Towards Scalable Visual Data Wrangling via Direct Manipulation

El Kindi Rezig
University of Utah
elkindi.rezig@utah.edu

Mir Mahathir Mohammad
University of Utah
mahathir.mohammad@utah.edu

Nicolas Baret
University of Utah
nicolas.baret@utah.edu

Ricardo Mayerhofer
Hopara, Inc
ricardo@hopara.io

Andrew McNutt
University of Utah
andrew.mcnutt@utah.edu

Paul Rosen
University of Utah
paul.rosen@utah.edu

ABSTRACT

Data wrangling—the process of cleaning, transforming, and preparing data for analysis—is a well-known bottleneck in data science workflows. Existing tools either rely on manual scripting, which is error-prone and hard to debug, or automate cleaning through opaque black-box pipelines that offer limited control. We present Buckaroo, a scalable visual data wrangling system that restructures data preparation as a direct manipulation task over visualizations. Buckaroo enables users to explore and repair data anomalies—such as missing values, outliers, and type mismatches—by interacting directly with coordinated data visualizations. The system extensibly supports user-defined error detectors and wranglers, tracks provenance for undo/redo, and generates reproducible scripts for downstream tasks. Buckaroo maintains efficient indexing data structures and differential storage to localize anomaly detection and minimize recomputation. To demonstrate the applicability of our model, Buckaroo is integrated with the *Hopara* pan-and-zoom engine, which enables multi-layered navigation over large datasets without sacrificing interactivity. Through empirical evaluation and an expert review we show that Buckaroo makes visual data wrangling scalable—bridging the gap between visual inspection and programmable repairs.

1 INTRODUCTION

The promise of data-driven decision-making relies critically on the quality and readiness of underlying datasets [5]. Yet, before any analysis, modeling, or visualization can occur, practitioners must invest substantial effort into *data wrangling*—the process of transforming raw, messy data into a structured form suitable for downstream tasks. Despite its importance, data wrangling remains one of the most labor-intensive and error-prone phases of the data science lifecycle, accounting for up to 80% of the total project time [24].

Data wrangling involves a wide range of tasks, including parsing, deduplication, missing value imputation, anomaly detection, and type normalization. These tasks are often executed through manual scripts or ad-hoc spreadsheet operations, introducing two significant problems: (1) errors introduced during wrangling are difficult to detect and propagate silently into downstream models, and (2) the lack of interactivity and visibility in existing tools limits user understanding and trust, especially when dealing with complex group-level anomalies.

Bridging the gap between iterative scripting and visual exploration is essential. Data wrangling is rarely a one-shot operation—it is inherently iterative, involving trial-and-error, context-aware corrections, and domain judgment [30]. While prior work has proposed automated or script-based approaches to support cleaning operations [10], these systems often assume a fixed pipeline and offer little visibility into how each transformation affects the dataset, i.e., fixing one error might lead to new errors in the dataset. This limits their usability in real-world settings where errors are heterogeneous, subtle, and contextual.

Conversely, visualization tools excel at surfacing structure [26, 31], guiding attention, and enabling human reasoning, but they are typically detached from the wrangling process itself. That is, they are used post-hoc for validation rather than as integral components of repair. This disconnect leads to workflows where users must constantly switch between error inspection and editing scripts to fix problems in the data, which is laborious and error-prone. Our proposed system, Buckaroo, bridges this gap by reimagining data wrangling as a visual, interactive, and iterative process. It allows users to identify and resolve data anomalies by directly manipulating visual representations of data groups. Buckaroo automatically detects anomalous groups—e.g., those with missing values or outliers—maps them to interactive charts, and offers recommended repairs that can be applied, visualized, and reverted in real time. This tight integration of detection, visualization, and repair enables users to understand the impact of each action across the dataset and supports the inherently exploratory nature of data preparation.

Transforming visualizations into repair interfaces. A key technical insight of Buckaroo is to treat visualizations not merely as passive outputs but as active substrates for user-driven data transformation. By constructing index structures that link anomalies to data groups and anomaly types, Buckaroo enables responsive, bidirectional interactions: users can trigger repairs through visual selections and observe the systemic consequences instantly. Moreover, through user-defined detector and repair functions, the system accommodates domain-specific anomalies while preserving flexibility and reproducibility through automatic code generation.

Motivating Example. Consider the table in Figure 1, which shows two groups from a larger dataset: $G1$ = {`Income | Country =` "Bhutan"} and $G2$ = {`Income | Degree = "BS"`}. Both groups have `Income` anomalies, including outliers, missing values, and inconsistent data types.

Lou, a data scientist, is tasked with preparing this dataset for downstream analysis. Using a traditional workflow—such as importing the data into Python—Lou encounters several challenges:

