification code panel is not show in this screenshot.

nodes in the specification visualisation to see what values where extracted by operations in that node. As shown in Figure 12, clicking on the middle extract node highlights all the cells matched by that pattern in the input table, instances mini-map, and also highlights the corresponding extracted values in the output table.

The matching instances mini-map gives an overview of the entire input table showing where each instance is matched. It shows the total number of top-level instances, and gives controls to step through these instances one by one. As the user does this, the overview visualisation shows the location of the selected instance along with an inset which displays a representation of the matched region and shows the target column name for any extracted cells highlighted with the colour of the responsible extraction pattern type. As the user steps though match instances, such as the two shown in Figure 13 the corresponding cells are highlighted in the input and output tables. This allows the user to easily see and

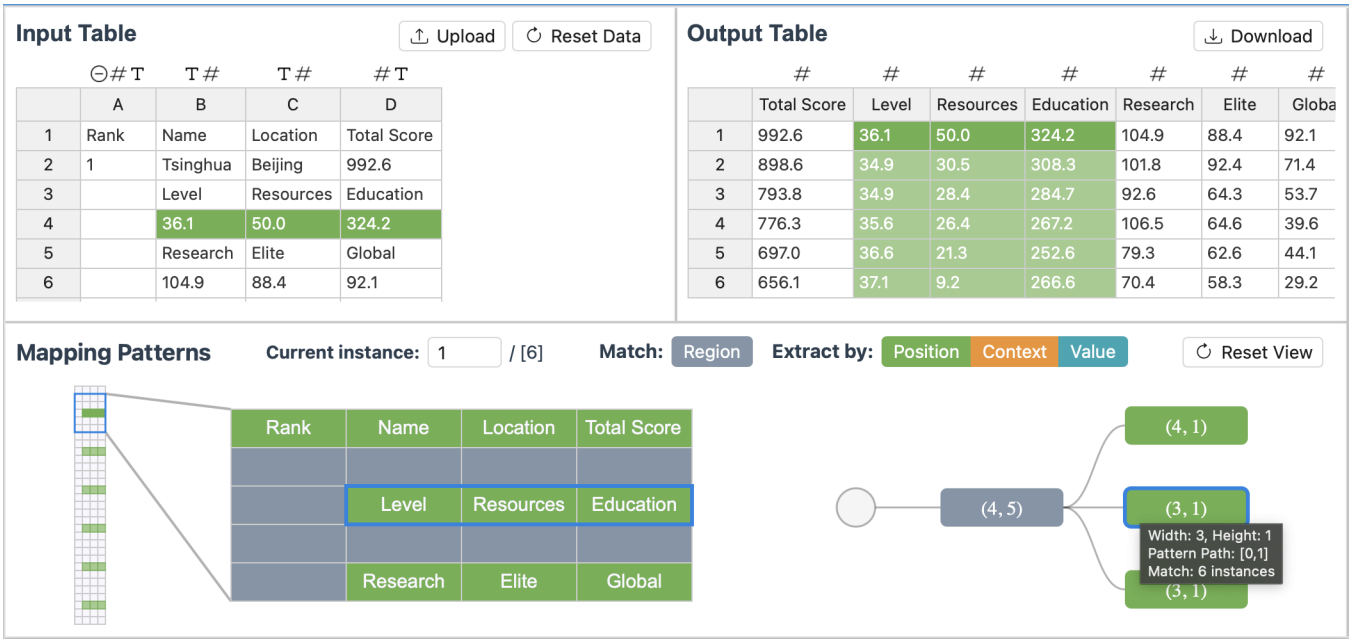


Figure 12: [Demo Case 1a] When the user selects the middle extract-by-position pattern node in the specification tree visualisation, the cells matched by this pattern are highlighted in the input data, the matching instances overview visualisation, and in the output table. Note: the specification code panel is not show in this screenshot, but the code for this extract pattern is highlighted.

correct situations where a matching pattern works for some but not all of the input data.

TableCanoniser supports automatic detection of data types within each column of both the input and output tables, aiding users in identifying type inconsistent issues. These are shown as icons above the column, as shown in all screenshots: *T* for text, # for number and ⊖ for empty cells. When the user clicks on one of these, it highlights all cells in that column, and the corresponding cells in the mini-map overview and in the other table, as shown in Figure 14. These data type icons use the same visual icons as the constraint icons for the pattern nodes in the specification tree visualisation.

