CoWrangler: Recommender System for Data-Wrangling Scripts

Bhavya Chopra, Anna Fariha, Sumit Gulwani, Austin Z. Henley, Daniel Perelman, Mohammad Raza, Sherry Shi, Danny Simmons, Ashish Tiwari
Microsoft
{t-bhchopra,annafariha,sumitg,austinhenley,danpere,moraza,shersh,dsimmons,astiwar}@microsoft.com

ABSTRACT

We present CoWrangler, a real-time data wrangling recommender system, which can recommend the next-best data wrangling operations along with the corresponding human-readable and efficient code snippets to expedite data exploration and wrangling efforts. A key feature of CoWrangler is that it provides explanations for the generated suggestions in the form of data insights, allowing the user to place confidence in the system. Under the hood, CoWrangler relies on intelligent generation of candidate suggestions using program synthesis techniques and ranking of a set of suggestions based on the notion of data quality improvement. We demonstrate how CoWrangler provides a human-in-the-loop data wrangling experience, and helps users make informed data pre-processing decisions, while saving their time and effort.

CCS CONCEPTS

• **Information systems → Data management systems**; *Data cleaning*; • **Software and its engineering** → Domain specific languages; Programming by example.

KEYWORDS

data wrangling, automated suggestions, predictive synthesis

ACM Reference Format:
Bhavya Chopra, Anna Fariha, Sumit Gulwani, Austin Z. Henley, Daniel Perelman, Mohammad Raza, Sherry Shi, Danny Simmons, Ashish Tiwari. 2023. CoWrangler: Recommender System for Data-Wrangling Scripts. In *Companion of the 2023 International Conference on Management of Data (SIGMOD-Companion '23), June 18–23, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3555041.3589722

1 INTRODUCTION

Real-world data is often riddled with data-quality issues, such as missing values, inconsistent/poor formatting, and duplicate entries. Data wrangling is an iterative process that involves data preparation for analysis by imputing missing values, performing appropriate typecasting and encoding, extracting features, removing duplicates, and so on. Data scientists spend up to 80% of their time in data extraction and preparation—continuously assessing and improving data quality by applying and validating chains of transformations—making wrangling a tedious and error-prone process [7].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD-Companion '23, June 18–23, 2023, Seattle, WA, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9507-6/23/06…$15.00
https://doi.org/10.1145/3555041.3589722

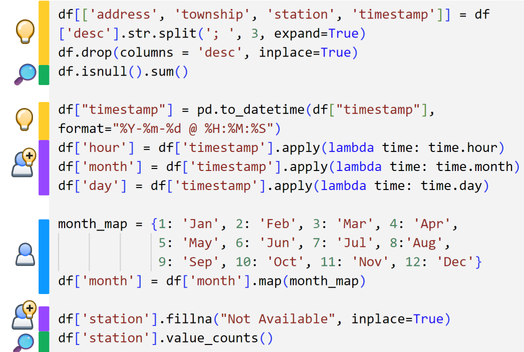

Figure 1: Snapshot of Liam's code. CoWrangler can generate wrangling code (), and eliminate the need for exploratory code () with data insights. CoWrangler adopts a HITL approach for custom operations () and modified suggestions ().

Several libraries and interface designs allow effective data exploration and wrangling. However, the onus of writing correct code and learning specific tools still lies with the data scientist. Organizations strive to be data-driven with the advent of big data, and data analysis is performed by a wide range of users—from programmers to low-code and no-code professionals in management roles—to propel business decisions. Several industry products enable visual data preparation with pre-built transformations and wrangling suggestions. However, the capabilities of such tools are limited due to the lack of human-readable code, which is essential to customize transformations, gain visibility, and replicate results across runs.

Example 1.1. Consider a 911 emergency calls dataset (Fig. 2), where the `desc` attribute contains call information—address, township, station, and timestamp—separated by semicolons. Liam, a data scientist, wants to prepare the data for temporal analysis, and tries to split `desc` with ';' as the delimiter using Python (`pandas`).

He believed he knew how to split a string column, but it takes 4 tries and a web search for Liam to find the correct API and parameters. Next, he casts the derived attribute `timestamp` to DateTime format, and subsequently derives attributes: `day`, `month`, and `hour`. Liam now checks for missing values and realizes that missing values in the derived attribute `station` must be imputed. Fig. 1 displays the 11 lines of wrangling code authored by Liam in 38 minutes. □

Building a tool to ease and accelerate the task of data preparation is challenging. First, the tool needs to generate human-readable and efficient code because the user would want to read, understand, and edit the code to achieve the intended transformation in their desired development environment. Moreover, readable code promotes trust and transparency in the tool's working. While it is often possible to generate working code, it is challenging to generate code that the user would have preferred writing themselves. Second, the space

of possible code snippets is enormous. For instance, the pandas library has a wide breadth of an order of 100 methods, where each method has an order of 10 arguments. Data scientists (as seen in Example 1.1) find it increasingly difficult to recall the correct API method and its arguments. It is computationally intensive to search the space of possible programs to find relevant ones [1]. Finally, there are often many valid steps that can be presented as recommendations to expedite the wrangling process, and it is challenging to pick the most relevant snippets.