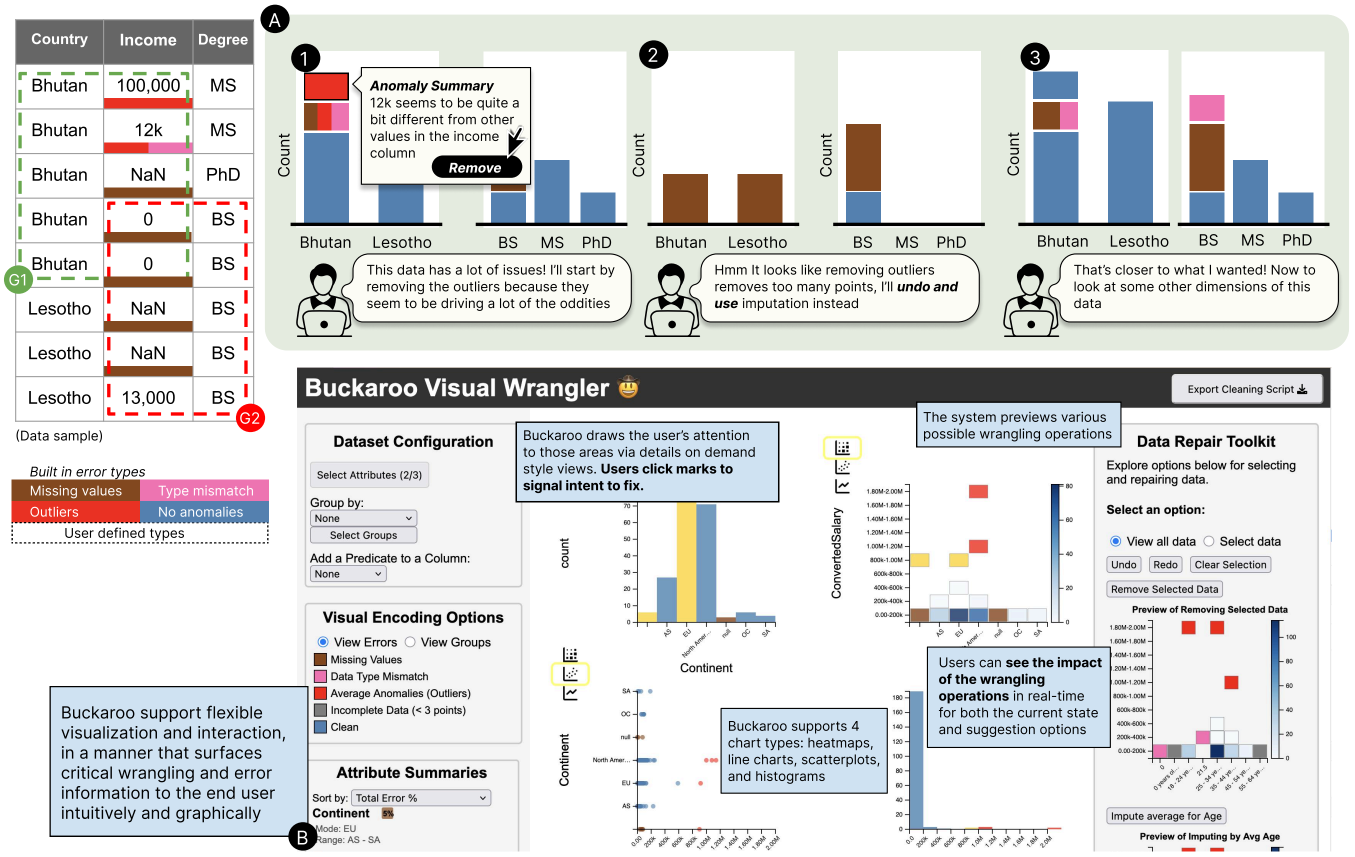

Country Income Degree
Bhutan 100,000 MS
Bhutan 12k MS
Bhutan NaN PhD
Bhutan 0 BS
Bhutan 0 BS
Lesotho NaN BS
Lesotho NaN BS
Lesotho 13,000 BS
G1
G2
(Data sample)
Built in error types
Missing values
Type mismatch
Outliers
No anomalies
User defined types
A
1
2
3
Count
Bhutan
Lesotho
BS
MS
PhD
Anomaly Summary
12k seems to be quite a bit different from other values in the income column
Remove
This data has a lot of issues! I'll start by removing the outliers because they seem to be driving a lot of the oddities
Hmm It looks like removing outliers removes too many points, I'll undo and use imputation instead
That's closer to what I wanted! Now to look at some other dimensions of this data
Buckaroo Visual Wrangler
Export Cleaning Script
Dataset Configuration
Select Attributes (2/3)
Group by:
None
Select Groups
Add a Predicate to a Column:
None
Buckaroo draws the user's attention to those areas via details on demand style views. Users click marks to signal intent to fix.
The system previews various possible wrangling operations
Data Repair Toolkit
Explore options below for selecting and repairing data.
Select an option:
View all data
Select data
Undo
Redo
Clear Selection
Remove Selected Data
Preview of Removing Selected Data
Visual Encoding Options
View Errors
View Groups
Missing Values
Data Type Mismatch
Average Anomalies (Outliers)
Incomplete Data (< 3 points)
Clean
count
Continent
ConvertedSalary
Users can see the impact of the wrangling operations in real-time for both the current state and suggestion options
Buckaroo support flexible visualization and interaction, in a manner that surfaces critical wrangling and error information to the end user intuitively and graphically
Buckaroo supports 4 chart types: heatmaps, line charts, scatterplots, and histograms
Attribute Summaries
Sort by: Total Error %
Continent
Mode: EU
Range: AS - SA
Impute average for Age
Preview of Imputing by Avg Age
B

Figure 1: An overview of repairing an error through Buckaroo's user interface. (A) highlights a user working through iterative and backtrack-laden process of cleaning a dataset. (B) shows the full interface for a sample of the StackOverflow dataset. Each error type has a distinct color (e.g., red groups correspond to average anomalies), Upon selecting the group, Buckaroo shows a list of wrangling/repair actions on the right and shows a visual preview of the chart after the repair.

Sparse anomalies: Errors are scattered and infrequent, making them hard to detect. **Interdependent groups:** Fixing an anomaly in one group may unintentionally distort others. For instance, removing all zero-income rows from *G1* could leave *G2* with insufficient data. **Iterative debugging:** Writing and refining data cleaning scripts requires multiple trial-and-error cycles to validate correctness and completeness [13].

By contrast, if Lou uses Buckaroo (Figure 1 (A)), the system automatically highlights anomalous groups for inspection. Lou is presented with wrangling suggestions specific to each error type—such as imputation, deletion, or conversion—and can apply these repairs interactively. Crucially, the visual interface reveals how each action affects related groups in real time, allowing Lou to iteratively explore, evaluate, and undo changes as needed. Once the data is in a satisfactory state, Buckaroo can export a Python script encoding all the wrangling steps for future reuse or automation.

A prototype of Buckaroo was demonstrated at VLDB 2025 [37]. The demo version operates entirely in client-side memory and is intended for smaller datasets. While suitable for showcasing interaction design, it lacks the scalability features required for real-world deployment. In this paper, we present our ongoing efforts to scale Buckaroo by introducing server-side storage, differential snapshot management, and efficient update propagation—making the system scalable and practical for large, real-world datasets. Throughout the paper, we use the terms "anomaly" and "error" interchangeably, as well as "wrangling" and "repair."

Contributions. Buckaroo introduces a new paradigm for data wrangling by tightly integrating anomaly detection, visual exploration, and repair within a single interactive interface. This paper makes the following contributions:

- A group-based abstraction that organizes anomalies into interpretable visual summaries, enabling users to interact with data through an orchestrated set of interactive charts.
- An extensible framework for registering custom error detectors and repair functions, allowing domain-specific wrangling logic to be incorporated.
- Efficient indexing and overlap-tracking structures that support localized, low-latency anomaly detection and visualization updates across interdependent views.
- A snapshot storage module that enables undo/redo actions and code generation, while maintaining scalability over large datasets.