6 Usage Scenarios

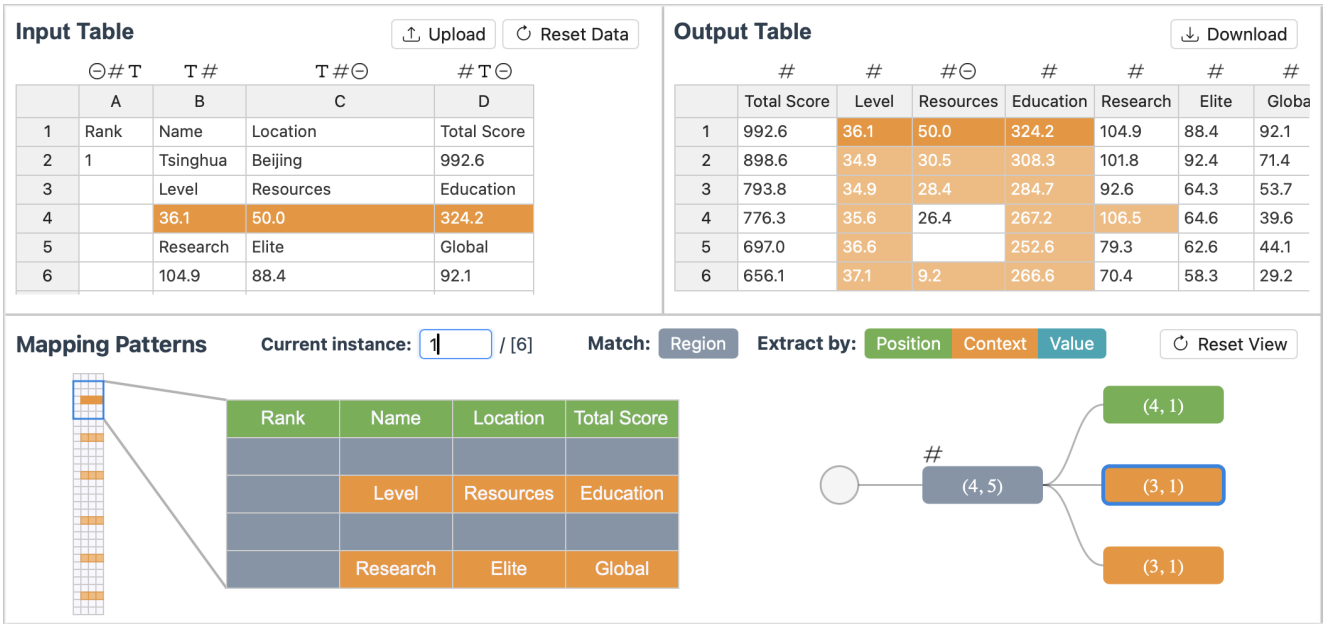
To demonstrate the effectiveness and usability of TableCanoniser, we have implemented the demo cases based on the data described in subsection 3.2. The cases are accessible through the drop-down menu in our system interface. For each case, we provide at least one specification option, enabling users to explore different transformation strategies and gain hands-on experience with TableCanoniser. The following walk-through of two selected demo cases illustrate how TableCanoniser can support users in extracting canonical tables from messy inputs across different scenarios. A narrated walk-through of Demo Case 1 is also provided in the supplementary video figure.

6.1 Demo Case 1: Transforming Messy Web Tabular Data

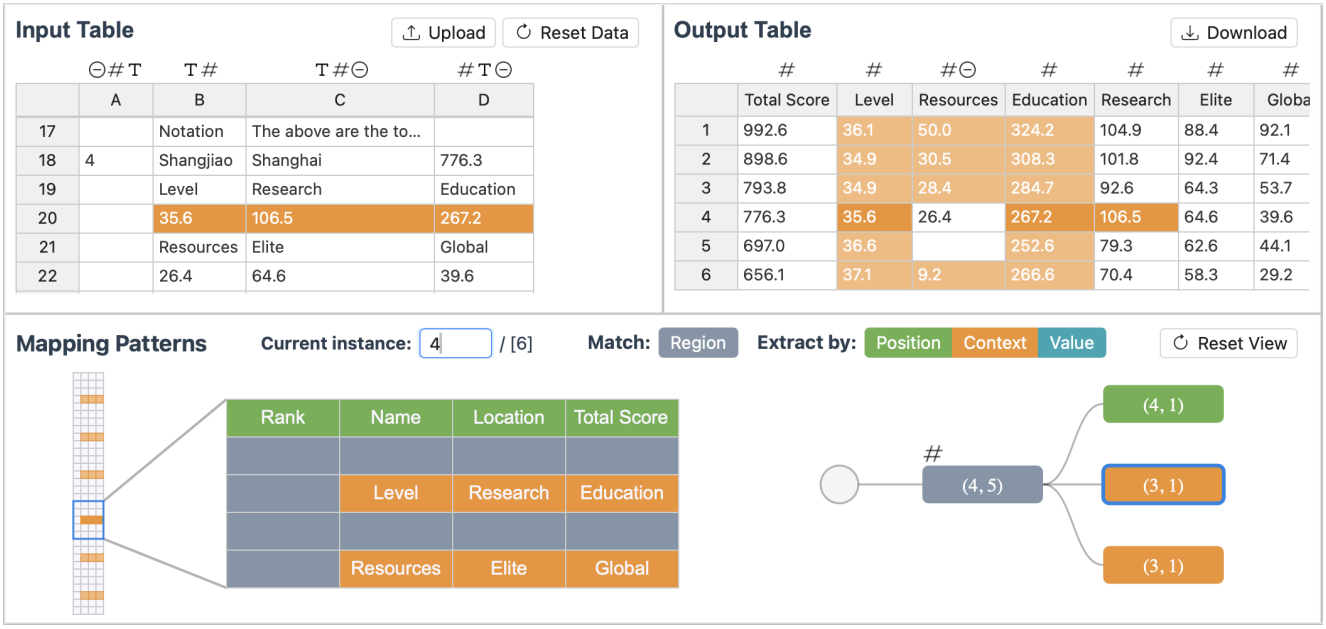
This scenario demonstrates how TableCanoniser aids users in iteratively transforming diverse messy web data and validating the transformation result. Assume Emily, a data analyst, is interested in analysing the top ranked universities in China. She finds data from a web blog that contains the 2024 Best Chinese Universities Ranking (as shown in Figure 6(a)). However, the table is structured for presentation rather than analysis purposes. Each university entity spans multiple rows and some columns contain multiple variables. It is difficult to specify appropriate transformation and extraction operations using existing data analysis and visualisation tools. Hence, she decides to use TableCanoniser to produce a canonical table before analysis.

Step 1: Match the university region. After uploading the data and glancing through it, Emily sees that every five rows correspond to a university. Therefore, she clicks the "Match-Region" button and selects rows 2–6 in the input table, generating a pattern with a size of (4, 5) (representing width and height, respectively). The system detects six instances matching this pattern but without any output table yet, as no "Extract" operation is specified.

Step 2: Extract the first row by position. Emily inspects each instance by clicking the matched areas in the minimap and notes that variable values (e.g., "Rank", "Global") are consistently positioned within the pattern. To extract these values, she clicks the "Extract-by-Position" button and selects the first row in the pattern, creating a position-based extract sub-pattern. By default, the columns are named "C1", "C2", etc.



