PERSIST: PERSISTENT AND REUSABLE INTERACTIONS IN COMPUTATIONAL NOTEBOOKS

A PREPRINT

Kiran Gadhave
University of Utah
kirangadhave2@gmail.com

Zach Cutler
University of Utah
zach.t.cutler@gmail.com

Alexander Lex
University of Utah
alex@sci.utah.edu

ABSTRACT

Computational notebooks, such as Jupyter, support rich data visualization. However, even when visualizations in notebooks are interactive, they still are a dead end: Interactive data manipulations, such as selections, applying labels, filters, categorizations, or fixes to column or cell values, could be efficiently apply in interactive visual components, but interactive components typically cannot manipulate Python data structures. Furthermore, actions performed in interactive plots are volatile, i.e., they are lost as soon as the cell is re-run, prohibiting reusability and reproducibility. To remedy this, we introduce Persist, a family of techniques to capture and apply interaction provenance to enable persistence of interactions. When interactions manipulate data, we make the transformed data available in dataframes that can be accessed in downstream code cells. We implement our approach as a JupyterLab extension that supports tracking interactions in Vega-Altair plots and in a data table view. Persist can re-execute the interaction provenance when a notebook or a cell is re-executed enabling reproducibility and re-use.

We evaluated Persist in a user study targeting data manipulations with 11 participants skilled in Python and Pandas, comparing it to traditional code-based approaches. Participants were consistently faster with Persist, were able to correctly complete more tasks, and expressed a strong preference for Persist.

1 Introduction

Computational notebooks allow for narrative data analysis combining code, data visualizations, text, figures, etc., in the spirit of literate programming [20] proposed by Knuth. Data visualizations in computational notebooks are treated as outputs, similar to text or data tables. As notebooks are code-based, they are (conceptually) reproducible and reusable [19]. The downside of notebook-based approaches is that they require programming skills to use, and that data wrangling operations can be time consuming to get right and may require consulting reference material even for experienced programmers. On a spectrum from usability to complexity, programming is complex, yet it can be applied in generic contexts.

On the other side of the spectrum are specialized interactive visualization tools. Interactive analysis tools can make advanced operations simple, but lack generality: they are good at specific tasks, but lack other desirable characteristics, such as broad applicability, re-usability of analysis processes, reproducibility, etc.

It is unlikely that there are data analysis solutions that are simple yet equally expressive as programming languages; yet certain data operations are much easier to achieve in interactive and visual interfaces than they are in code. We postulate that hybrid, well-integrated solutions can be a significant improvement over the current, mostly isolated state of programming vs interactive visualization. A hybrid solution would allow skilled data analysts to use simple and effective interactive approaches when appropriate, and fall back to expressive code-based operation for tasks that cannot be efficiently completed with interactive tools.

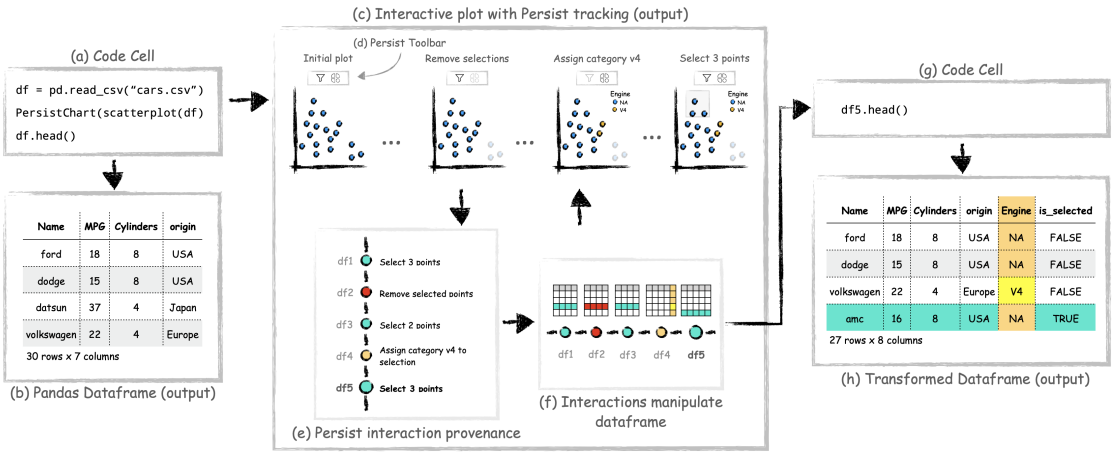

Figure 1: *Persist captures interaction provenance for visualizations embedded in computational notebooks and applies them to data structures. The code cell in (a) shows that an analyst loaded a table, created a Persist-enabled scatterplot, and printed the head of the dataframe (b). The second output of the cell is an interactive scatterplot instrumented with Persist (c). Persist tracks the interaction provenance and supplements operations, such as filters, labeling, or categorization. Using interactions available on the Persist toolbar (d) an analyst has filtered three points, assigned a new category to two points and currently has three points selected. (e) A provenance graph where the analysts interactions are tracked. (f) Persist manipulates the underlying dataframe by translating the interaction provenance to dataframe operations. The updated dataframe,* ***df5****, is then fed back into scatterplot, but is also available in subsequent code cells (g). (h) The manipulated dataframe contains a new categorical column (Engine) and a new Boolean column capturing selections. 3 items were removed (cf. 27 vs 30 rows in (b)).*

Current approaches to integrate visualizations with code are typically one-way streets: plots are generated to inform the analysts and may tell a story, but they cannot be leveraged to manipulate the data. Recent developments in computational notebooks have led to increased support for interactive outputs in notebooks. Libraries like Vega-Altair [27], Plotly, bokeh [3], and IPython widgets [15] can be used to create complex interactive visualizations as outputs, but typically do not support data manipulation. Interactions such as selections and filters cannot be accessed in the code cells. Analysts can use the interactions to explore the data using multiple linked views but must write code to manipulate the data, making it less efficient and more difficult.

Furthermore, there is a mismatch between the persistence of code-cells and actions taken in interactive outputs. Changes to code cells are persistent across notebook saves, and the cells can be re-executed to get the same results. However, the interactions taken in visualizations are transient and are not saved even across cell re-executions. To make the insights made by interactive analysis in such outputs permanent, extensive documentation of the interactions themselves is required, which is a burden to analyst [23]. Hence, interaction is rarely used in practice hampering its adoption.

To remedy this, we introduce a new principle for integrating interaction with code in Persist, our primary contribution. As illustrated in Figure 1, Persist tracks actions taken in interactive visualizations that are specified in code and applies them to a dataset. This dataset can then be used in subsequent analysis. Actions tracked with persist are fully reproducible and reusable: when re-running a notebook, all actions are applied. Even updating a dataset or the visualization itself is possible, as long as the actions are meaningful in the new context.

We implement Persist as a minimal layer on top of existing interactive visualizations. As a proof of concept, we instrument arbitrary Vega-Altair plots: except for a single line invoking Persist, no changes to the plots are necessary. We inject a suite of useful data operations (such as filters, labeling, categorization, changing data types, changing values) that can be triggered either with direct manipulation, a toolbar, or a combination thereof.