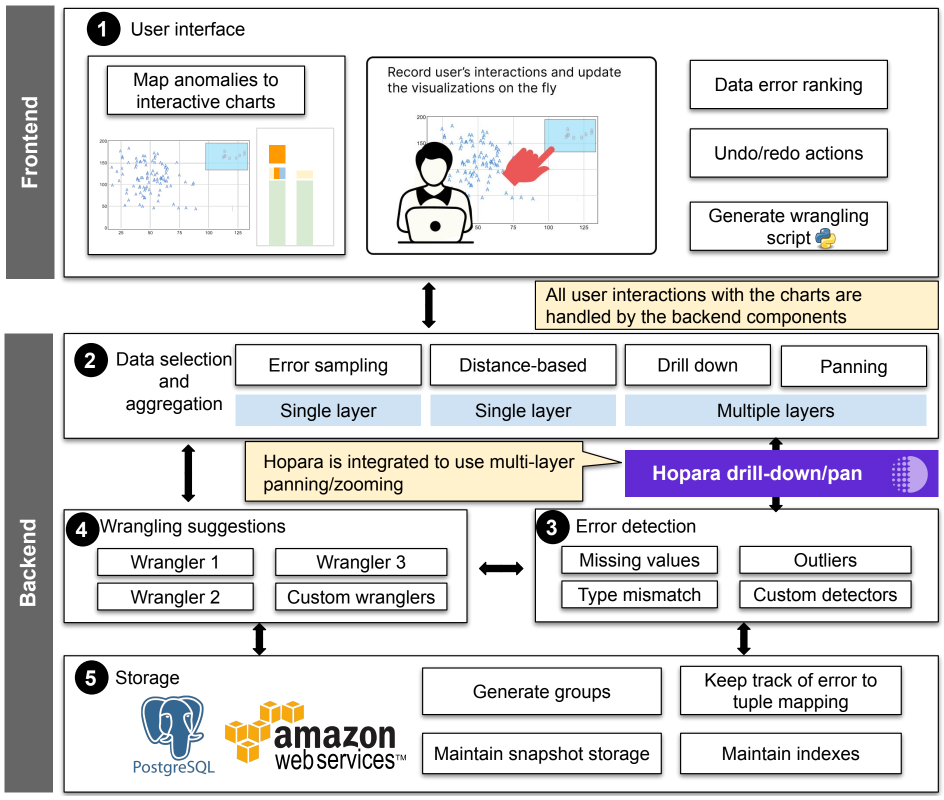


Figure 2: Buckaroo system architecture. ❶ The user interface visualizes anomalies and supports interactive exploration, ranking errors, undo/redo, and script generation. ❷ The data selection and aggregation layer provides single-layer and multi-layer navigation through error sampling, distance-based sampling, drill-down, and panning. ❸ The backend implements built-in and custom error detectors. ❹ Buckaroo offers built-in and user-defined wrangling functions. ❺ The storage layer manages group generation, snapshot storage for undo/redo, error–tuple mappings, and indexes using PostgreSQL and AWS infrastructure.

By enabling users to *see*, *understand*, and *repair* anomalies through a single, unified visual interface, Buckaroo represents a paradigm shift in how practitioners wrangle data. It transforms data wrangling from a brittle and opaque task into an intuitive, transparent, and reproducible process.

2 SYSTEM OVERVIEW

Buckaroo is a visual data wrangling system that couples anomaly detection, visualization, and guided repair through a direct manipulation interface. Figure 2 illustrates the overall architecture, comprising five components that span frontend interactions and backend processing. To support large datasets, Buckaroo manages all data storage and access through a Postgres backend.

The workflow begins when a user uploads a tabular dataset through the user interface (UI) as illustrated in Figure 2 ❶. Buckaroo then stores the data into a Posgres database and generates groups by projecting numerical attributes onto categorical attributes (Figure 2 ❺). The database also stores metadata linking each tuple to its associated errors. For each group, built-in or user-defined detectors are used to identify anomalies such as missing values, outliers, or type mismatches (Figure 2 ❸). These anomalies are visualized through interactive charts, where users can inspect and select problematic groups. Since plotting every data point is impractical, Buckaroo employs data selection and aggregation strategies to determine which subset of the table to visualize (Figure 2 ❷). Based on the anomaly type, Buckaroo presents corresponding wrangling suggestions—both default and user-defined—that users can apply directly to the chart (Figure 2 ❹).

As users manipulate the data visually, the system tracks changes, re-runs localized detection only on affected groups, and updates all impacted views efficiently. All user actions are logged, and a differential snapshot mechanism ensures storage efficiency and supports undo/redo functionality (Figure 2 ❺). Buckaroo also creates Postgres indexes for all the attribute combinations in the charts for efficient data lookups. Finally, once the user is satisfied with the cleaned data, Buckaroo generates an executable Python script that captures the full sequence of wrangling operations for reuse or automation. Currently, Buckaroo only generates Python scripts, but we intend to support other target languages such as R. We now describe the major components of Buckaroo.

2.1 Group Generation

In Buckaroo, groups serve as the fundamental abstraction for detecting and visualizing anomalies. A group is defined as a subset of the dataset obtained by projecting a numerical attribute (e.g., `Income`) onto a categorical attribute (e.g., `Country`). For example, the group {`Income | Country = "Bhutan"`} corresponds to the set of `Income` values for all records where the country is "Bhutan". This group, along with others defined by different country values, can be visualized in a chart such as a heatmap with `Country` on the X-axis and `Income` on the Y-axis. Users can control this process by selecting the projection columns and adjusting granularity (e.g., setting a minimum group size). Using group-based analysis, rather than inspecting individual rows, offers several benefits:

- **Summarization:** Groups compress many data points into coherent aggregates, making it easier to detect outliers, missing values, or irregular patterns at a glance.
- **Isolation within attributes:** Grouping has long been used to isolate error detection and repair (e.g., blocking [10] and subgroup discovery [8]). Groups defined over a single categorical attribute are disjoint—each row belongs to exactly one group per attribute. This means that repairing an anomaly in one group (e.g., `Country = Bhutan`) does not require updates to other groups using the same attribute with a different value.

However, as in the motivating example, groups defined over different attributes can overlap, since a single row may belong to multiple groups across multiple charts. Buckaroo tracks these dependencies and selectively updates only affected groups when a repair is made (more details in Section 3.3).

2.2 Interactive User Interface

Buckaroo generates a chart matrix (see cropped view in Figure 1 B) where data groups are represented in a heat map. Detected anomalies are visually overlaid across chart types—scatterplots, histograms, heatmaps—with groups color-coded by their dominant anomaly type. Users can interactively explore, filter, and manipulate these groups directly through the visual interface.