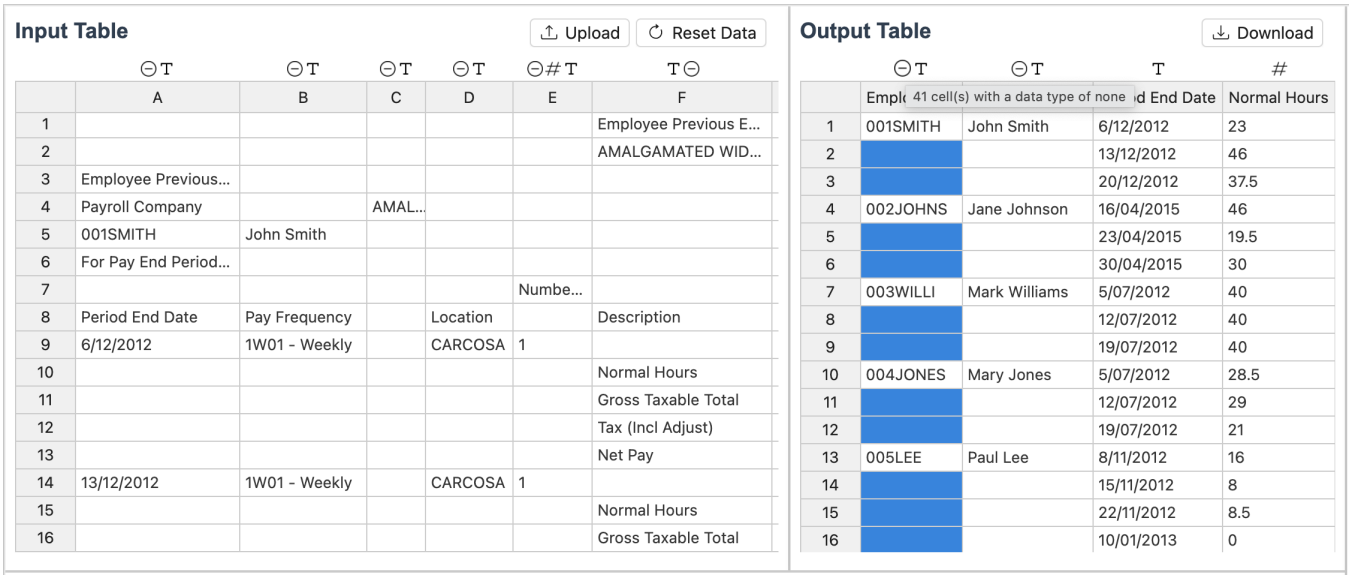
(a) Viewing the first instance with corresponding input and output data highlighted (dark-orange highlight).



(b) Viewing the fourth instance with corresponding input and output data highlighted. Notice the instances mini-map visualisation (bottom-left) show "Resources" and "Research" in different rows than the first instance.

Figure 13: [Demo Case 1b] The user selects the first extract-by-context (orange) node in the specification tree visualisation to verify its matches. Stepping through instances they can see which data cells were matched based on the context (label) node posited directly above them in the input data.

Step 3: Adjust the specification. Observing that the output table closely resembles the input table, Emily, in conjunction with the instances mini-map visualisation, confirms that each row of the parent pattern matches the sub-pattern, as the system automatically detects the potential traversal for the first sub-pattern. Emily clicks the newly created sub-node in the structure visualisation to navigate its corresponding specification in the code panel. After commenting the `traverse` property out (indicating no traversal is needed) and modifying the `byPositionToTargetCols` property to



	A	B	C	D	E	F
1						Employee Previous E...
2						AMALGAMATED WID...
3	Employee Previous...					
4	Payroll Company		AMAL...			
5	001SMITH	John Smith				
6	For Pay End Period...					
7					Numbe...	
8	Period End Date	Pay Frequency		Location		Description
9	6/12/2012	1W01 - Weekly		CARCOSA	1	
10						Normal Hours
11						Gross Taxable Total
12						Tax (Incl Adjust)
13						Net Pay
14	13/12/2012	1W01 - Weekly		CARCOSA	1	
15						Normal Hours
16						Gross Taxable Total

	Empl		d End Date	Normal Hours
1	001SMITH	John Smith	6/12/2012	23
2			13/12/2012	46
3			20/12/2012	37.5
4	002JOHNS	Jane Johnson	16/04/2015	46
5			23/04/2015	19.5
6			30/04/2015	30
7	003WILLI	Mark Williams	5/07/2012	40
8			12/07/2012	40
9			19/07/2012	40
10	004JONES	Mary Jones	5/07/2012	28.5
11			12/07/2012	29
12			19/07/2012	21
13	005LEE	Paul Lee	8/11/2012	16
14			15/11/2012	8
15			22/11/2012	8.5
16			10/01/2013	0

Figure 14: [Demo Case 5] When the user clicks the ⊖ type icon above the first column in the output table, the corresponding cells with that data type (i.e., the empty cells) in the column are highlighted.

change the default column names, each row in the current output table corresponds to a single university.

Step 4: Extract other rows by position. Subsequently, Emily continues to click the "Extract-by-Position" button to select the third and fifth rows within the parent pattern (skipping the first cell as it is empty), and adjust their column names, yielding another two sub-nodes as shown in Figure 12. Finally, Emily obtains a canonical table ready for downstream analysis.

This input data follows a regular pattern and all the variables are extracted by position, making it relatively straightforward for Emily to handle. To further simulate real-world diversities and reveal the capability of TableCanoniser in dealing with more complex data, let Emily attempt to transform the variation of case 1 described in Figure 6(b). Emily tackles this modified version with the same steps as before, but when the first sub-pattern from Step 3 is applied, she notices data type inconsistencies in all output columns.

