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 automation. Buckaroo maintains efficient indexing 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 computational experiments 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. 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 [14].

Data wrangling involves a wide range of tasks, including parsing, reshaping, 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. While prior work has proposed automated or script-based approaches to support cleaning operations [6], 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 [16, 18], 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. 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 2, 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: **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,

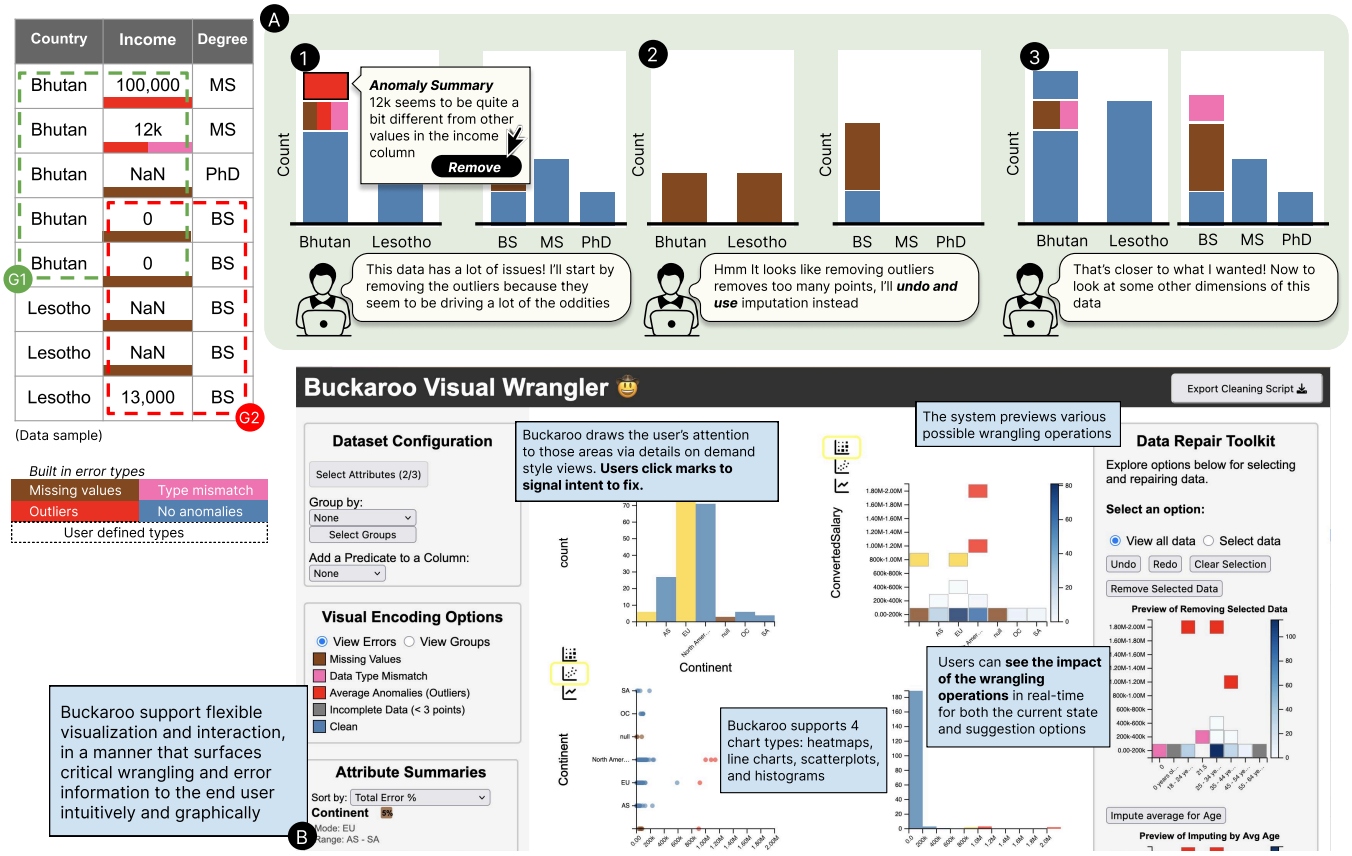

Figure 1: An overview of repairing an error through Buckaroo's user interface. A highlights a user working through iterative and backtrack-laden process of cleaning a dataset. B shows the full interface for a sample of the StackOverflow dataset. Each error type has a distinct color (e.g., red groups 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.

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 [8].

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 will be demonstrated at VLDB 2025 [20]. 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.

By enabling users to *see*, *understand*, and *repair* anomalies through a single, unified visual interface, Buckaroo represents a paradigm shift in how practitioners interact with 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 [6] and subgroup discovery [5]). 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.