We also provide a custom interactive data table that enables a range of manipulations that can be tedious to achieve in code but are natural in an interactive system.

Persist is an open-source JupyterLab extension. The source code is available at `https://github.com/visdesignlab/persist`. Persist can be installed from PyPi `https://pypi.org/project/persist_ext`) using any Python package manager.

We evaluate the efficacy of Persist by comparing it to the traditional code-based approach in a user study with eleven participants skilled in Python and Pandas. The study focused on data cleaning and manipulation operations. The results show that participants were consistently faster, could complete the tasks more often when using Persist, and rated their perceived workload as lower. Par-

ticipants also expressed a preference for Persist over code-based approaches for all tested operations.

2 Persist Walk-Through

To demonstrate how data analysis with Persist works we here describe an analysis session following a user as they work on the Utah avalanche dataset [4]. This analysis session is also available in the supplemental materials. The dataset contains reported instances of avalanches in Utah's mountains. After loading the data, the analyst starts by creating a Persist-enabled Vega-Altair scatterplot of the `Elevation_feet` (the altitude at which the avalanche occurred) vs. `Vertical_inches` (the thickness of the avalanche) columns of the dataset. This visualization helps in identifying the general data distribution and potential outliers. The analyst notices that some records show elevations below 2,000 and above 15,000 ft, which are outside Utah's elevation range (2,300–13,500 ft) and concludes that these must be erroneous entries (see Figure 2a). Using a brush, the analyst selects these points and uses the **Remove Selection** button in the Persist toolbar to remove the selected points (Figure 2b). Persist automatically creates a variable which holds the resulting dataframe.

Next, the analyst uses the updated data to explore monthly avalanche trends by creating a composite Vega-Altair plot containing a bar chart for avalanches aggregated by month and the scatterplot from previous analysis. The bar chart reveals a seasonal pattern, with avalanches peaking in February. The analyst creates a new category called *Avalanche Season* with three options, *Start*, *Middle*, and *End* using the **Add Category** button on the Persist toolbar. They now brush ranges in the bar chart and assign them to one of the avalanche season phases using the **Assign Category** button from the toolbar (Figure 2c). The chart's color-encoding automatically updates to show the new data. In the dataset, Persist automatically created a new column in the dataframe that reflects the interactions done in the bar chart

To examine the data in a tabular format, the analyst employs the interactive Persist Table (see Figure 3), applying it to the current dataframe. This reveals some data artifacts, such as extraneous semicolons in column headers. They correct these in the table by double-clicking and editing the affected column header.

3 Related Work

We review relevant prior work to notebooks and interactive data analysis in general, followed by a discussion of interactive visualization within notebooks.

3.1 Computational Notebooks

Computational notebooks are a popular tool for data exploration and analysis. They fulfill Knuth's [20] vision of literate programming by blending together code, text, figures and data to develop a narrative data analysis. Jupyter [19] is the most popular literate programming tool and has support for wide variety of data analysis and visualization libraries. Jupyter notebooks support iterations during the analysis process, however analysts report ending up with a 'messy' notebook having 'ugly' or 'hacky' code [23]. While a notebook's linear structure is great for narrating the final data analysis process, the data analysis process itself is non-linear, which presents challenges. In order to iterate over different approaches, analysts end up copying code and running cells out of order which reduces the reproducibility of the notebook [23].

3.2 Interactive Data Analysis

Interactive data analysis environments like Tableau [26], Power BI [14], or bespoke custom tools are popular because they support direct manipulation of the data. Certain tasks such as removing outliers or assigning labels are easier to perform directly with an interactive plot. However interactive analysis has limited reproducibility and reusability. It is difficult to replicate an analysis session on updated datasets. These tools also have limited support for transitioning to other tools directly. Usually analysts have to export the results as a data file and then load it in the next tool. Our previous work [10] introduces methods of extracting workflows developed in an interactive visualization system and re-using them, e.g., on new datasets. We also demonstrated exporting workflows so that they can used in a computational notebook. In this work, we are building on this foundation to instrument interactive visualizations directly in notebooks instead.

Kandel et al. [17] introduced an interactive system for data transformations called Wrangler. Wrangler allowed direct manipulation of visualized data and offers an interactive history for review and refinement, but also can export wrangling steps to code. Guo et al. [11] extended the Wrangler system to enable proactive suggestions for data analysis based on inputs from the analyst. MS VS Code has an extension called Data Wrangler [22] which is described as code-centric data cleaning tool. The extension allows for interactive data cleaning and simultaneously generates Python/Pandas code which corresponds to the cleaning operations.

Unlike for these approaches, operations executed in Persist do not directly generate code, but rather contribute to a history of transformation steps that can be re-executed just like code, enabling a close integration of the actions with the visualization, as well as advanced features such as branching states.

3.3 Interactive Analysis in Notebooks

In many scenarios analysts have to frequently switch between interactive and computational tools [28]. However, transitioning between tools in different environments is not straightforward and often requires repetition of analysis steps when moving from code to GUI and vice-versa [5].

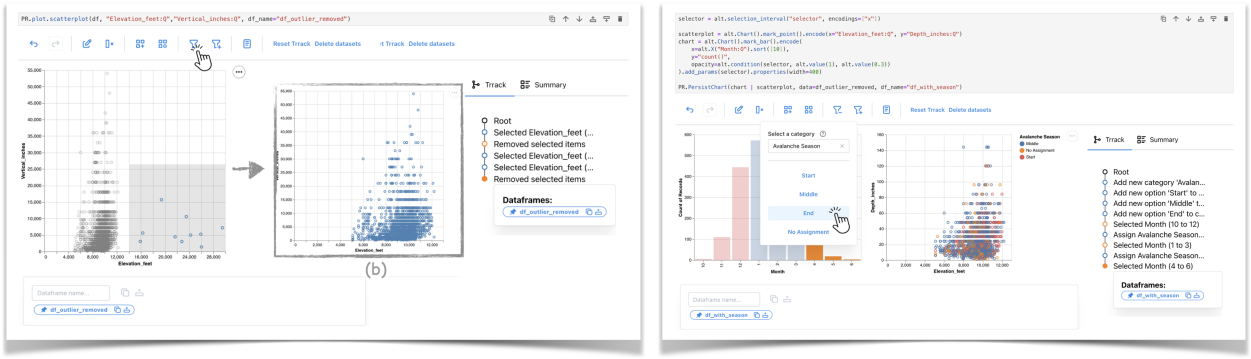

Figure 2: Examples of how Persist is used for data manipulation. (a) An analyst creates an interactive Vega-Altair scatterplot showing elevation and depth of avalanches. They notice the outliers in the elevation and proceed to select the outliers and remove them (b). The interactions are tracked in the provenance graph, and Persist creates a dataframe containing the updated data. In a follow-up cell they use the cleaned data to create a composite Vega-Altair chart with an interactive bar chart showing avalanche records aggregated by month next to the scatterplot of elevation vs. depth. They want to categorize avalanches by start/middle/end of the season using the Persist UI. The colors indicating categories were added without modification to the visualization code, and again all steps are tracked and applied to the dataframe.