Step 3'-1: Add a constraint. By clicking the type icons above the column headers, she inspects these inconsistent cells and discovers that the 17th row contains irrelevant notes in the input table (highlighted on Line 17 of Figure 6(b)), causing the parent pattern to mismatch instances. To fix this, she right-clicks the parent node in the structure visualisation, selects "Add Constraint", and clicks on the top-left cell within the pattern in the input table, ensuring that the first cell of the pattern should be a numeric value. Once the constraint is applied, all instances are correctly matched.

Step 4'-1: Examine the empty cell. After doing Step 4 as above, Emily notices another type inconsistency in *Resources*. By examining the highlighted cell, she finds missing data in the input table (highlighted in lines 15–16 in Figure 6(b)), which accounts for the empty value in the output.

Step 4'-2: Reset extraction to context. Additionally, Emily observes an outlier ('9.2') in *Education* and realises that scores in the input table should be extracted based on their above cells (i.e., by context) rather than fixed positions after viewing the cell mapping between the input and output tables, as demonstrated in Figure 11. Therefore, she right-clicks the last two child nodes in the tree-based visualisation and selects "Reset Target Cols" to "context". After switching this and further validating interaction, Emily confirms the final transformation result is accurate and downloads the output table.

6.2 Demo Case 5: Merging Multiple Embedded Tables

This scenario is motivated by a real-world task assigned to Luke, an enterprise employee. Luke is tasked with organising his company's payroll data into a format that can be easily queried. Specifically, giving an employee ID/name and a period end date can obtain the normal work hours and the corresponding pay. However, the original payroll data is embedded in a spreadsheet with multiple sub-tables. A small region of the spreadsheet can be seen in the input table panel in Figure 14. Each sub-table represents an employee's payroll records for a specific time period, with fixed header and footer information.

Step 1: Match the sub-table region. After uploading the spreadsheet, Luke begins by clicking the "Match-Region" button to select the first sub-table (rows 1–26) in the Input Table, creating a pattern with a size of (12, 26). Based on prior knowledge that the height of each sub-table is not fixed, Luke expects some incorrect matches. He confirms this by navigating through instances using the up and down keys with the "Current instance" field in the instances mini-map visualisation.

Step 2: Set region height via constraints. Luke adjusts the height in `size` property to `null`, allowing the pattern to handle instances with varying heights (determined by constraints). To correctly match each sub-table, Luke adds two constraints to the pattern, which specify the beginning (row 1, column 6) and end (last row,

first column, corresponding to the `offsetFrom` property set to "`bottomLeft`") of each instance to be specific text strings (similar to Step 3'-1 in Case 1).

Step 3: Extract employee info by position. Knowing that the employee ID and name are located at fixed positions in each sub-table, Luke clicks the "Match-Position" button to select the first two cells of the 5th row. Then he comments out the `traverse` property and renames the default columns to "Employee ID" and "Employee Name" (similar to Step 3 in Case 1).

Step 4: Match the Date region. Each sub-table contains multiple period dates with different region heights. Luke applies a similar method to the previous steps to define a "Region" pattern for these dates, enabling downward traversal and adding constraints. Initially, Luke sets two constraints: the top-left must be a string, and the bottom-right must be a number. However, upon inspecting the matching results by clicking the constraint icons above the pattern node, Luke realises that the region stops matching after the second row, prematurely satisfying the constraints. After several attempts, he adjusts the first constraint to require that the cell below the left-bottom cell (`offsetY=1`) must be `NotNone`.

Step 5: Extract other data by position. Next, Luke uses the "Position" button again to select the relevant cells for period end date, normal hours worked, and corresponding pay. This generates two additional position-based extract sub-patterns. Luke then modifies their `traverse` and column names. As a result, all required data is successfully extracted into the output table.

Step 6: Fill employee info through Forward. Luke notices many empty values in the first two columns of the output table, shown in output panel of Figure 14. This occurs because the pattern which extracts employee information has different traversal behaviour, leading to mismatched numbers of matched instances compared to other patterns which extract the date and hours attributes show in the last two columns. To prepare the data for downstream querying, Luke sets the `fill` property of the parent pattern to `'Forward'` to fill the empty cells with the preceding value.

7 Critical Reflections

Traditional comparative methods are often ill-suited for evaluating complex interactive systems, especially when there are inconsistencies or differences between the abstractions and underlying assumptions used by other seemingly comparable systems [34]. Given the novel task conceptualisation and grammar-powered design of TableCanoniser, we follow Satyanarayan et al. [34]'s suggestion of providing detailed *critical reflections* instead of attempting to conduct user studies. In the following reflections, we revisit and further clarify the task scope of TableCanoniser relative to other existing tools, reflect on how these distinctions informed key design choices, and discuss opportunities for extending this work.

7.1 Verification Support

As discussed in subsection 3.3, the task support of TableCanoniser may seem at a functional level comparable to existing table processing and spreadsheet wrangling tools, whereby "dirty" spreadsheet inputs are converted into "clean" relational tables. However, even in the overlapping case of axis-aligned inputs, the approach presented in this work addresses important auxiliary tasks that surpass simple "dirty" to "clean" conceptualisations of table canonicalisation. In particular, the TableCanoniser application explicitly supports iterative and interactive refinement and verification of mapping specifications, tasks not directly addressed by existing tools. Rather than simply verifying that an output matches the desired format, i.e., tidy relational table, we directly address the user's need to verify the output contents in terms of specific cell values in the input table (e.g., Demo Case 1b: Step 4'-2, as shown in Figure 11).

The desire to directly support verification strongly influenced the design of our system. As discussed in subsection 3.4, it cannot be assumed that tools that return "clean" relational tables are sufficiently instrumented to provide the provenance traces required for verifying the many possible canonical tables that could be produced from a given messy input spreadsheet. By contrast, in our system, unambiguous linked highlighting depends on cell-to-cell mappings of the form shown in Figure 8 derived from the declarative grammar specification, which was designed to provide not only this bidirectional linkage of cells but also information on the part of the specification that resulted in the transformation.