Buckaroo records all user actions—such as applying a repair, exploring a group, or undoing a prior fix—and communicates them to the backend to maintain a synchronized and consistent snapshot. The UI also displays ranked anomaly (based on their frequency in the data) summaries and offers a repair kit sidebar to surface appropriate wrangling options for selected groups. The main features of the UI are as follows:

- **Dynamic anomaly mapping:** Buckaroo continuously overlays detected errors onto the corresponding chart elements, visually encoding their severity and type. This visual contextualization allows users to spot data anomalies across groups at a glance and understand how errors are distributed throughout the dataset before initiating repairs.
- **Immediate feedback:** When a user applies a wrangling operation, all affected charts and summaries update instantly to reflect the modified data. This tight feedback loop helps users reason about both the local and global consequences of their actions, preventing unintended distortions and supporting rapid trial-and-error exploration.
- **Iterative editing:** Every transformation—whether a value imputation, deletion, or type correction—is logged and reversible. Users can freely undo or redo prior steps, enabling a flexible and exploratory workflow in which they can test alternative repair strategies without committing prematurely or losing previous progress.
- **Script generation:** After users reach a satisfactory data state, Buckaroo compiles the full sequence of wrangling actions into a Python script. This script preserves provenance, supports reproducibility, and allows users to integrate their visually authored cleaning pipeline into downstream analytical workflows.

3 ERROR DETECTION AND WRANGLING

Buckaroo supports both generic and domain-specific error detection to surface data issues during interactive wrangling. For each detected error type, Buckaroo provides corresponding wranglers that can be applied directly through the interactive charts.

3.1 Error Detection

Built-in detectors identify common anomalies such as missing values, outliers, type mismatches, and small groups (groups containing few points). However, data quality is often domain-dependent [10], requiring customized logic. To address this, Buckaroo offers an extensible API through which users can define their own detectors that operate at the group level, enabling flexible and reusable domain-specific validation.

Built-in Error Types. Buckaroo supports the following built-in error types:

- **Missing Values:** Identifies null or undefined cells within groups.
- **Outliers:** Flags values outside a configurable threshold (e.g., 2 standard deviations from the global mean).
- **Type Mismatches:** Detects non-numeric entries in numeric columns (e.g., "12k" in a salary field).
- **Group Incompleteness:** Marks groups with cardinality below a minimum threshold.

In the current Buckaroo prototype, built-in error detectors are implemented as SQL queries, allowing them to run directly on the underlying database.

Custom Error Detectors. Users can register domain-specific detectors via a simple API. A detector is a function that receives a group and target attribute and returns a list of anomalous tuples. Each custom detector is mapped to a unique error code. This extensibility allows domain experts to define tailored quality checks (e.g., clinical code mismatches or sensor dropouts). For instance, the following custom detector detects if an income is less than 0:

```python
def custom_detector(df: pd.DataFrame = None,
                        target_column: str = "",
                        error_type_code: str = "") -> list
                            :
    if error_type_code == "negative_income":
        if df is None:
            # Write SQL query to detect the error
            query = f"SELECT id FROM salary_table
                WHERE {target_column} < 0"

            return sys.get_row_ids(query)
```

The detector returns a list of row indices corresponding to tuples exhibiting a specified error type. As shown in the listing above, certain errors—such as negative values—can be efficiently detected using a SQL query. However, not all error types are expressible in SQL [13]. To accommodate such cases, Buckaroo supports an optional Pandas DataFrame input, allowing detectors to operate directly on in-memory data when SQL alone is insufficient.

3.2 Data Wrangling

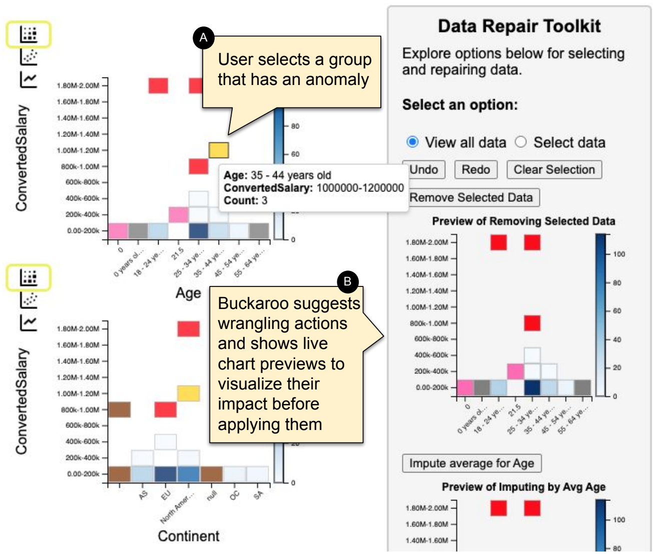
Once errors are detected, Buckaroo enables users to explore and resolve them through interactive, direct manipulation of visualizations. Users can select anomalous groups or individual data points in the chart and invoke repair operations from a contextual repair suggestion toolkit. These actions are reversible and can be iteratively applied to explore their effect on the overall dataset.

Buckaroo provides built-in wranglers for common error types and allows users to define custom wranglers via the Buckaroo API, by mapping a user-defined function to a specific error code.

Repair Suggestions. Upon selecting a group or data point with an anomaly, Buckaroo presents a menu of repair options tailored to the error type. For instance, a missing value might prompt the user to choose between imputation (e.g., using the group mean) or row deletion, while a type mismatch might offer a conversion routine. For example, in Figure 1 (A) a user selects a data anomaly, which prompts Buckaroo to suggest appropriate wrangling actions. For each suggestion, a preview (Figure 1 (B)) of the intended repair is generated. Since datasets may contain a large number of errors, Buckaroo prioritizes user attention by ranking data groups based on the number of anomalies they contain, surfacing the most erroneous groups first. Similarly, wrangling suggestions are ranked by their effectiveness—favoring repairs that resolve the anomaly with minimal side effects on other groups, i.e., minimal errors are caused for the other data groups.

Figure 3 shows an enlarged cropped view of the wrangling UI of Buckaroo. When a user selects a problematic group (Figure 3 (A)), Buckaroo surfaces targeted wrangling options—such as imputation, deletion, or type conversion. Each candidate repair is accompanied by a live chart preview (Figure 3 (B)), allowing users to assess the expected impact on the dataset before applying a change. This preview-driven interaction provides transparent, real-time feedback that makes the consequences of each action visible across related groups, helping users reason about complex group interdependencies without writing any code. By integrating visual anomaly highlighting, repair suggestion, and visual repair preview into a single loop, Buckaroo enables data cleaning through direct manipulation of visualizations, where users can iteratively explore, refine, and undo operations with clarity and control.