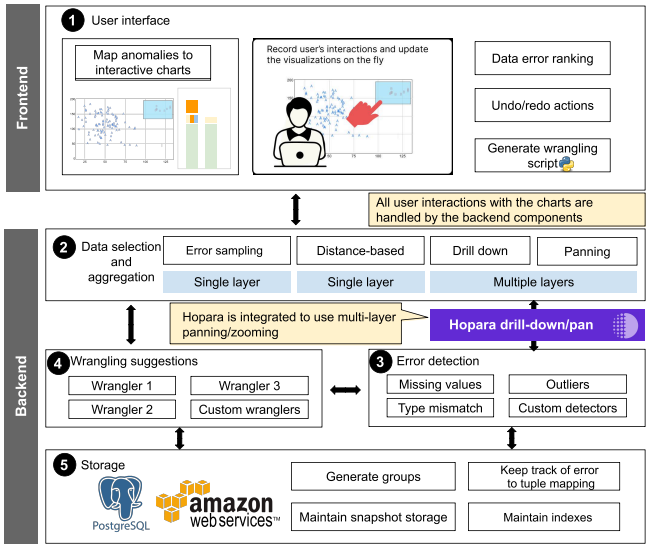

Figure 2: System architecture of Buckaroo. All interactions are mediated by backend components that handle error detection, wrangling suggestions, and state management.

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 Ⓑ) where each histogram bar is subdivided into boxes representing data groups. 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:** Errors are overlaid onto chart elements, enabling real-time visual diagnosis.
- **Immediate feedback:** Each repair updates all the visualizations instantly, allowing users to assess downstream effects.
- **Iterative editing:** Users can undo/redo actions, making wrangling an exploratory, reversible process.
- **Script generation:** Once the wrangling process is complete, the interface offers a downloadable Python script.

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 [6], 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:

```
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 your_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 [8]. 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 user selects a data anomaly, which prompts Buckaroo to suggest appropriate wrangling actions. For each suggestion, a preview (Figure 1 Ⓑ) 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.

Interactive feedback. After a repair is applied, Buckaroo immediately updates the visualization to reflect the modified data. Following previous architectures [19], 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 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 [6]. 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.

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.

Single-Layer Navigation In single-layer navigation, Buckaroo aims to surface anomalies in a global view without overwhelming users, despite errors often being rare and visually drowned out. To address this, Buckaroo uses two sampling strategies: **Error Sampling**, which only shows the groups that exhibit anomalies, and samples from all error types; and **Distance-Based Sampling**, which selects points close to anomalies in the feature space to illustrate how errors deviate from normal patterns.

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 to zoom, 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] (based on Kyrix [19]), whose high-performance pan-and-zoom engine has been embedded into Buckaroo. The result is 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 datasets in a responsive manner while ensuring that anomalies—no matter how rare—remain visually accessible and actionable.

5 RELATED WORK

Anomaly detection and data cleaning. Subgroup discovery is a well-established task in data engineering [1, 5], focused on identifying statistically distinct and interpretable subsets within a dataset.

[1] https://hopara.io

Data repair techniques aim to correct erroneous or inconsistent entries to improve downstream data utility. Classical approaches often rely on rule-based formalisms such as functional dependencies and denial constraints [10, 13, 15], while other systems apply statistical or heuristic methods without enforcing hard integrity constraints [11–13]. While these systems focus on scalable and principled error correction, they lack tight integration with user-facing visual interfaces. Buckaroo complements this body of work by enabling direct manipulation of error-driven repairs through a visual substrate, while remaining agnostic to the underlying detection and repair algorithms. In doing so, Buckaroo offers an interactive layer on top of existing data repair logic, bridging human expertise with automated wrangling.

Wrangling script visualization. A range of systems have investigated how to visually support the data wrangling process. For example, Xiong et al. [21] explore techniques for visualizing wrangling scripts. Buckaroo takes the inverse approach by embedding wrangling actions directly within visualizations. Kasica et al. [9] 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. [21]. Similarly, Wrangler [7]—later commercialized as Trifacta—synthesizes transformation scripts based on user interactions, a concept recently extended with natural language prompting in Dango [2]. Shrestha et al. [17] develop a method for visualizing and manipulating data frame wrangling operations governed by fluent interfaces like pandas. Unlike these systems, which rely on programming-by-demonstration, Buckaroo emphasizes real-time, direct manipulation of visual elements to guide repairs. Additionally, Unlike Mosaic [4], which follows a rule-based, staged pipeline, Buckaroo enables freeform, direct manipulation of data through visualizations. Ruddle et al. [16] describe a taxonomy of data profiling tasks and a mapping to charts that support those tasks. We draw on this mapping in our visualizations that surface varied errors to the end user.

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 [3], 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.

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

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. 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.

[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

REFERENCES

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

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

[3] Aurora Harley. 2018. UX Expert Reviews. *Nielsen Norman Group* (25 February 2018). https://www.nngroup.com/articles/uxâĂŚexpertâĂŚreviews/

[4] Jeffrey Heer and Dominik Moritz. 2024. Mosaic: An Architecture for Scalable & Interoperable Data Views. *IEEE Trans. Visualization & Comp. Graphics (Proc. VIS)* (2024). https://doi.org/10.1109/TVCG.2023.3327189

[5] Franciso 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

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

[7] 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.

[8] 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

[9] 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.

[10] 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

[11] 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.

[12] 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

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

[14] 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*.

[15] 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.

[16] 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).

[17] 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.

[18] 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

[19] 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.

[20] Annabelle Warner, Andrew McNutt, Paul Rosen, and El Kindi Rezig. 2025. Buckaroo: A Direct Manipulation Visual Data Wrangler. In *Proceedings of the VLDB Endowment (Demonstration Track)*. Demo.

[21] 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