7.2 Grammar Readability

Readability considerations, both of the declarative grammar and the specification tree visualisation, informed the structure and naming of the `TableCanoniserTemplate` interface. Earlier specification versions did not organise specification properties into the `match`, `extract` and `children` concepts evident in the final implemented grammar in subsection 4.1. The addition of hierarchical structure within the grammar reflected the gradual distinction of our system scope beyond automated "black-box" table canonicalisation towards supporting users to describe and verify that process in terms of meaningful and possibly nested *match* patterns of cells in the messy inputs and *extract* logic information for leveraging those patterns to produce a canonical relational table (e.g., the specification of Demo Case 2 shown in Figure 9).

7.3 Unified Visual Encoding Design

Our work offers three visual encodings to support the use of our declarative grammar as introduced in section 4. In addition to designing each visualisation to support particular verification and transformation subtasks, careful consideration of shared encoding channels such as colour were also required to ensure cohesion across the interface.

Colour was used to encode additional information about how input cells were extracted into output tables across the three different visualisations and the code specification panel. In the mapping tree specification, each node is coloured according to the chosen extract method, or coloured grey if the node is only specifying a matching pattern. The same colour-coding is used on cells in the matching instances mini-map and the input table view to show which methods are used to extract values from particular cells across the entire input spreadsheets, and for a specific matched instance. In the output table view the same colour-coding is used to show where cell values for the selected mapping instances were extracted to, and a lighter shade of the same colour is used to show all other cell values extracted by the logic in the selected tree node. This shared

colour-coding between visualisations can be seen in Figures 12 and 13. Finally, the same colour-coding is also used to highlight the lines of the specification code corresponding to the selected tree node, as shown in Figure 10 (Demo Case 3).

In addition to the shared colour-coding, the same cell-type icons were used across the input-output views and the specification tree to support the use of value type information for verifying mappings effects and constraint logic. For example, consider a misspecified mapping for Demo Case 5, whereby the "Normal Hours" column contained both numeric and string values. This would cause an additional *T* text type icon to be shown next to the # number icon shown above the "Normal Hours" column in Figure 14. The presence of the additional icon indicates to the user that there is an issue in their specification code requiring further investigation and refinement. One possible solution could involve adding a numeric type constraint to a relevant node in the specification tree visualisation, similar to that seen above the grey node in Figure 13.

7.4 Interaction, Execution and Evaluation

Our work attempts to bridge what Hutchins et al. [18] define as the "gulfs of execution and evaluation" in table canonicalisation. Although the declarative grammar and visual encodings provide important building blocks for crossing this gulf. The interactive system is a realisation of the bridge between users obtaining a canonical table from a messy spreadsheet and evaluating whether the output meets their expectations and requirements. Borrowing language from Myers et al. [30], we use visual encodings and interactive affordances to reduce the "threshold" for learning to author mapping specifications, and provide a code panel for users to express transformations directly in the declarative grammar to extend the "ceiling" of the system beyond what can be expressed via interactions alone.

The TableCanoniser application provides several demo inputs and example mapping specifications for users to learn from and adapt for their use cases. We chose demo cases that illustrated various matching patterns expressible in our grammar from relatively simple position- or context-based patterns (e.g., Demo 1a and 1b) to more complex patterns of variable area sizes defined by boundary constraints (e.g., Demo 5). We also provide example specifications for table content patterns commonly found within scientific research papers. This includes cross tabulation patterns similar to Figure 3 (c) (e.g., Demo 6), and spanning headers similar to Figure 4 (d) (e.g., Demo 2).

We attempt to further flatten the learning curve for users by providing direct support for common tasks and logic rather than requiring users to construct their custom functions. For example, the `byPositionToTargetCols` and `byContext` properties introduced in subsection 4.1 could have been left to user implementation via custom functions passed to `byValue`.

7.5 Limitations

TableCanoniser is designed to improve the extraction of information from messy tables with complex but repeating patterns. As shown in Figure 5 (d), spreadsheets that lack discernible patterns are not well-supported. In such cases, it may be more appropriate to use approaches tailored for unstructured text and treat the cells as natural language instead of loosely structured cells. Furthermore, as mentioned in subsection 3.3, there are some table reshaping and value extraction tasks, which although expressible in the TableCanoniser grammar, would more elegantly expressed using functional rules. Consider for example the "Info" column in Figure 5 (b). Extracting the gender and age values into separate columns can easily handled using a string splitting function with "-" as the delimiter argument. However, handling using TableCanoniser would require much a much more verbose specification. Although users could leverage the fixed position range of the cells for matching, they would need to specify a custom string parsing function for extraction. These examples highlight trade-offs in efficiency and parsimony when designing tools for easy-to-index transformation cases, compared to the relative verbosity required to support the verification of more irregular transformation cases as handled by TableCanoniser.

Another subtle limitation of the current grammar is the lack of support for directly excluding areas in the input spreadsheet from being matched. During the development of the system, we found the optional `constraints` property and custom extraction functions passed to `byValue` were sufficiently expressive to address all the demo cases. However, it is conceivable that an input spreadsheet might lack sufficient discriminating features to exclude all undesirable areas based on inclusion constraints, and/or some users might find it helpful or convenient to specify that certain areas should never be matched as pattern instances to extract values from.

8 Future Work

Based on our critical reflections, future work could involve implementing and evaluating additional helper extensions and exclusion-based constrictions. Similarly, we note that our proposed visual encodings and interactive applications are singular instances of grammar-powered tools that could support verifiable table canonicalisation. As such, there is a clear scope for exploration of other implementations of these visualisation and interaction layers and comparative evaluation through user studies and observations in the future.