Interactive feedback. After a repair is applied, Buckaroo immediately updates the visualization to reflect the modified data. Following previous architectures [33], Buckaroo maintains a backend cache. When a data group is modified, only the affected rows in the backend cache are updated. To balance performance and persistence, Buckaroo periodically flushes these changes to the Postgres database—by default, after every three updates, which can be configured by the user. This feedback loop allows users



Explore options below for selecting and repairing data.

Age: 35 - 44 years old
ConvertedSalary: 1000000-1200000
Count: 3

Figure 3: An overview of repairing an error through Buckaroo's user interface. Each error type has a distinct color (e.g., red groups correspond to average anomalies), Upon selecting the group (A), Buckaroo shows a list of wrangling/repair actions on the right (B) and shows a visual preview of the chart after the repair.

to observe the downstream consequences of a change, including the emergence or resolution of other errors across related groups. By visualizing repair effects, users can make informed decisions without needing to script or rerun batch detection or repair jobs.

3.3 Localized Error Detection and Cross-Chart Dependencies

A key technical challenge in Buckaroo is ensuring that error detection remains efficient during interactive wrangling. Running anomaly detectors across the entire dataset after every repair would be prohibitively expensive and break the real-time user experience [10]. To address this, Buckaroo adopts a localized and incremental error detection strategy grounded in group-based computation.

Groups defined by categorical attributes are the atomic units of visualization and error tracking. Each group is associated with a set of row identifiers (IDs). Anomaly detection is scoped to those IDs. When a repair is applied, we re-run detection only for the affected groups, avoiding unnecessary recomputation.

However, a single row can belong to multiple groups in different charts depending on the grouping attributes. A row with a missing `Income` might appear in a group under `Country=Bhutan` in one chart and under `Degree=BS` in another. A wrangling action on that row could therefore impact multiple visualizations.

To efficiently handle such cross-chart dependencies, Buckaroo maintains a *group overlap graph*, where each node corresponds to a group and an undirected edge connects any two groups that share one or more rows. When a group is updated, Buckaroo consults this graph to determine which groups are affected and selectively re-runs error detection only on those connected components.

This strategy strikes a balance between responsiveness and correctness. It avoids full dataset reprocessing while preserving the accuracy of visual feedback. In practice, most wrangling actions affect only a small number of rows, making this approach highly scalable for interactive use.

4 NAVIGATING DATA ERRORS THROUGH INTERACTIVE CHARTS

A central design goal of BUCKAROO is to enable users to identify and fix data errors entirely through interactions with visualizations. However, this raises an important challenge: how can we efficiently visualize large datasets—especially when most of the data is clean—and still surface rare but critical errors? To address this, BUCKAROO supports two navigation strategies: single-layer navigation, which presents an aggregated or sampled view without panning or drill-down, which is ideal for smaller datasets; and multi-layer drill-down, which enables scalable, details-on-demand exploration through interactive panning and zooming for larger datasets.

4.1 Single-Layer Navigation

In single-layer navigation, the goal is to expose errors within a global view of the dataset, without overwhelming the chart with excessive data points. This is nontrivial because most datasets contain a small fraction of anomalous entries, making it difficult to visually distinguish them from the bulk of the data.

BUCKAROO implements two sampling-based strategies to make errors salient in single-layer views:

- **Error-First Sampling:** For each group, BUCKAROO includes all anomalous records in the chart, ensuring no error is left unvisualized. To provide context, it randomly samples a small number of non-anomalous records from the same group or surrounding groups. This preserves visual contrast while maintaining a manageable rendering cost.
- **Distance-Based Sampling:** In cases where context is important (e.g., for identifying borderline outliers), BUCKAROO also supports sampling based on similarity to error points. For instance, it may select points close to the error cluster in feature space to help users understand how the anomaly deviates from the norm.

These strategies allow users to rapidly scan the dataset for anomalies, explore diverse error types, and compare them against typical values—all within a single visual layer.

4.2 Multi-Layer Navigation

While single-layer navigation offers broad coverage, it is insufficient for large-scale or high-dimensional datasets. To support scalable exploration, BUCKAROO integrates a multi-layer navigation engine that enables users pan, and drill down into data regions of interest—loading only the relevant subsets into view.

This functionality is implemented through a close collaboration with Hopara[1], whose high-performance pan-and-zoom engine has been embedded into BUCKAROO. The result is a an interaction model where users can:

- **Drill Down:** Click on a region or cluster in the chart to reveal a more detailed view of the underlying data, including subgroup breakdowns and localized anomaly summaries.
- **Pan and Zoom:** Move across the chart space without reloading the entire dataset. This ensures that only the visible portion of the data is loaded and rendered at any given time.

The Hopara engine automatically runs SQL queries to fetch each region. Multi-layer navigation achieves two key goals: (1) it ensures that only a manageable volume of data is loaded into memory and visualized at once, improving scalability; and (2) it allows users to focus their attention on data regions of interest.

Together, these navigation strategies make BUCKAROO capable of handling large, messy datasets in a responsive manner while ensuring that anomalies—no matter how rare—remain visually accessible and actionable.

5 RELATED WORK

Efforts to improve data quality and make data preparation more accessible have evolved along two trajectories: (1) Data cleaning research [10] has produced numerous techniques for detecting and repairing errors; however, little attention was given to user-friendly interfaces for data cleaning. (2) Visualization and visual analytics systems provide rich, interactive substrates for examining data, yet they are predominantly employed in a post-hoc, observational capacity rather than as active components of the data cleaning process. BUCKAROO operates at the intersection of these two lines of work, integrating the algorithmic foundations of data cleaning with direct visual manipulation to enable users to visually inspect, adjust, and steer repairs as part of an exploratory workflow.

5.1 Error detection and correction

Subgroup discovery is a well-established task in data engineering [2, 8], focused on identifying statistically distinct and interpretable subsets within a dataset.

A large body of work in the data management community addresses error detection [1] and correction [10, 20]. Error detection methods fall broadly into two categories. Error-type-agnostic systems, such as HoloDetect [7], treat detection as a few-shot learning problem, leveraging rich representations and data augmentation to classify cells as clean or erroneous. In contrast, specialized detectors target specific classes of errors, including anomaly detection [9], functional-dependency violations [25], outlier detection and summarization [42], and duplicate detection [44].