Therefore, despite the advantages of an interactive visualization, analysts frequently stick to code environments to avoid the extra work involved with switching between tools [28]. Batch and Elmqvist discuss the need of interactive visual analysis systems that can export the interactive action performed in them [2]. Alspaugh et al. [1] propose developing integrated tools which combine expressiveness of code environments with direct manipulation available in interactive visualizations.

Wrex [8] is a Jupyter notebook extension which implements the programming-by-example principle. Wrex allows analysts to work on samples of a dataframe, and use an interactive grid that provides data transformation examples. Wrex synthesises code from these interactions which the analyst can modify and use subsequently. Wrex does not support persistence of the interactions beyond the generated code.

Support for interactive outputs in notebooks is becoming more common. Jupyter Notebook can be used to create interactive outputs with libraries like JuptyerWidgets [15], Bokeh [3] and Streamlit. However, interactive data analysis is rare to see in notebooks. Schmidt and Ortner [25] cite limited interaction capabilities native to the environment as one of the reasons for the lack of interactive data analysis. Native visualization libraries, such as Matplotlib [13], have only basic interactive capabilities and cannot feed back actions from visualizations to code. Complex visualizations, such as custom tools can be embedded in Jupyter notebooks but typically cannot manipulate data in the notebooks. Libraries such as Vega-Altair [27] support interactive visualizations, but the interactions primarily serve the purpose of coordinating between multiple views and do not natively manipulate data.

Two recent works that align with Persist's vision of seamless transition between code and interactions are Mage and B2. Mage [18] is a Jupyter extension which adds API to support graphical interfaces that interact with the notebook state directly. Mage instruments string templates and patterns to synthesize code from interactions and then inserts the code back into the cell. When the cell is re-executed the generated code has to be mapped back to the interaction, which can be affected by the users changing the generated code. B2 [29] is a Jupyter extension which adds an interactive dashboard to the side of the notebook and "reifies" the interactions as data-queries in the code cell. The data-queries act as a shared-abstraction between the code and the interactive visualizations. Unlike Mage which provides an API and can be extended to any graphical interface, B2 focuses on interactive visualizations it generates in the dashboard area.

Despite the shared vision, Persist approaches the problem of bridging between code and interactions in a different manner. Persist uses interaction provenance as a bridge between code and the output. Having a rich abstraction like interaction provenance allows Persist to handle interactions beyond selections and create Python variables directly. Similar to Mage, Persist can be extended by interface developers. Further, Persist uses the captured interaction provenance to enable persistence of interactions. Persist directly saves the interaction provenance in code cell metadata. It can then load and replay the interaction provenance automatically when the cell is re-executed. A provenance visualization shown directly in the cell output enable seamless transitions between different states of analysis. Persist does not rely on code generation to track the interactions, preventing the cell input area from being cluttered with generated code snippets. The interaction provenance is part of the cell and not just the input code, allowing analysts to update input dataframes or visualizations without losing the interaction history.

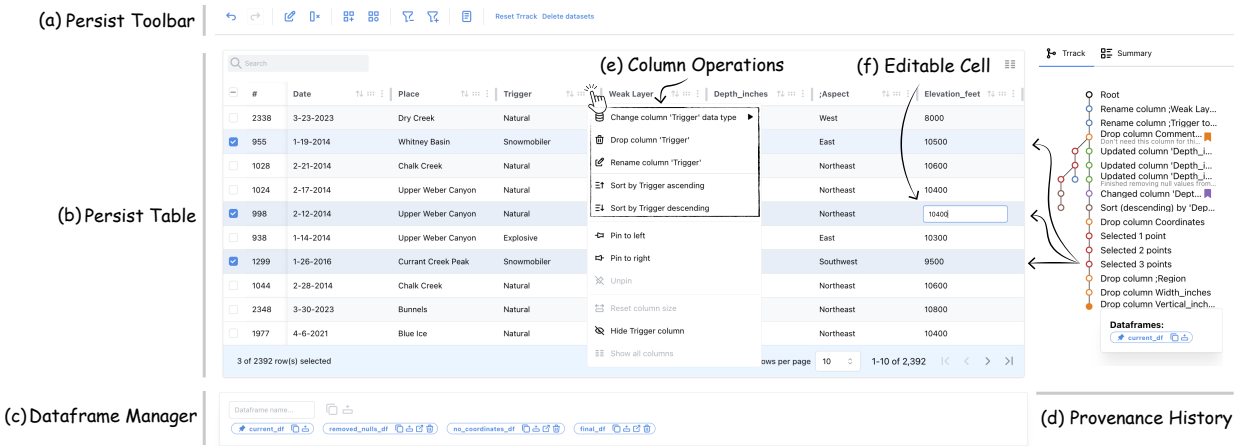

Figure 3: The Persist Table is an interactive, Persist-enabled data table that can be used for manipulating data frames. (a) Shows the Persist toolbar that is also injected into Vega-Altair charts. (b) Shows the paginated table. Analysts can interact with the headers (e), rows, or individual cells (f). (c) The Dataframe Manager serves as the interface between the dataframes created and maintained by Persist and subsequent code. (d) The Provenance History view enables browsing the history, branching, annotating states, creating dataframes for specific states, etc. A summary view (not shown) gives a textual description of the active operations.

4 Persist Principles

Persist can be used with any supported interactive output in Jupyter with minimal changes to code, and to leverage existing libraries. It does so by wrapping interactive components in a layer that (a) tracks interaction provenance, (b) makes data operations available through a GUI, and (c) applies these operations to the data structures, as shown in Figure 1c.

Injecting Operations Persist both listens to native operations of a component, such as selection, and also injects operations into the component, as illustrated in Figure 1d, where a toolbar was added to the native Vega-Altair chart. Persist can also listen to keyboard events or direct manipulation events if they are supported by the component.

Tracking Provenance. Persist captures the interaction events from the interactive component it observes and translates the events into meaningful operations that Persist can track and operate on where necessary. It also injects a provenance visualization widget into the notebook, as illustrated in Figure 1e and shown in Figure 2. The provenance graph documents all the steps taken and can be used to navigate back in history and even to create branches.

Transforming Data. Based on the provenance information, Persist applies the operations to a dataframe in sequence, as illustrated in Figure 1f. Different operations map to different dataset manipulations. For example, selections create a new Boolean column indicating whether an item is selected; other operations might change values, delete rows, re-order columns, etc.

Persist maintains one dynamic dataframe that represents the active state of the user interface. That means that this dataframe is updated every time a new operation is added, but also if the history is used to navigate to a previous state or a different branch. This dataframe can then be used in subsequent code cells, as shown in Figure 1g, with the changes being illustrated in Figure 1h.

Persist also enables users to explicitly create a static dataframe for every provenance state. In this way, one interactive component could be used, to e.g., create two separate dataframes with different subsets of rows which both can then be used in the notebook.