8.1 Grammar Extensions and Interface Support

Although TableCanoniser supports more complex match and extract operations via the use of JavaScript functions for some properties, these require both programming expertise and a level of familiarity with the workings of the grammar. As such, there scope for defining new operations tailored to messy, non-axis aligned tables. These operations would to reduce the need for users to write custom functions or ad-hoc code. We plan to gather commonly used "match and extract" methods (e.g., `valueBetween` for constraining numerical values) and formalise them as general operators or keywords in future versions of TableCanoniser. Further work could also investigate how to improve the interactive interface to support the authoring of more complex operations.

8.2 Visualisations and Provenance

In future, we plan to explore new designs that provide a clearer and more holistic overview of position mappings. A key aspect of this will be to strengthen the support for verification and trust

through visualisations focusing on tracing the origin of information in the output table and surfacing assumptions in the extraction process. This could involve the visualisation of repeated instances of matches leading to patterns of extracted data, and highlighting the instances that break these patterns to support verification of user assumptions about the input table. Another challenge for future work is designing an alternative mapping instances overview that more accurately conveys more complex and variable match and extract logic across all matched instances. For example, in Demo Case 5 (Figure 14 and subsection 6.2), the position of each sub-region relative to its parent region is not fixed, and the cells with the same positions could be transformed into different columns in the output table depending on the constraints or context/value differences.

8.3 AI-supported Table Transformation

While our grammar is designed to support fine-grained parameter configurations (such as `offsetLayer` and `offsetFrom`) and custom function development, such flexibility makes it difficult to specify complex transformations through interaction alone. However, we believe that large language models (LLMs) could be leveraged to create an alternative, more accessible, interface for authoring specifications in our declarative grammar. For instance, users could provide their messy tables to an LLM primed with knowledge of the TableCanoniser grammar, along with a prompt specifying the variables they would like extracted into their canonical output table. The LLMs would return a suggested specifications, which could then be loaded into TableCanoniser, allowing the user to comprehend and verify the transformation logic and outputs. If the extracted output tables does not meet user expectations, it might also be possible to utilise a transform-by-example approach to refining the prompt and generated output. This could involve the user providing some sample output rows or describing in additional constraints.

8.4 Upstream and Downstream Integrations

There are several opportunities to extend TableCanoniser to support a wider range of workflows through integration with upstream and downstream tools: (1) *Upstream integrations to expand input support.* TableCanoniser could be integrated or used in conjunction with Table Detection and Table Structure Recognition tools to extend support for tractable canonicalisation of tables from print-oriented documents (e.g., PDFs and images) and web pages, in addition to the current support for spreadsheet data. (2) *Downstream integration for holistic workflows.* Table Canonicalisation is only one part of the data preparation and analysis workflow. Integration with existing downstream data wrangling and table normalisation tools could be provide a more seamless experience for users to generate analysis or storage ready tables, which can also be passed into visualisation or analysis applications.

9 Conclusion

This paper has presented several important contributions for table canonicalisation. We introduced the concept of "axis-alignment" to characterise a class of commonly-found input data that our approach can handle, and which is beyond the capabilities of existing tools to easily transform. We have presented a declarative "match-and-extract" grammar that allows users to describe the table transformation process, as well as an open-source software package that handles parsing of the grammar and transformation of tables. We have shown the grammar is simple enough to quickly implement transformations for common types of messy tables, while also being expressive enough to describe more complex transformations required for non-axis-aligned table content. We have presented an interactive visualisation system (also open-source), that allows a person to interactively author the grammar-based transformation specification through interaction with the input data. Importantly, due to our approach of generating the transformation code from the specification, we can allow the user to see the provenance data in the output table and the transformation process, allowing them to understand the steps and operations responsible. This facilitates a user interactively exploring and interrogating the transformation process in order to debug it and ultimately ensure its correctness. Finally, we have demonstrated the usability of the TableCanoniser grammar and interface through two usage scenarios based on real-world data, including one derived from the use case of a data practitioner in an industry setting.

Acknowledgments

The work was supported by National Key R&D Program of China (2022YFE0137800) and an Australian Government Research Training Program (RTP) Scholarship.

References

[1] Daniel W. Barowy, Sumit Gulwani, Ted Hart, and Benjamin Zorn. 2015. FlashRelate: Extracting Relational Data from Semi-Structured Spreadsheets Using Examples. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, Portland OR USA, 218–228. doi:10.1145/2737924.2737952

[2] Lyn Bartram, Michael Correll, and Melanie Tory. 2021. Untidy data: The unreasonable effectiveness of tables. *IEEE Transactions on Visualization and Computer Graphics* 28, 1 (2021), 686–696.

[3] Sara Bonfitto, Elena Casiraghi, and Marco Mesiti. 2021. Table Understanding Approaches for Extracting Knowledge from Heterogeneous Tables. *WIREs Data Mining and Knowledge Discovery* 11, 4 (July 2021), e1407. doi:10.1002/widm.1407

[4] Christian Bors, Theresia Gschwandtner, and Silvia Miksch. 2019. Capturing and Visualizing Provenance From Data Wrangling. *IEEE Computer Graphics and Applications* 39, 6 (Nov. 2019), 61–75. doi:10.1109/MCG.2019.2941856

[5] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (2011), 2301–2309. doi:10.1109/TVCG.2011.185

[6] Steven P. Callahan, Juliana Freire, Emanuele Santos, Carlos E. Scheidegger, Cláudio T. Silva, and Huy T. Vo. 2006. VisTrails: Visualization Meets Data Management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*. ACM, Chicago IL USA, 745–747. doi:10.1145/1142473.1142574

[7] Ran Chen, Di Weng, Yanwei Huang, Xinhuan Shu, Jiayi Zhou, Guodao Sun, and Yingcai Wu. 2023. Rigel: Transforming Tabular Data by Declarative Mapping. *IEEE Transactions on Visualization and Computer Graphics* 29, 1 (2023), 128–138. doi:10.1109/TVCG.2022.3209385

[8] Mike Cheng. 2024. *emphatic: Exploratory Analysis of Tabular Data using Colour Highlighting*. https://coolbutuseless.github.io/package/emphatic/, https://github.com/coolbutuseless/emphatic.