Data repair techniques aim to correct erroneous or inconsistent data. Classical approaches rely on declarative rules—most notably functional dependencies (FDs) and denial constraints—to identify and repair inconsistencies [15, 23, 25]. These rule-driven systems typically formulate repairing as an optimization problem, searching for minimal data updates that satisfy a set of constraints. For example, [15] proposes an approximation to repair FD violations, while Horizon [25] scales FD-based repairing by leveraging the interactions between FDs.

Beyond rule-based techniques, several systems adopt statistical or probabilistic strategies that do not aim to enforce hard integrity constraints. HoloClean [23], for instance, frames repairing as probabilistic inference over a factor graph that integrates signals from

[1]https://hopara.io

constraints, co-occurrence statistics, and external data. Baran [17] learns to repair data errors by unifying multiple correction models and combining their predictions through a binary ensemble classifier. RetClean and Lakefill [19, 43] leverage Large Language Models and data lakes to perform missing data imputation. OTClean [21] is a data cleaning framework that uses optimal transport to repair datasets violating conditional independence constraints while minimally altering their underlying distributions. These systems focus on automated repair generation engines but do not address user interface usability.

Buckaroo complements this body of work by introducing a visual, interactive substrate for exploring and applying repairs. Rather than replacing existing detection or correction algorithms, Buckaroo remains agnostic to how errors are identified or candidate fixes are produced. It instead focuses on enabling direct manipulation of data repairs through visualizations that expose anomalies, and preview the effects of alternative fixes. Buckaroo provides a unifying interactive layer that bridges the gap between back-end repair logic and the user-facing component of data preparation.

5.2 Visualization

A range of systems have investigated how to visually support the data wrangling process [22]. For example, Xiong et al. [40, 41] explore techniques for visualizing wrangling scripts. Buckaroo takes the inverse approach by embedding wrangling actions directly within visualizations. Kasica et al. [14] developed a framework describing the wrangling operations available to multi-table data, specifically in data journalism contexts, which was subsequently used by Xiong et al. [41]. Similarly, Wrangler [11]—later commercialized as Trifacta—synthesizes transformation scripts based on user interactions, a concept recently extended with natural language prompting in Dango [4]. Profiler [12] subsequently developed statistical displays of collections of errors across a dataset, a design which we draw on in Buckaroo. Shrestha et al. [29] develop a method for visualizing and manipulating data frame wrangling operations governed by fluent interfaces like Pandas. Zhu et al. [46] explore how visualization can be used to support understanding automatically generated data wrangling scripts through a bespoke Gantt chart. Scorpion [39] draws on user-identified outliers to synthesize predicates that explain outliers. Unlike these systems, which rely on programming-by-demonstration, Buckaroo emphasizes real-time, direct manipulation of visual elements to guide repairs.

These systems in essence are a specialized form of visualization recommendation system (for which Zeng et al. [45] provide an overview), in which the gestural input to that recommendation system that is used to drive the recommendation is the wrangling process itself. Our work draws on this tradition with a more specific focus. Other systems in this area bear a connection with ours. For instance, Data formulator [35, 36] combines AI-based natural language guidance with a closely related visualization-by-demonstration [34] technique, which observes that preparation and mapping are interwoven components of the presentation process. To that end, Lux [16] weaves visualization recommendation into notebooks, in a manner which centers a predefined collection of analysis tasks. We invert this design, by placing wrangling inside of a visualization environment, rather than placing visualizations in a data programming context.

Our work also draws on those that seek to typify errors visually. For instance, McNutt et al. [18] enumerates a wide range of errors that occur throughout such pipelines, typifying them into various error categories. Ruddle et al. [26] focus on a subset of this process through a taxonomy of data profiling tasks and a mapping to charts that support those tasks. We draw on these mappings in our visualizations that surface varied errors to the end user.

Closely related to our own work is Arachnid [28] which explores using visualizations as medium for modifying data, particularly through modifying visual representations of data. In contrast, we use visualizations as a platform for already identified errors. More distantly Saket et al. [27] explores the use of direct manipulation as a means to specify graphical encodings. In contrast, we draw on this interaction channel as a means to provide guidance to the data cleaning system.

Lastly, our work draws on the long history of visual analytics systems connected with data management systems. These include a wide array of different tools and systems [3, 38]. Of particular relevance are GUI-based visual analytics systems, such as Polaris [32] (subsequently commercialized as Tableau). Our work draws on the patterns and traditions of these systems, but focuses on the particular task of data wrangling.

6 PRELIMINARY RESULTS AND NEXT STEPS

Buckaroo is a work in progress and is actively being developed. The current prototype is available at https://github.com/shape-vis/BuckarooVisualWrangler. Below, we present a preliminary evaluation of the system.

6.1 Expert review

As a basic exploration of the applicability of our design, we consulted two CTOs—one at a data integration company and the other at a data visualization company—in an expert review [6], in which experts were shown a prototype version of Buckaroo. Both agreed that the system would likely be useful for data wrangling, particularly highlighting that many users are impeded by high-barriers on wrangling for large datasets, and that Buckaroo had the potential to significantly lower those barriers.

However, a common concern was related to the usability of the system in the presence of large-scale datasets. They also stressed that the usability of the system will depend on how well we can summarize erroneous data on the charts as having charts with too many errors can be overwhelming.

Additional validation of the usability of this design, particularly for the novice user we target, is necessary future work. However, this initial review is heartening to the validity of this design: centering wrangling in a visualization-based medium seems promising.

6.2 Runtime results

To explore the worries expressed by our experts, we ran a set of preliminary experiments on the Buckaroo runtime. Each experiment simulates a workload of 50 front-end wrangling operations, measuring backend processing time and frontend re-plotting latency.

Table 1: Runtime comparison of wrangling operations in Postgres vs. Pandas. Across all wrangling operations, Postgres significantly outperforms Pandas.

Dataset	Postgres (removal)	Postgres (impute)	Pandas (removal)	Pandas (impute)
StackOverflow	0.18 sec	0.16 sec	1.69 sec	1.27 sec
Adult Income	0.15 sec	0.13 sec	1.40 sec	1.17 sec
Chicago Crime	0.71 sec	0.68 sec	5.87 sec	5.29 sec

These experiments were run on a MacBook Pro with an Apple M4 CPU and 16 GB of RAM.

We use three datasets: StackOverFlow [2] which has 38,091 rows and 21 columns, The Chicago Crime dataset [3] containing 249,542 rows and 17 columns, and the Adult Income dataset [4] which has 48,843 rows and 15 columns. We compare Buckaroo's performance using direct SQL queries over PostgreSQL versus relying on Pandas DataFrames for backend computation. We can clearly see from Table 1 that the average response time is much lower when using Postgres, and that Buckaroo achieves a response time of less than a second for the data removal (remove a data point) and data imputation (replace value by average of column) wrangling operations.