Updating Interactive Components Operations also change how data is best displayed in the interactive components, illustrated by the arrow connecting Figure 1f to the scatterplots. For example, a Vega-Altair chart should update after a filter was applied, or after a category was assigned. We distinguish between two scenarios: (1) The change is to the data, but no additional "channel" has to be encoded. In this case, we can just use the exact same chart specification and re-render with the new data. Example operations that fall into this category are filters. (2) The change in the data has created an additional "channel" that should be encoded. For example, when a category is applied, that category could be shown as a color. Or when a data point is labelled, that label should be displayed in the chart. We do this by updating the Vega-Altair chart specification to encode the category column to the color channel. In case of labelling operations, we encode the added label as tool-tips on the data points.

Re-Execution. When a Persist-enabled cell is re-executed or the notebook is rerun, Persist reapplies the interactions from the beginning to restore the interactive

analysis done by the analyst. Therefore Persist fills the temporal gap by making the interactions persistent as well as supporting revisiting previous interactions.

The interaction provenance saved by Persist is output and data agnostic. Every entry in the provenance can be thought of as a line of Python code. If the Persist-enabled cell is re-run with **updated data**, or even **changes to the visualization code**, Persist will still attempt to apply the interactions to the new output and dataframe. In this manner, analysts can, for example, update their styling or even change their visual encoding choices while retaining their interactive workflow.

However, there are scenarios where the operations may not be compatible with the changed data or changed visualization. If either of them are incompatible, Persist will raise an error similar to Python. If, e.g., if an interaction deletes the column that the updated Vega-Altair chart uses, the chart breaks and may be unusable. However the analyst can use the interaction history to go to the point just before the column is used and branch off into a new analysis session.

5 Persist Design

The previous section described the principles behind Persist. Here we describe our concrete instantiation of these ideas in the Persist prototypes, including a description of the supported operations, the interactive components we provide, and the UI choices we made.

To demonstrate the flexibility of the Persist library, we implement two different visualization options: (1) an interface to arbitrary Vega-Altair charts and (2) an interactive table that can be used to view and manipulate dataframes directly.

As part of the Vega-Altair integration, we inject the Persist toolbar, shown in Figure 3a. *Selection* is natively supported by Vega-Altair charts. The toolbar adds options to *rename columns*, *remove columns*, *label items*, *filter items*, and *categorize items*. It also provides an interface to general Persist operations, such as undo/redo (traversing the provenance graph), resetting all operations, and deleting all dynamic datasets.

The Persist Table, shown in Figure 3 uses the same toolbar and hence supports all the same operations as Vega-Altair charts. The table, however, also enables operations through direct manipulation, by either interacting with the column headers (Figure 3e) or even with individual cells (Figure 3f). These actions include *sorting items*, *renaming columns*, *editing values*, and *reordering columns*. Additional operations, such as *find and replace*, would be easily implementable in the future.

Persist also adds a dataframe manager at the bottom of all Persist-enabled views (Figure 3c). Here, analysts can view existing dataframes (including the dynamic dataframe discussed before) and create and name new dataframes based on the current state of the visualization. The manager also provides buttons to copy the dataframe name and inject a new code cell that prints the dataframe into the notebook. Based on this interface, analysts can easily transition between Persist enabled visualizations and python code to use the created dataframes.

All Persist enabled components are also accompanied with a provenance visualization, rendered as a tree (Figure 3d). Any interaction, in the toolbar or the visualization, creates a new node in the graph. Analysts may revisit any captured step in the graph, updating the visualization. Any existing dataframes associated with the current step are visible, and their name can be copied or inserted into a new cell. Analysts can bookmark steps they deem important, or add annotations. Additionally, a separate tab displays a summary of interactions leading to the current state, instead of the entire provenance graph, if desired.

6 Implementation

Persist is a JupyterLab extension designed to work with JupyterLab 4 and Jupyter Notebook 7 interfaces. Persist uses the Notebook API to access the cell metadata for persistence. Other notebook frontends like VS Code Jupyter and Google Colab do not support the Notebook front end API directly. Persist uses the Trrack [6] provenance tracking library which we developed previously. Persist is available on Python Package Index (PyPi) and can be installed with `pip install persist_ext`.

The Persist extension package contains two modules: The first is the JupyterLab Code Cell extension which augments the CodeCell API with Persist specific features like managers for provenance and generated datasets. The cell extension is developed with TypeScript and React. The second module is PersistOutput widget developed using anywidget [21] which is an abstraction over the popular JupyterWidgets [15]. The PersistOutput widget contains the core Persist module and the interfaces for Vega-Altair charts and the custom data table UI. The widget backend is developed in Python and the front end is developed using TypeScript, React and Mantine React Table.

While Persist attempts to support Vega-Altair charts with minimal overhead, certain Vega-Altair defaults cannot be meaningfully supported. To make the chart compatible with Persist, analyst should follow a few guidelines described in the documentation.

7 Evaluation

We evaluate Persist by comparing it with traditional Pandas-based data analysis in Jupyter in an empirical, lab-based study. Our goal was to find out if using Persist extension made the data analysis faster, more accurate and reproducible, and to collect preferences and opinions from participants with experience in data analysis. Our hypothesis were:

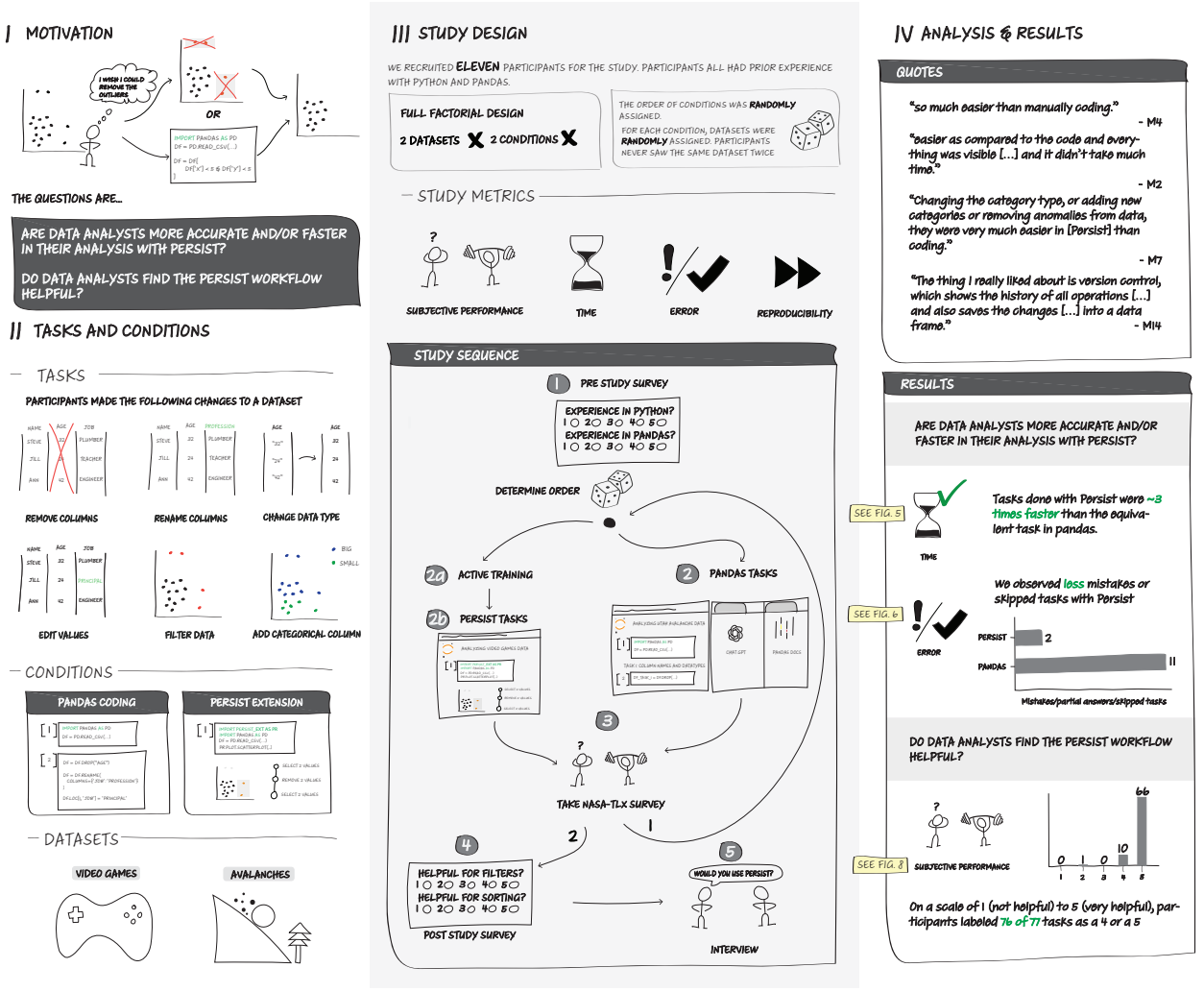

Figure 4: Illustration of study tasks and conditions, design, analysis and results.

- **H1-Speed:** That participants would perform the tasks faster in the Persist condition.
- **H2-Correctness:** Most participants will be able to complete most tasks, but Persist would result in fewer incorrect solutions.
- **H3-Completeness:** That using Persist would result in fewer skipped tasks.
- **H4-Reproducibility:** That using Persist would result in more reproducible notebooks.
- **H5-Workload:** That participants have lower subjective workload using Persist.
- **H6-Helpfulness:** That participants find Persist helpful.

An overview of the study tasks, design, analysis and results is given in Figure 4. We recruited 11 participants who have experience in data analysis using Jupyter notebooks and Pandas (4 identified as women, 7 as men). We recruited from our local student pool; our participants included undergraduate students (2) and graduate students (4 PhD and 5 Masters). The self-reported experience on a 5-point Likert scale for Python was 3.6 ($\sigma = 1.12$), for Pandas was 3 ($\sigma = 1.26$), and for data wrangling was 3.18 ($\sigma = 0.87$). Seven participants had experience using data analysis in research or an industry setting. The study was deemed exempt from full review by the University of Utah IRB (IRB 00167331).

7.1 Procedure

The study employed a within-subject design using two datasets: records about avalanches from the Utah Avalanche Center [4] and Video Games sales data from the Corgis Dataset Project [16].

The tasks involved data cleaning and data manipulation, such as (1a) removing columns not required for analysis, (1b) fixing column names to remove stray characters, (1c) changing the data type of a column, (2a) removing outlier

records, (2b) removing records within a range, (3a) deriving a categorical column based on a numerical column, and a final task (3b) where participants looked at a plot with the new derived column to answer a question. Participants were given a prepared Jupyter notebook for each condition that contained instructions about each task and included boilerplate code, such as imports and data loading. Also, all visualization code (for both conditions) was given, so that participants only had to execute data manipulation steps. See the supplemental material for the notebooks used.

Each participant completed these tasks under both the Persist and traditional Pandas conditions. Tasks were matched between the datasets but varied slightly to fit each dataset. The order and dataset assignment was randomized using a Latin square to counterbalance any effects of datasets and order of conditions.

The study began with an introduction that included disclosures that the screens and room audio were recorded and their notebook would be analyzed, after which we obtained consent. This was followed by a survey where participants reported their experiences with Python, Pandas, and data wrangling. Following this, participants performed the main data analysis tasks under each condition. For the Persist condition, the participants were first given a 15-minute tutorial about Persist followed by a hands-on session in the tutorial notebook. For the Pandas condition, participants were permitted to use any resource to find help, including using their own laptops to use search engines, consult documentation, or employ LLMs like ChatGPT for help. Participants could skip any task if they felt they could not proceed; if a participant skipped, the experimenter loaded a prepared dataset so that they could proceed with subsequent tasks. After each condition, the participants complete the NASA TLX [12] questionnaire to assess their subjective workload. Upon completing the tasks, participants filled out a post-study survey and completed a semi-structured debrief interview to discuss their experiences with Persist. A session lasted approximately 1 hour and 45 minutes. Participants were compensated with a $30 gift card.

7.1.1 Measures

We measured time to completion for each task (using post-hoc video analysis), task correctness (correct, partially correct, skipped, wrong), reproducibility of the notebooks by attempting to re-execute them after the session, and subjective performance (using NASA TLX [12]). We also record preferences and feedback in a survey and a semi-structured interview.

7.1.2 Pilots, Analysis, and Experiment Planning

We conducted four pilots to evaluate tasks, different conditions and datasets, experimental setup, and our procedure. Due to the limitations of null hypothesis significance testing, we base our analysis on best practices for fair statistical communication in HCI [7] by reporting confidence intervals and effect sizes. We compute 95% confidence intervals and effect sizes using Cohen's d to indicate a standardized difference between two means. For the time values, we also supplement our analysis by including p-values from Wilcoxon signed-rank tests (given the non-normal distributions of our data and the within-subjects design). We use a Bonferroni-corrected significance threshold of $p = 0.0071$. We do not compute statistical tests for correctness, as we expected most participants to be able to complete all tasks given the well-defined tasks based on common data manipulation patterns, our participant inclusion criteria (experience in data wrangling), and the ability of participants to use arbitrary reference materials. We also do not perform statistical tests on our perceived workload measures and other survey responses due to the complexity of analyzing subjective scores and our relatively low number of participants.

7.2 Quantitative Results

Each participant attempted 14 tasks in this study, equally distributed between two conditions. Task 3b, which involved interpretation from a pre-generated plot, was identical between the two conditions, so we expected results to be consistent between conditions and exclude it from condition-specific discussions below.

Figure 5a shows the time participants require for the tasks, means, 95% confidence intervals, and statistical information. Participants completed all tasks more quickly using Persist; with means being about 3x lower in the Persist condition, confirming H1-Speed. For all tasks that were different in the conditions, we observe a significant relationship, with a *very large* to *huge* effect size [24]. Moreover, the data shown in Figure 5b indicate that the Persist condition resulted in fewer errors (albeit overall correctness was high). Of the 77 tasks undertaken with Persist, only one was partially wrong (e.g., when instructions asked to filter items large than some value but smaller than another, and participants only filtered based on one condition), and another was incorrect. No tasks were skipped. Conversely, in the Pandas condition, two were incorrect and one was partially correct, while eight tasks were skipped, lending some support to H2–H3.