[9] Edgar F. Codd. 1991. *The Relational Model for Database Management: Version 2* (reprinted with corr ed.). Addison-Wesley, Reading, Mass.

[10] Upol Ehsan and Mark O. Riedl. 2024. Explainability Pitfalls: Beyond Dark Patterns in Explainable AI. *Patterns* 5, 6 (June 2024), 100971. doi:10.1016/j.patter.2024.100971

[11] Duncan Garmonsway. 2023. *Tidyxl: Read Untidy Excel Files.*

[12] Ken Gu, Ruoxi Shang, Tim Althoff, Chenglong Wang, and Steven M. Drucker. 2024. How Do Analysts Understand and Verify AI-Assisted Data Analyses?. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. ACM, Honolulu HI USA, 1–22. doi:10.1145/3613904.3642497

[13] Sumit Gulwani, William R. Harris, and Rishabh Singh. 2012. Spreadsheet Data Manipulation Using Examples. *Commun. ACM* 55, 8 (Aug. 2012), 97–105. doi:10.1145/2240236.2240260

[14] Handsontable. 2024. Handsontable is a JavaScript data grid that looks and feels like a spreadsheet. https://handsontable.com/. Accessed: August 2024.

[15] Cynthia A. Huang. 2023. Visualising category recoding and numeric redistributions. arXiv:2308.06535 [cs.HC] https://arxiv.org/abs/2308.06535

[16] Yanwei Huang, Yurun Yang, Xinhuan Shu, Ran Chen, Di Weng, and Yingcai Wu. 2024. Table Illustrator: Puzzle-based Interactive Authoring of Plain Tables. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. ACM, Honolulu HI USA, 1–18. doi:10.1145/3613904.3642415

[17] Vu Hung, Boualem Benatallah, and Regis Saint-Paul. 2011. Spreadsheet-Based Complex Data Transformation. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*. ACM, Glasgow Scotland, UK, 1749–1754. doi:10.1145/2063576.2063829

[18] Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. 1985. Direct Manipulation Interfaces. *Human–Computer Interaction* 1, 4 (Dec. 1985), 311–338. doi:10.1207/s15327051hci0104_2

[19] David Huynh. [n. d.]. OpenRefine. https://openrefine.org. Accessed: August 2024.

[20] Richard Iannone, Joe Cheng, Barret Schloerke, Ellis Hughes, Alexandra Lauer, and JooYoung Seo. 2023. *gt: Easily Create Presentation-Ready Display Tables*. https://CRAN.R-project.org/package=gt R package version 0.9.0.

[21] Zhongjun Jin, Michael R. Anderson, Michael Cafarella, and H. V. Jagadish. 2017. Foofah: Transforming Data By Example. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, Chicago Illinois USA, 683–698. doi:10.1145/3035918.3064034

[22] Sean Kandel, Jeffrey Heer, Catherine Plaisant, Jessie Kennedy, Frank van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, Dominique Brodbeck, and Paolo Buono. 2011. Research Directions in Data Wrangling: Visualizations and Transformations for Usable and Credible Data. *Information Visualization* 10, 4 (Oct. 2011), 271–288. doi:10.1177/1473871611415994

[23] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Vancouver BC Canada, 3363–3372. doi:10.1145/1978942.1979444

[24] Sunnie S. Y. Kim, Elizabeth Anne Watkins, Olga Russakovsky, Ruth Fong, and Andrés Monroy-Hernández. 2023. "Help Me Help the AI": Understanding How Explainability Can Support Human-AI Interaction. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. ACM, Hamburg Germany, 1–17. doi:10.1145/3544548.3581001

[25] O. Kostyleva, V. Paramonov, A. Shigarov, and V. Vetrova. 2022. Towards Comparison of Table Type Taxonomies. In *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*. IEEE, Opatija, Croatia, 1461–1465. doi:10.23919/MIPRO55190.2022.9803520

[26] Guozheng Li, Runfei Li, Zicheng Wang, Chi Harold Liu, Min Lu, and Guoren Wang. 2023. HiTailor: Interactive Transformation and Visualization for Hierarchical Tabular Data. *IEEE Transactions on Visualization and Computer Graphics* 29, 1 (2023), 139–148. doi:10.1109/TVCG.2022.3209354

[27] Peng Li, Yeye He, Cong Yan, Yue Wang, and Surajit Chaudhuri. 2023. Auto-Tables: Synthesizing Multi-Step Transformations to Relationalize Tables without Using Examples. *Proceedings of the VLDB Endowment* 16, 11 (July 2023), 3391–3403. doi:10.14778/3611479.3611534

[28] Lydia R. Lucchesi, Petra M. Kuhnert, Jenny L. Davis, and Lexing Xie. 2022. Smallset Timelines: A Visual Representation of Data Preprocessing Decisions. In *2022 ACM Conference on Fairness, Accountability, and Transparency*. ACM, Seoul Republic of Korea, 1136–1153. doi:10.1145/3531146.3533175

[29] Microsoft. 2024. Monaco Editor. https://microsoft.github.io/monaco-editor/. Accessed: August 2024.