Hopara evaluation. While full integration of Buckaroo with Hopara is still in progress, we successfully implemented wrangling actions within a Hopara drill-down application backed by a Postgres instance on Amazon Web Services. In particular, we measured the latency of row removal triggered from an interactive Hopara bar chart. Across 20 interactions, the average response time was **173 ms** and **201 ms** for the Adult Income dataset, and the StackOverFlow dataset, respectively.

6.3 Next Steps and Concluding Remarks

We are finalizing the implementation of Buckaroo and its integration with Hopara. As part of this effort, we are developing an efficient storage layer based on differential snapshots, avoiding the overhead of storing full copies after each repair.

In conclusion, this work develops the idea of a direct manipulation-based data wrangling tool that is mediated through the graphical medium of visualization. We demonstrate, through our prototype Buckaroo, how a number of key usability features in such a system, such as undo-redo, can be integrated into such a design in an inherently scalable manner. A key facet of this scalability is our notions of extensibility, which allow for the construction of domain-specific error detectors and wranglers. Through this work, we seek to make data wrangling more approachable by shifting to a straightforward-to-understand graphical medium.

REFERENCES

[1] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting Data Errors: Where are we and what needs to be done? *PVLDB* 9, 12 (2016), 993–1004. https://doi.org/10.14778/2994509.2994518

[2]https://survey.stackoverflow.co

[3]https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2/about_data

[4]https://www.kaggle.com/datasets/wenruliu/adult-income-dataset

[2] Jakob Bach. 2025. Using Constraints to Discover Sparse and Alternative Subgroup Descriptions. arXiv:2406.01411 [cs.LG] https://arxiv.org/abs/2406.01411

[3] Leilani Battle and Carlos Scheidegger. 2020. A structured review of data management technology for interactive visualization and analysis. *IEEE transactions on visualization and computer graphics* 27, 2 (2020), 1128–1138.

[4] Wei-Hao Chen, Weixi Tong, Amanda Case, and Tianyi Zhang. 2025. Dango: A Mixed-Initiative Data Wrangling System using Large Language Model. (2025).

[5] Gartner. 2025. *Data Quality: Best Practices for Accurate Insights.* https://www.gartner.com/en/data-analytics/topics/data-quality Accessed: 2025-04-11.

[6] Aurora Harley. 2018. UX Expert Reviews. *Nielsen Norman Group* (25 February 2018). https://www.nngroup.com/articles/ux-expert-reviews/

[7] Alireza Heidari, Joshua McGrath, Ihab F. Ilyas, and Theodoros Rekatsinas. [n. d.]. HoloDetect: Few-Shot Learning for Error Detection. In *SIGMOD '19.*

[8] Francisco Herrera, Cristóbal José Carmona, Pedro González, and María José del Jesus. 2011. An overview on subgroup discovery: foundations and applications. *Knowl. Inf. Syst.* 29, 3 (2011), 495–525. https://doi.org/10.1007/s10115-010-0356-2

[9] Dennis M. Hofmann, Peter M. VanNostrand, Huayi Zhang, Yizhou Yan, Lei Cao, Samuel Madden, and Elke A. Rundensteiner. 2022. A Demonstration of AutoOD: A Self-tuning Anomaly Detection System. *Proc. VLDB Endow.* 15, 12 (2022), 3706–3709. https://doi.org/10.14778/3554821.3554880

[10] Ihab F. Ilyas and Xu Chu. 2019. *Data Cleaning.* ACM.

[11] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *SIGCHI Conference on Human Factors in Computing Systems.* 3363–3372.

[12] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. 2012. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Proceedings of the International Working Conference on Advanced Visual Interfaces.* 547–554.

[13] Bojan Karlaš, Babak Salimi, and Sebastian Schelter. 2025. Navigating Data Errors in Machine Learning Pipelines: Identify, Debug, and Learn. In *Companion of the 2025 International Conference on Management of Data* (Berlin, Germany) *(SIGMOD/PODS '25).* Association for Computing Machinery, New York, NY, USA, 813–820. https://doi.org/10.1145/3722212.3725636

[14] Stephen Kasica, Charles Berret, and Tamara Munzner. 2020. Table scraps: An actionable framework for multi-table data wrangling from an artifact study of computational journalism. *IEEE Transactions on visualization and computer graphics* 27, 2 (2020), 957–966.

[15] Solmaz Kolahi and Laks V. S. Lakshmanan. 2009. On approximating optimum repairs for functional dependency violations. In *Proceedings of the 12th International Conference on Database Theory (ICDT '09).* 53–62. https://doi.org/10.1145/1514894.1514901

[16] Doris Jung-Lin Lee, Dixin Tang, Kunal Agarwal, Thyne Boonmark, Caitlyn Chen, Jake Kang, Ujjaini Mukhopadhyay, Jerry Song, Micah Yong, Marti A Hearst, et al. 2021. Lux: always-on visualization recommendations for exploratory dataframe workflows. *arXiv preprint arXiv:2105.00121* (2021).

[17] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. *Proc. VLDB Endow.* 13, 11 (2020), 1948–1961.

[18] Andrew McNutt, Gordon Kindlmann, and Michael Correll. 2020. Surfacing visualization mirages. In *CHI Conference on human factors in computing systems.* 1–16.

[19] Zan Ahmad Naeem, Mohammad Shahmeer Ahmad, Mohamed Y. Eltabakh, Mourad Ouzzani, and Nan Tang. 2024. RetClean: Retrieval-Based Data Cleaning Using Foundation Models and Data Lakes. In *Proceedings of the VLDB Endowment*, Vol. 17. 4421–4424. https://doi.org/10.14778/3685800.3685890

[20] Wei Ni, Xiaoye Miao, Xiangyu Zhao, Yangyang Wu, Shuwei Liang, and Jianwei Yin. 2024. Automatic Data Repair: Are We Ready to Deploy? *Proc. VLDB Endow.* 17, 10 (June 2024), 2617–2630. https://doi.org/10.14778/3675034.3675051

[21] Alireza Pirhadi, Mohammad Hossein Moslemi, Alexander Cloninger, Mostafa Milani, and Babak Salimi. 2024. OTClean: Data Cleaning for Conditional Independence Violations using Optimal Transport. *Proc. ACM Manag. Data* 2, 3, Article 160 (May 2024), 26 pages. https://doi.org/10.1145/3654963

[22] Vijayshankar Raman and Joseph M. Hellerstein. 2001. Potter's Wheel: An Interactive Data Cleaning System. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 381–390.