Upon revisiting the notebooks after the study for re-execution, we found four from the Pandas condition that could not be entirely executed due to errors: manually bypassing some cells was necessary to complete the run. However, most of these cases also coincided with skipped tasks. Additionally, re-executing one notebook revealed an incorrect dataset state, despite the answer being correct during the study. In this case, it was necessary to execute a specific cell twice to get the correct results. In contrast, all 11 notebooks associated with tasks conducted using the Persist extension demonstrated seamless functionality, exhibiting no errors upon re-execution, making the participant sessions more consistently **reproducible**, lending support to H4-Reproducibility.

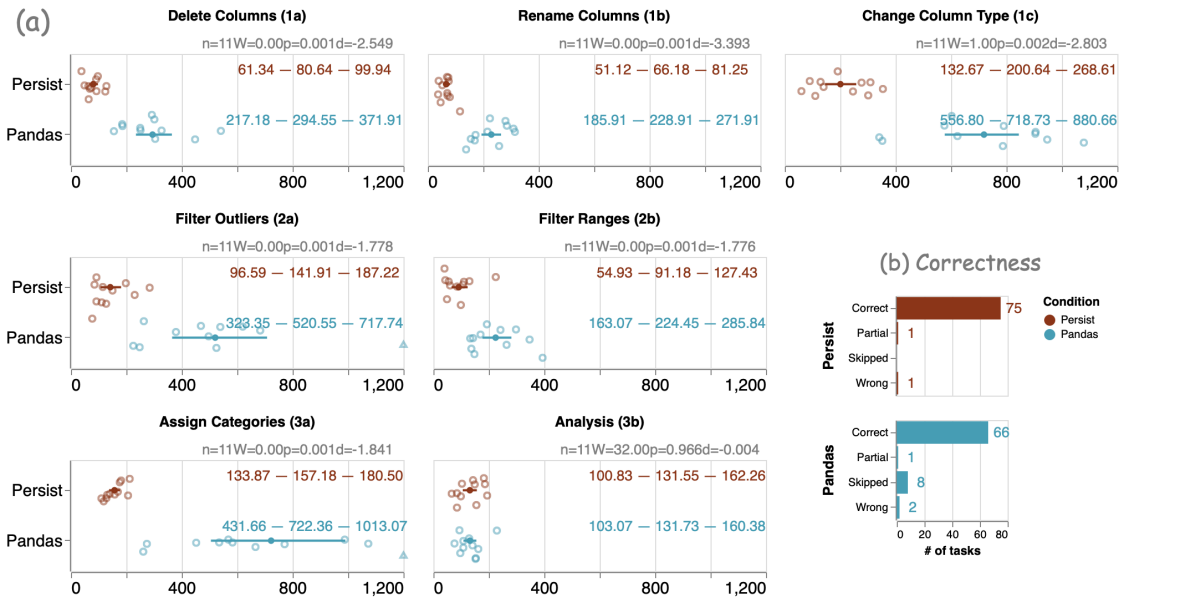

Figure 5: Time and Correctness Results (a) Overview of task completion times for both conditions in seconds. Raw values are shown in jittered dot plots; the solid dot and lines show the mean and the 95% confidence intervals. The colored numbers show the lower and upper bound of the confidence interval and the mean respectively. The plot is clamped at 20 minutes (1200 seconds); data points exceeding 1200 seconds are shown as triangles. Statistical information is provided above the plots in gray. Persist was at least three times faster in all tasks where it was used (note that Task 3b Analysis was a visual analysis task identical in both conditions). All the differences are statistically significant with "very large" to "huge" effect sizes [24]. (b) Overview of task correctness across conditions. In the Persist condition, 75 of 77 tasks were completed correctly, 1 was partially correct, and 1 was wrong. In the Pandas condition, 66 of 77 tasks were completed correctly, 1 was partially correct, 2 were wrong. In 8 cases, participants could not come up with a solution and skipped the task.

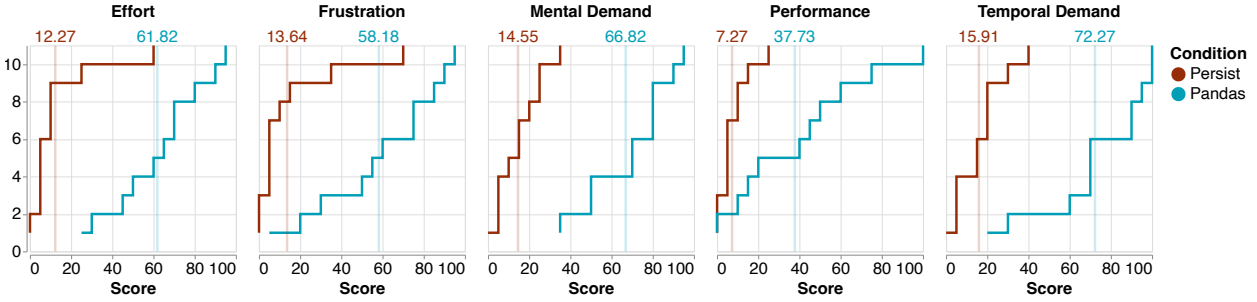

Figure 6: Subjective workload as measured by the NASA TLX shown as an empirical cumulative distribution function (eCDF), where the index of participants is on the y-axis, and the score is on the x axis. Low values are "good" in all cases (low effort, low frustration, etc.). For performance, a low value is on the scale "Good (0)" to "Poor (100)". Averages for the both conditions are given in a lighter line. The Persist condition was rated "better" across all dimensions, mostly with margins of about 50 points on average. The exception is performance, where participants rated their performance with Persist by about 30 points better than with Pandas.

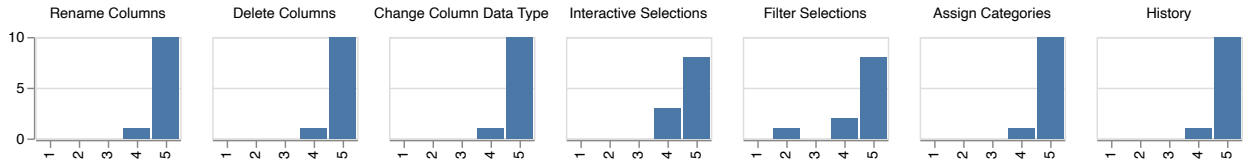

Figure 7: Histograms of ratings for helpfulness of Persist for tasks on a 5-point likert scale, where 1 corresponds to "not helpful", and 5 corresponds to "very helpful". Participants find Persist helpful or very helpful across tasks. For filters one user expressed a preference for entering precise queries and rated Persist lower.