Related Work. Visual data preparation services like Mito[1], Bamboolib[2] and Databricks[3] fit in data scientists' workflows to aid data exploration and wrangling. Commercial tools, such as Google Cloud DataPrep (Trifacta)[4], AWS Glue DataBrew[5], Einstein Discovery[6], and others [4, 6] attempt to alleviate data wrangling barriers for low-code and no-code users with visual interfaces and automated recommendations. However, due to the absence of code (no-code) for the applied wrangling steps, these tools prevent data scientists from: (1) customizing suggested transformations, which is necessary to achieve the desired effect, (2) trusting suggestions as there is no explanation towards why a suggestion is proposed, and (3) saving/replicating interleaved workflows in their desired development environment. Data scientists commonly use languages such as Python, R, and SQL to author wrangling code, making it difficult for no-code suggestions to match their complete intent.

The need to accelerate data wrangling is also illustrated by code-first recommendations for transformations such as data reshaping (join, pivot, unpivot), extraction, and error detection [9]. Recent work attempts to capture the user's intent in natural language (NL) to synthesize code using large language models (LLMs) [5]. However, to provide NL prompts, the user must know the exact steps and have sufficient knowledge of the data.

These limitations motivate the development of CoWrangler, a tool that aims to leverage the best of the three dimensions identified in prior work: (1) visual data preparation interfaces, (2) automated wrangling suggestions, and (3) code-first paradigm. CoWrangler presents meaningful and most-relevant wrangling suggestions with explanations, that not only automate, but also inform the wrangling process with a human-in-the-loop (HITL) approach.

The key properties of CoWrangler are: (1) it exploits the user's data context to automatically generate data wrangling suggestions using predictive program synthesis, along with explanations, (2) it is a visual and code-first tool that generates human-readable and efficient code for each wrangling suggestion, and (3) it ranks suggestions based on the principle that the wrangling step that improves the data quality more is more desirable. Consequently, CoWrangler provides users flexibility to use the suggested code as is or customize the suggested code as desired.

2 SOLUTION SKETCH

CoWrangler generates wrangling suggestions for the user based on the data in their working set. We assume we have access to (a sample of) the dataset. We internally learn programs in a domain-specific language (DSL). Each program transforms tables (2D data grid) into new tables. Unlike program synthesis by example where users provide example output [2], we synthesize program transformations just based on the input table (without any user prompt), which is called *predictive program synthesis* [8]. The synthesized transformations are internally applied over the input table, and then ranked based on their capability to improve the quality of the transformed table. Top-ranked programs are then translated to the user's target programming language and suggested as next possible wrangling steps.

2.1 Suggestion DSL

We analyzed 730 Python notebooks from Kaggle to identify and include the most frequently occurring data wrangling operators in the suggestions DSL for CoWrangler:

(1) **Drop column**: Indicates which columns to drop and the *reasons*; e.g., the column is mostly empty, has a constant value throughout, is a redundant column, or is an index column.
(2) **Drop row**: Indicates whether to drop certain rows and the *reasons*; e.g., the row is mostly empty, or is a redundant row.
(3) **Fill missing values**: Indicates the column with missing values, the markers for the missing values (e.g., NaN), and a replacement value (e.g., -1) or a mechanism to impute the missing values (e.g., mean, mode, or median).
(4) **Encoding**: Indicates the column that can be either label-encoded or one-hot encoded.
(5) **Split**: Indicates the column whose string values can be split to create multiple new columns with homogeneous data values.
(6) **Type cast**: Indicates that the data in the identified column can be cast to the identified data type (datetime, integer, float, boolean, or categorical).

CoWrangler is designed to be modularly extensible, and we envision extending the suggestions DSL with many more operators.

2.2 Predictive Synthesis of Suggestions

Given an input table, the synthesis problem is to generate programs in the above DSL that denote meaningful data-wrangling steps, which can be applied on the input table. We solve this synthesis problem by creating separate *individual learners* for each of the top-level DSL operators described above. Each learner analyzes the input table and determines if the corresponding operation can be applied to the input table, and if so, it determines the best values for the various arguments of that operator. We finally collect programs learned by individual learners and rank them.