[23] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. In *VLDB*, Vol. 10. 1190–1201.

[24] El Kindi Rezig, Lei Cao, Giovanni Simonini, Maxime Schoemans, Samuel Madden, Nan Tang, Mourad Ouzzani, and Michael Stonebraker. 2020. Dagger: A Data (not code) Debugger. In *CIDR.*

[25] El Kindi Rezig, Mourad Ouzzani, Walid G. Aref, Ahmed K. Elmagarmid, Ahmed R. Mahmood, and Michael Stonebraker. 2021. Horizon: scalable dependency-driven data cleaning. *Proc. VLDB Endow.* 14, 11 (2021), 2546–2554.

[26] Roy A Ruddle, James Cheshire, and Sara Johansson Fernstad. 2023. Tasks and visualizations used for data profiling: A survey and interview study. *IEEE Transactions on Visualization and Computer Graphics* (2023).

[27] Bahador Saket, Samuel Huron, Charles Perin, and Alex Endert. 2019. Investigating direct manipulation of graphical encodings as a method for user interaction. *IEEE transactions on visualization and computer graphics* 26, 1 (2019), 482–491.

[28] Conder L Shou and Amita Shukla. 2019. Arachnid: Generalized Visual Data Cleaning. In *Proceedings of the 2019 International Conference on Management of Data*. 1850–1852.

[29] Nischal Shrestha, Titus Barik, and Chris Parnin. 2021. Unravel: A fluent code explorer for data wrangling. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 198–207.

[30] Shafaq Siddiqi, Roman Kern, and Matthias Boehm. 2023. SAGA: A Scalable Framework for Optimizing Data Cleaning Pipelines for ML. In *ACM on Management of Data*, Vol. 1. 1–26.

[31] Cláudio T. Silva, Juliana Freire, Emanuele Santos, and Erik W. Anderson. 2010. Provenance-Enabled Data Exploration and Visualization with VisTrails. In *SIBGRAPI Conference on Graphics, Patterns and Images*. 1–9. https://doi.org/10.1109/SIBGRAPI-T.2010.9

[32] Chris Stolte, Diane Tang, and Pat Hanrahan. 2002. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on visualization and computer graphics* 8, 1 (2002), 52–65.

[33] Wenbo Tao, Xiaoyu Liu, Yedi Wang, Leilani Battle, Çağatay Demiralp, Remco Chang, and Michael Stonebraker. 2019. Kyrix: Interactive Pan/Zoom Visualizations at Scale. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library.

[34] Chenglong Wang, Yu Feng, Rastislav Bodik, Isil Dillig, Alvin Cheung, and Amy J Ko. 2021. Falx: Synthesis-powered visualization authoring. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.

[35] Chenglong Wang, Bongshin Lee, Steven M Drucker, Dan Marshall, and Jianfeng Gao. 2025. Data Formulator 2: Iterative Creation of Data Visualizations, with AI Transforming Data Along the Way. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–17.

[36] Chenglong Wang, John Thompson, and Bongshin Lee. 2023. Data formulator: Ai-powered concept-driven visualization authoring. *IEEE Transactions on Visualization and Computer Graphics* 30, 1 (2023), 1128–1138.

[37] Annabelle Warner, Andrew McNutt, Paul Rosen, and El Kindi Rezig. 2025. Buckaroo: A Direct Manipulation Visual Data Wrangler. *Proc. VLDB Endow.* 18, 12 (Aug. 2025), 5423–5426. https://doi.org/10.14778/3750601.3750687

[38] Eugene Wu, Leilani Battle, and Samuel R Madden. 2014. The case for data visualization management systems: vision paper. *Proceedings of the VLDB Endowment* 7, 10 (2014), 903–906.

[39] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining Away Outliers in Aggregate Queries. *Proc. VLDB Endow.* 6, 8 (2013), 553–564. https://doi.org/10.14778/2536354.2536356

[40] Kai Xiong, Siwei Fu, Guoming Ding, Zhongsu Luo, Rong Yu, Wei Chen, Hujun Bao, and Yingcai Wu. 2022. Visualizing the scripts of data wrangling with SOMNUS. *IEEE Transactions on Visualization and Computer Graphics* (2022). https://doi.org/10.1109/TVCG.2022.3144975

[41] Kai Xiong, Zhongsu Luo, Siwei Fu, Yongheng Wang, Mingliang Xu, and Yingcai Wu. 2022. Revealing the semantics of data wrangling scripts with COMANTICS. *IEEE Transactions on Visualization and Computer Graphics* 29, 1 (2022), 117–127. https://doi.org/10.1109/TVCG.2022.3209470

[42] Jingzhe Xu, Yuhao Deng, Chengliang Chai, Zequn Li, Yuping Wang, and Lei Cao. 2025. OIE: An Interpretable System for Outlier Explanation and Summarization. In *Companion of the 2025 International Conference on Management of Data, SIGMOD/PODS 2025, Berlin, Germany, June 22-27, 2025*, Volker Markl, Joseph M. Hellerstein, and Azza Abouzied (Eds.). ACM, 259–262. https://doi.org/10.1145/3722212.3725120

[43] Chenyu Yang, Yuyu Luo, Chuanxuan Cui, Ju Fan, Chengliang Chai, and Nan Tang. 2025. Data Imputation with Limited Data Redundancy Using Data Lakes. *Proc. VLDB Endow.* 18, 10 (June 2025), 3354–3367. https://doi.org/10.14778/3748191.3748200

[44] Alexandros Zeakis, George Papadakis, Dimitrios Skoutas, and Manolis Koubarakis. 2023. Pre-Trained Embeddings for Entity Resolution: An Experimental Analysis. *Proc. VLDB Endow.* 16, 9 (May 2023), 2225–2238. https://doi.org/10.14778/3598581.3598594

[45] Zehua Zeng, Leilani Battle, et al. 2024. A Systematic Review of Visualization Recommendation Systems: Goals, Strategies, Interfaces, and Evaluations. *Foundations and Trends® in Databases* 14, 1 (2024), 1–71.

[46] Jiajun Zhu, Xinyu Cheng, Zhongsu Luo, Yunfan Zhou, Xinhuan Shu, Di Weng, and Yingcai Wu. 2025. ViseGPT: Towards Better Alignment of LLM-generated Data Wrangling Scripts and User Prompts. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*. 1–16.