[30] Brad Myers, Scott E. Hudson, and Randy Pausch. 2000. Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction* 7, 1 (March 2000), 3–28. doi:10.1145/344949.344959

[31] Christina Niederer, Holger Stitz, Reem Hourieh, Florian Grassinger, Wolfgang Aigner, and Marc Streit. 2018. TACO: Visualizing Changes in Tables Over Time. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (Jan. 2018), 677–686. doi:10.1109/TVCG.2017.2745298

[32] pandas dev. 2024. pandas - Python Data Analysis Library. https://pandas.pydata.org/. doi:10.5281/zenodo.7741580

[33] Vijayshankar Raman and Joseph M Hellerstein. 2000. *An Interactive Framework for Data Cleaning*. Technical Report UCB/CSD-0-1110. University of California, Computer Science Division (EECS).

[34] Arvind Satyanarayan, Bongshin Lee, Donghao Ren, Jeffrey Heer, John Stasko, John Thompson, Matthew Brehmer, and Zhicheng Liu. 2020. Critical Reflections on Visualization Authoring Systems. *IEEE Transactions on Visualization and Computer Graphics* (2020), 1–1. doi:10.1109/TVCG.2019.2934281

[35] Alexey Shigarov. 2023. Table Understanding: Problem Overview. *WIREs Data Mining and Knowledge Discovery* 13, 1 (Jan. 2023), e1482. doi:10.1002/widm.1482

[36] A. Shigarov, V. Khristyuk, and A. Mikhailov. 2019. TabbyXL: Software Platform for Rule-Based Spreadsheet Data Extraction and Transformation. *SoftwareX* 10 (July 2019), 100270. doi:10.1016/j.softx.2019.100270

[37] Alexey O. Shigarov, Viacheslav V. Paramonov, Polina V. Belykh, and Alexander I. Bondarev. 2016. Rule-Based Canonicalization of Arbitrary Tables in Spreadsheets. In *Information and Software Technologies*, Giedre Dregvaite and Robertas Damasevicius (Eds.). Vol. 639. Springer International Publishing, Cham, 78–91. doi:10.1007/978-3-319-46254-7_7

[38] Kihyuk Sohn, Sifei Liu, Guangyu Zhong, Xiang Yu, Ming-Hsuan Yang, and Manmohan Chandraker. 2017. Unsupervised domain adaptation for face recognition in unlabeled videos. In *Proceedings of the IEEE International Conference on Computer Vision*. 3210–3218.

[39] Sara Steegen, Francis Tuerlinckx, Andrew Gelman, and Wolf Vanpaemel. 2016. Increasing Transparency Through a Multiverse Analysis. *Perspectives on Psychological Science* 11, 5 (Sept. 2016), 702–712. doi:10.1177/1745691616658637

[40] Tableau. 2024. Tableau Prep Builder | Combine, shape, and clean your data faster. https://www.tableau.com/products/prep. Accessed: August 2024.

[41] Trifacta. [n. d.]. Data Wrangling Software and Tools - Trifacta. https://www.trifacta.com/. Accessed: August 2024.

[42] Vue. 2024. Vue.js - The Progressive JavaScript Framework. https://vuejs.org/. Accessed: August 2024.

[43] April Yi Wang, Will Epperson, Robert A DeLine, and Steven M. Drucker. 2022. Diff in the Loop: Supporting Data Comparison in Exploratory Data Analysis. In *CHI Conference on Human Factors in Computing Systems*. ACM, New Orleans LA USA, 1–10. doi:10.1145/3491102.3502123

[44] Xinxin Wang and Derick Wood. 1996. *Xtable : a tabular editor and formatter*. Technical Report. HKUST. https://api.semanticscholar.org/CorpusID:2239168

[45] Hadley Wickham. 2007. Reshaping Data with the Reshape Package. *Journal of Statistical Software* 21, 12 (2007), 1–20. doi:10.18637/jss.v021.i12

[46] Hadley Wickham. 2014. Tidy Data. *Journal of Statistical Software* 59, 10 (2014), 1–23. doi:10.18637/jss.v059.i10

[47] Hadley Wickham, Mara Averick, Jennifer Bryan, Winston Chang, Lucy McGowan, Romain François, Garrett Grolemund, Alex Hayes, Lionel Henry, Jim Hester, Max Kuhn, Thomas Pedersen, Evan Miller, Stephan Bache, Kirill Müller, Jeroen Ooms, David Robinson, Dana Seidel, Vitalie Spinu, Kohske Takahashi, Davis Vaughan, Claus Wilke, Kara Woo, and Hiroaki Yutani. 2019. Welcome to the Tidyverse. *Journal of Open Source Software* 4, 43 (Nov. 2019), 1686. doi:10.21105/joss.01686

[48] Hadley Wickham, Davis Vaughan, and Maximilian Girlich. 2023. *Tidyr: Tidy Messy Data*.

[49] Kai Xiong, Siwei Fu, Guoming Ding, Zhongsu Luo, Rong Yu, Wei Chen, Hujun Bao, and Yingcai Wu. 2023. Visualizing the Scripts of Data Wrangling with SOMNUS. *IEEE Transactions on Visualization and Computer Graphics* 29, 6 (June 2023), 2950–2964. doi:10.1109/TVCG.2022.3144975

[50] Kai Xiong, Zhongsu Luo, Siwei Fu, Yongheng Wang, Mingliang Xu, and Yingcai Wu. 2023. Revealing the Semantics of Data Wrangling Scripts With COMANTICS. *IEEE Transactions on Visualization and Computer Graphics* 29, 1 (2023), 117–127. doi:10.1109/TVCG.2022.3209470

[51] Kai Xu, Alvitta Ottley, Conny Walchshofer, Marc Streit, Remco Chang, and John Wenskovitch. 2020. Survey on the Analysis of User Interactions and Visualization Provenance. *Computer Graphics Forum* 39, 3 (June 2020), 757–783. doi:10.1111/cgf.14035

[52] Bin Yu and Karl Kumbier. 2020. Veridical Data Science. *Proceedings of the National Academy of Sciences* 117, 8 (Feb. 2020), 3920–3929. doi:10.1073/pnas.1901326117