Certain individual learners are simple. For example, the drop row (or drop column) learner detects if the input table contains rows (columns) that are duplicates of another row (column). It similarly checks if the input table has mostly empty rows (columns). If either is true, then it generates the corresponding DSL program. We also have some complex learners. The "type cast learner", for example, finds all possible types that are consistent with values in a column and then uses complex logic to disambiguate and pick a fixed type for that column. The "split learner" analyzes the strings in a column to find delimiting characters or strings that can be used to generate consistent splits across all column values [8].

[1] trymito.io
[2] bamboolib.8080labs.com
[3] databricks.com/product/data-lakehouse
[4] docs.trifacta.com/display/SS/Overview+of+Predictive+Transformation
[5] docs.aws.amazon.com/databrew/latest/dg/
[6] help.salesforce.com/s/articleView?id=sf.bi_edd_prep_terminology.htm

2.3 Ranking using Data Quality Metric

Given a set of possible data wrangling operations learned in the above step, we next rank them to identify the most relevant operations to surface to the user. We rank by assigning a score to each operation that measures improvement in terms of *data quality* of the transformed table obtained by applying that operation over the input table. When computing data quality, we penalize missing values and redundancy, while giving credit for uniformity in the form of data. Thus, data quality is high if the concrete data has high entropy (less redundancy), but its abstraction (as patterns or types) has low entropy (uniformity). While we use a certain data quality metric in this work, this is a pluggable unit and CoWrangler can work with any user-defined notion of data quality.

2.4 Translation

The top-ranked DSL programs are translated to a user-specified target language. We focus on Python (pandas), but CoWrangler can support translations to other languages such as PySpark and Power Query M. The translation generates the most performant code (using vector APIs wherever possible) and uses comments and meaningful variables to improve code readability.

2.5 Preliminary Evaluation

We performed an initial evaluation over 730 notebooks obtained from Kaggle, consisting of 2248 transformations using pandas APIs. We found that CoWrangler's DSL vocabulary supports **33%** (742 *of* 2248) transformations and CoWrangler's suggestions accurately predict **53.4%** (396 *of* 742) of the supported transformations.

3 DEMONSTRATION

We demonstrate CoWrangler on a real-world dataset of emergency 911 calls from Montgomery County, Pennsylvania[7]. It contains several attributes including `latitude`, `longitude`, `desc`, `zip`, `title`, and `emergency`. Here, the user wants to pre-process the dataset to identify associations between variables, such as: reason of emergency, location, time, day of the week, and so on. Fig. 2 shows the user's data wrangling journey with CoWrangler. Below we describe the demonstration scenario based on CoWrangler's user interface in Fig. 3:

• **Uploading the data:** The user first uploads a subset of the emergency 911 calls dataset Ⓐ to CoWrangler.

• **View wrangling suggestions:** Once the data is loaded Ⓑ, CoWrangler displays wrangling suggestions in natural language Ⓒ. CoWrangler generates 4 different types of suggestions: (1) & (2) split column using delimiter, (3) drop column, (4) fill missing values, and (5) label-encode. Upon observing the suggestions and their explanations, the user immediately identifies important characteristics of the data: (1) missing values in `zip`, (2) categorical nature of `title`, (3) common pattern across `desc` and `title`, and (4) `emergency` being a constant-valued column. This provides data insights and a quick and informed assessment of suggestions, along with identification of other actionable wrangling steps.

• **Select and preview a suggestion:** The user finds the first suggestion, Split `title` using delimiter **colon (:)**, relevant for feature