The results of the subjective workload assessment are shown in Figure 6. We omitted the *physical demand* metric from these results as it was not relevant to our study context. Figure 6 presents the empirical CDF plot for both conditions, revealing consistently higher levels of effort, mental demand, temporal demand, frustration, and lower performance associated with the Pandas condition. These findings suggest that participants felt more efficient and less burdened while performing tasks with Persist, confirming H5-Workload. In the post-study survey, participants assessed the helpfulness of Persist in completing various tasks such as renaming columns, deleting columns, changing column data types, interactive selections and filtering, categorizing, and navigating to prior states in the interaction provenance (history). On a 5-point Likert scale, where five denotes 'very helpful' and one signifies 'not helpful,' all participants consistently rated the helpfulness of Persist as either 4 or 5 for every task, with a single exception, as shown in Figure 7, lending support to H6-Helpfulness.

7.3 Qualitative Results

Here we report on the follow-up interviews, summarizing and providing quotes for context. Quotes are edited for grammar; for full transcripts refer to the supplemental material. During the debrief interview, participants were asked about the learning curve of the extension given the short time they had to familiarize themselves with Persist . Participants expressed that Persist was easy to learn and the appropriate icons and tooltips help them discover the feature required for a particular task. P4 recalled, *"I couldn't remember what button it was, but it only took me one second to find it"*.

Participants were asked about their preference between Persist and Pandas. They expressed that they preferred using Persist for most tasks because of the ease of using interactions. P8 described their experience with Persist as, *"what you think you can just do it right away"*. P1 skipped Task 3a in the Pandas notebook but completed it successfully with Persist. They said, *"that was kind of hard for me to write in code. By using the interactive tool, it was super easy."* P6 was one of the most experienced participants who also performed the Pandas tasks the fastest compared to other participants. When discussing the preference between Persist and Pandas they said, *"[with Pandas] I know what I want to do, but I still get stuck, because of the [...] syntax. But with Persist, I don't have to write anything."* While all participants preferred Persist for most of the tasks, some participants had concerns with interactive selections. These concerns stem from the task instructions giving precise numbers, and hence participants were required to accurately select something with a mouse; whereas in code they could put in exact values. P6 said, *"...I'm just just not comfortable with visual selections. Because, as you know, there are edge cases with human errors."* P11 had worked as data scientist in industry. They commented, *"the selections part—I felt it's rough around the edges. [...] if there are many cluttered points [...], I can't nail the selection exactly."* However they later added, *"apart from that point, if there are anomalies or outliers, it's extremely helpful."*

Our goal with Persist is to enable seamless switching between code and interactions, allowing the analyst to use the best tool for the job. Therefore we asked participants about their thoughts on switching between Persist and Pandas. P2 responded, *"maybe there are some features which are not present in this and we might want to use the code. So it is helpful to have both the things"*. We also asked participants if they would like to use Persist in their own data analysis. All participants showed interest in adopting Persist. P6 said, *" I actually really find it helpful and I'm planning to use it on my own research."* P3 responded *"I would definitely use it, because I felt it's really intuitive"*

Participants also brought up the interaction provenance and ability to traverse between the states . P11 said, *"the thing I really liked about is version control, which shows the history of all operations [. . .] and also saves the changes [. . .] into a data frame."* P8 described in detail their struggle with creating multiple temporary variables, copies of notebooks and out of order cells that happen as part of exploratory analysis, *"I do want to say that when you are working on a larger project, you tend to create so many variables [...]. So instead of that, I would definitely want to highlight how you don't have to [create new named variables] for every small change that you make [in Persist], you just have to [create a name] for the one that you wish to retain."*

7.4 Study Discussion

Our results unambiguously demonstrate that participants were, on average, significantly faster using Persist than using standard data frame operations, validating H1-Speed. We also have some evidence to support H2-Correctness, H3-Completeness, and H4-Reproducibility, although the overall low number of errors, skipped tasks, and not-reproducible notebooks indicate that a more powerful study is necessary to make definite statements on these hypothesis. While we haven't conducted statistical tests to evaluate H5-Workload, we think the evidence from the NASA-TLX survey and the interviews unambiguously supports that participants do have lower subjective workload using Persist than using data frame operations. Similarly, our survey data and the qualitative interviews validate H6-Helpfulness.

Looking closer at the data for completion times, it is noticeable that the Pandas condition has a higher variance in completion times, while the Persist condition has minimal variation. This could hint at the fact that Persist is easy to learn and can be consistently applied, while the ease of using Pandas operations depends on the analysts' experience. Almost all participants had to look up syntax for most pandas operations. It is notable though, that even the more proficient participants in our study expressed that they found Persist helpful. We conclude that a tool like

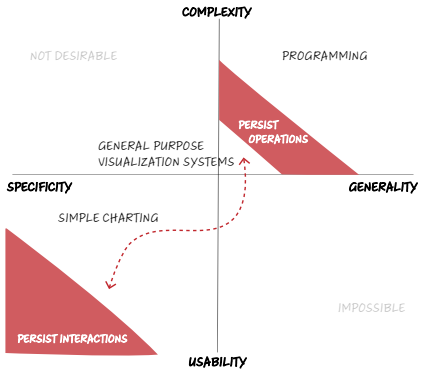

Figure 8: Conceptual trade-offs of data analysis systems.

Persist can significantly speed up the workflow of most somewhat proficient data scientists, while still being an appreciated tool for experts, thereby contributing to making computational data analysis accessible to a wider audience.

Another interesting observation is comparing the completion times of the two filter tasks (2a and 2b). While the difference in the Persist condition is negligible, the difference in the Pandas condition is large: cutting the average from 520 seconds to 224 seconds. We observed that this speed-up is because participants copied the code they had written for task 2a when completing 2b. This observation makes us consider whether we can also in the future copy workflows created with Persist [10].

One critique of our study design could be that we tested tasks and operations that we expected would perform well with Persist, but didn't test tasks that possibly can't be completed with Persist alone, and that hence, the benefits of Persist shown in the study are not surprising. We did test a representative set of operations supported by Persist. We also acknowledge that there are operations that are just easier to execute in code, such as when using regular expressions, or when applying complicated conditional data transformations. Yet the point of Persist is that it allows a seamless transition between interactivity and code, allowing analysts to use the right tool for the job without incurring the costs usually associated with switching between analysis modalities. Hence, we believe that our study demonstrates that Persists is an overall valuable addition to the data science tool-kit.

8 Discussion & Limitations

Most data analysis systems are either useful in a narrow context (specific) and easy to use (such as simple interactive charts, or systems designed for a specific workflow), or general and complex (such as programming languages). This relationship is illustrated in Figure 8. Most visualization systems fall somewhere in the middle between these tools: it takes effort to learn to use a general purpose visualization tool, yet it can be used for many things. The complex–specific quadrant is undesirable, while the easy-to-use–general quadrant is likely impossible to populate. We believe that Persist fills a unique niche by seamlessly bridging between the usability–specificity and the complexity–generality quadrants, thereby allowing some *operations* that would usually be in the domain of programming languages to be executed with easy-to-use interactive systems, while not reducing the overall generality of the data analysis approach.