[7]https://www.kaggle.com/datasets/mchirico/montcoalert

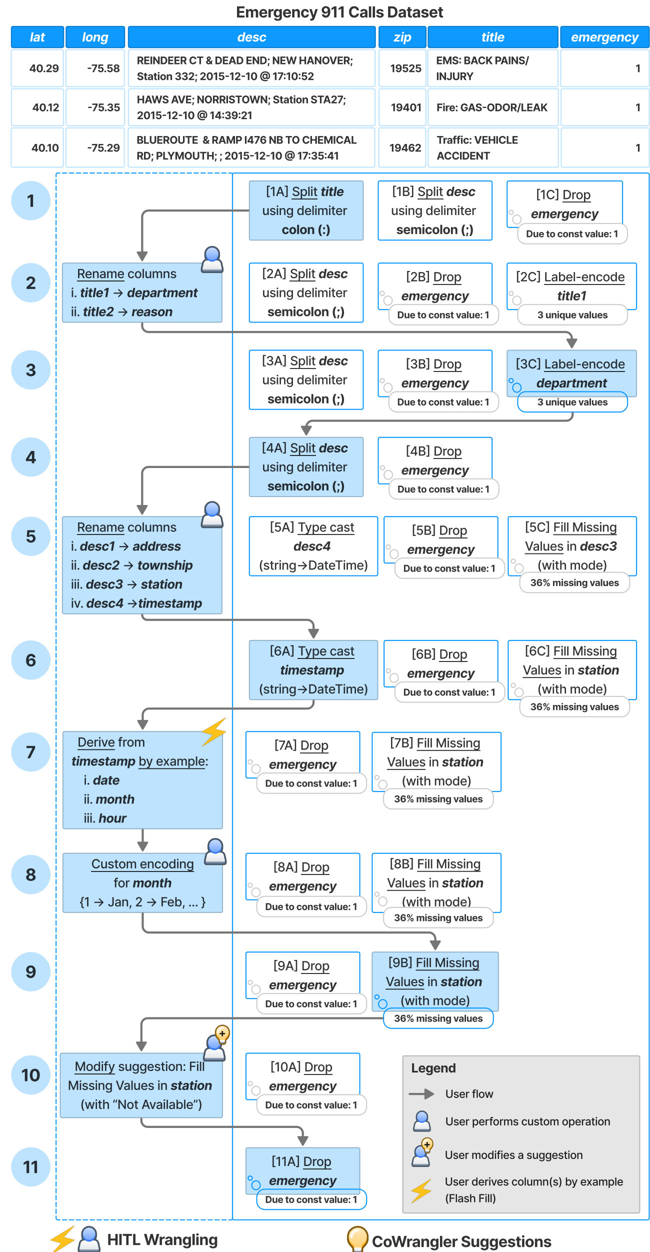

Figure 2: A user's 11 step journey with CoWrangler to prepare their dataset for visualization tasks. Arrows indicate the user flow, with applied transformations highlighted in blue. For each step (row), the first column shows HITL actions (if any), while the second column shows an ordered list (left to right) of CoWrangler suggestions.

extraction and selects it. CoWrangler then generates code in the selected language (Python–pandas) Ⓓ and presents a preview of the two new derived columns (`title1` and `title2`) denoted in green Ⓔ. Users can validate the transformation by looking at the to-be-transformed data preview (supporting low-code/no-code users) and the generated code (supporting data scientists and experienced programmers) before accepting it.

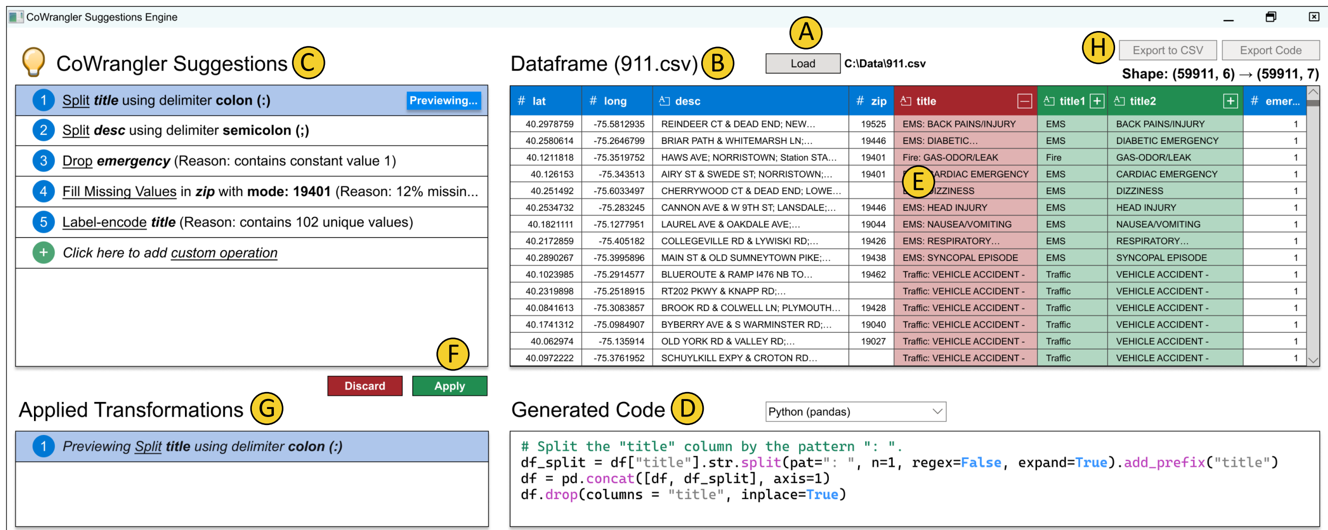

Figure 3: Demonstrating CoWrangler: Ⓐ Upload reference data; Ⓑ 911 calls data is loaded into the data preview panel; Ⓒ CoWrangler populates the suggestions pane with wrangling suggestions. The user selects the first suggestion, highlighted in blue; Ⓓ Human-readable pandas code appears in the 'Generated Code' panel; Ⓔ Data preview panel highlights changes that will be made if the suggestion is accepted; Ⓕ Click the 'Apply' button to accept the transformation being previewed; Ⓖ Wrangling history is tracked in the 'Applied Transformations' panel; Ⓗ Export the resulting CSV file and wrangling code.

• **Accept a suggestion:** Once convinced, the user clicks 'Apply' to accept the suggested transformation Ⓕ, which then reflects in the 'Applied Transformations' pane Ⓖ. CoWrangler then refreshes the data preview Ⓑ and the suggestions pane Ⓒ (not visible in Fig. 3), populating it with suggestions for the updated data.

• **Human-in-the-loop wrangling:** The user now wants to rename the newly derived columns: `title1` to `department`, and `title2` to `reason`. To achieve this, the user selects 'Click here to add custom operation' from the suggestions pane Ⓒ and writes the desired code using the `pandas rename` API in the code editor Ⓓ. The user then applies their custom operation to the dataset, which gets appended to the 'Applied Transformations' pane Ⓖ.

Users can collaborate with CoWrangler in three different modes to customize wrangling transformations as desired (Fig. 2): (1) editing the generated code while previewing (steps 9–10), (2) expressing intent by providing examples with Flash Fill [3] (step 7), and (3) authoring code from scratch (steps 2, 5, and 8).

• **Select and preview subsequent suggestion:** When the data preview and suggestions get refreshed Ⓑ & Ⓒ, a new suggestion, Label-encode `department` (REASON: contains 3 unique values), appears and the user selects this suggestion. They validate the transformation by previewing the data and the generated code.

• **Accept subsequent suggestion:** The user accepts the label-encode suggestion, which is appended to the 'Applied Transformations' pane Ⓖ. CoWrangler again refreshes the data preview Ⓑ and wrangling suggestions Ⓒ and the above processes go on.

• **Export generated code and wrangled data:** After making the desired transformations, the user can export the transformed data to a CSV file and export the code to a notebook Ⓗ.

Demonstration engagement. Following our demonstration, participants can load their own datasets into CoWrangler to receive automated wrangling suggestions depending on their data context. They can then preview the effects of different suggestions and make modifications to the generated code to fulfil their requirements.

4 CONCLUSIONS AND FUTURE WORK

This demonstration is our attempt to inform and accelerate the data wrangling process by (1) recommending the most meaningful transformations with explanations in real-time, (2) generating human-readable and efficient code, and (3) enabling flexibility with HITL interactions for intent expression. In future, we aim to extend CoWrangler's vocabulary to increase the coverage of suggestions, and add capabilities to understand data semantics for more contextualized suggestions.

REFERENCES

[1] R Bavishi, C Lemieux, R Fox, K Sen, and I Stoica. 2019. AutoPandas: Neural-Backed Generators for Program Synthesis. *Proc. ACM Program. Lang.* OOPSLA, Article 168 (oct 2019), 27 pages.

[2] A Fariha and A Meliou. 2019. Example-Driven Query Intent Discovery: Abductive Reasoning using Semantic Similarity. *PVLDB* 12, 11 (2019), 1262–1275.

[3] S Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *POPL*. 317–330.

[4] P J Guo, S Kandel, J Hellerstein, and J Heer. 2011. Proactive Wrangling: Mixed-Initiative End-User Programming of Data Transformation Scripts. In *UIST*. 65–74.

[5] N Jain, S Vaidyanath, A Iyer, N Natarajan, S Parthasarathy, S Rajamani, and R Sharma. 2022. Jigsaw: Large Language Models Meet Program Synthesis. In *ICSE*. 1219–1231.

[6] S Kandel, A Paepcke, J Hellerstein, and J Heer. 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *CHI*. 3363–3372.

[7] G Press. 2016. Cleaning Big Data. https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/.

[8] M Raza and S Gulwani. 2017. Automated data extraction using predictive program synthesis. In *AAAI*, Vol. 31.

[9] C Yan and Y He. 2020. Auto-Suggest: Learning-to-Recommend Data Preparation Steps Using Data Science Notebooks. In *SIGMOD*. 1539–1554.