However, we recognize that Persist has limitations. First, while data operations on pandas and similar tools are scalable to large datasets with millions of rows, Persist is limited w.r.t. scalability by what can be plotted in a reasonable way. While scalable visualization solutions, such as sampling, binning, or indexing exist, they are not implemented in our prototype.

Second, Persist is currently limited to our custom table and Vega-Altair charts. However, since the ecosystem for interactive visualizations in Python is small, we expect it to be feasible to extend our approach to libraries such as Bokeh and Plot.ly.

Similarly, Persist is currently limited to pandas dataframes, and doesn't yet implement all reasonable operation for dataframes. We believe that an abstraction to SQL would open up compatibility with other data structures such as DuckDB and other databases.

9 Conclusion and Future Work

We have introduced Persist, an approach for bringing data operations to interactive visualizations in notebooks and seamlessly bridging the gap between interactive visualizations and code.

While we believe that Persist is useful right now in day-to-day data analysis, there are several immediate extensions we want to implement. Low hanging fruit would be to include other operations, or to improve how persist views are shown in "preview" mode, e.g., when a notebook is rendered in static form on Github. Also, Persist is currently limited to Jupyter, and cannot be used, for example, in Visual Studio code. We expect that a Visual Studio code extension would be easy to implement, yet that extending it to Google Colab would be difficult due to the closed source nature of the platform.

One aspect that Persist doesn't simplify is chart creation. It would be desirable to combine Persist with the chart creation technology shown by others [29, 9]. Also, the Persist principles could be used for changing the visualizations, e.g., by removing or changing titles, visual encodings, etc. In that case, operations would have to be applied to the input visualization instead of to the data frame.

Finally, while we have compared Persist to traditional analysis, it would also be interesting to compare it to alternative code-generating approaches, such as B2, so that we can develop a better understanding of the trade-offs of both approaches.

References

[1] S. Alspaugh, N. Zokaei, A. Liu, C. Jin, and M. A. Hearst. Futzing and Moseying: Interviews with Professional Data Analysts on Exploration Practices. IEEE Transactions on Visualization and Computer Graphics, 25(1):22–31, 2019.

[2] A. Batch and N. Elmqvist. The Interactive Visualization Gap in Initial Exploratory Data Analysis. IEEE Transactions on Visualization and Computer Graphics, 24:278–287, 2018.

[3] Bokeh Development Team. Bokeh: Python library for interactive visualization. https://docs.bokeh.org/en/1.0.1/docs/citation.html, 2018.

[4] U. A. Center. Utah Avalanche Center. https://utahavalanchecenter.org/observations, 2023.

[5] S. Chattopadhyay, I. Prasad, A. Z. Henley, A. Sarma, and T. Barik. What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, CHI '20, pages 1–12, New York, NY, USA, Apr. 2020. Association for Computing Machinery.

[6] Z. T. Cutler, K. Gadhave, and A. Lex. Trrack: A Library for Provenance Tracking in Web-Based Visualizations. In IEEE Visualization Conference (VIS), pages 116–120. IEEE, 2020.

[7] P. Dragicevic. Fair Statistical Communication in HCI. In J. Robertson and M. Kaptein, editors, Modern Statistical Methods for HCI, pages 291–330. Springer International Publishing, Cham, 2016.

[8] I. Drosos, T. Barik, P. J. Guo, R. DeLine, and S. Gulwani. Wrex: A Unified Programming-by-Example Interaction for Synthesizing Readable Code for Data Scientists. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, CHI '20, pages 1–12, New York, NY, USA, Apr. 2020. Association for Computing Machinery.

[9] W. Epperson, V. Gorantla, D. Moritz, and A. Perer. Dead or Alive: Continuous Data Profiling for Interactive Data Science. arXiv preprint arXiv:2308.03964, 2023.

[10] K. Gadhave, Z. Cutler, and A. Lex. Reusing Interactive Analysis Workflows. Computer Graphics Forum, 41(3):133–144, 2022.

[11] P. J. Guo, S. Kandel, J. M. Hellerstein, and J. Heer. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, UIST '11, pages 65–74, New York, NY, USA, Oct. 2011. Association for Computing Machinery.

[12] S. G. Hart and L. E. Staveland. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In P. A. Hancock and N. Meshkati, editors, Advances in Psychology, volume 52 of Human Mental Workload, pages 139–183. North-Holland, Jan. 1988.

[13] J. D. Hunter. Matplotlib: A 2D Graphics Environment. Computing in Science Engineering, 9(3):90–95, May 2007.

[14] M. Inc. Microsoft Power BI. https://powerbi.microsoft.com/de-de/, 2019.

[15] IPython Widget Team. Jupyter Widgets — Jupyter Widgets 8.1.1 documentation. https://ipywidgets.readthedocs.io/en/stable/, 2015.

[16] A. C. B. Kafura, R. Whitcomb, J. Riddle, O. Saleem, D. E. Tilevich, Dr. Clifford, A. Shaffer, and Dr. Dennis. CORGIS Datasets Project. https://corgis-edu.github.io/corgis/, 2023.

[17] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '11, pages 3363–3372, New York, NY, USA, 2011. ACM.

[18] M. B. Kery, D. Ren, F. Hohman, D. Moritz, K. Wongsuphasawat, and K. Patel. Mage: Fluid Moves Between Code and Graphical Work in Computational Notebooks. Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology, pages 140–151, Oct. 2020.

[19] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and Jupyter development team. Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Scmidt, editors, 20th International Conference on Electronic Publishing (01/01/16), pages 87–90. IOS Press, 2016.

[20] D. E. Knuth. Literate Programming. The Computer Journal, 27(2):97–111, Jan. 1984.

[21] Manz, T. Anywidget. 2023.

[22] microsoft. Data Wrangler Extension for Visual Studio Code. Microsoft, Nov. 2023.

[23] A. Rule, A. Tabard, and J. D. Hollan. Exploration and Explanation in Computational Notebooks. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pages 1–12, Montreal QC Canada, Apr. 2018. ACM.

[24] S. Sawilowsky. New Effect Size Rules of Thumb. Journal of Modern Applied Statistical Methods, 8(2), Nov. 2009.

[25] J. Schmidt and T. Ortner. Visualization in Notebook-Style Interfaces. In Proceedings of the Workshop

on the Gap between Visualization Research and Visualization Software (VisGap), May 2020.

[26] Tableau Software. Tableau Software. http://www.tableau.com/, Dec. 2015.

[27] J. VanderPlas, B. E. Granger, J. Heer, D. Moritz, K. Wongsuphasawat, A. Satyanarayan, E. Lees, I. Timofeev, B. Welsh, and S. Sievert. Altair: Interactive Statistical Visualizations for Python. Journal of Open Source Software, 3(32):1057, Dec. 2018.

[28] K. Wongsuphasawat, Y. Liu, and J. Heer. Goals, Process, and Challenges of Exploratory Data Analysis: An Interview Study. ArXiv, 2019.

[29] Y. Wu, J. M. Hellerstein, and A. Satyanarayan. B2: Bridging Code and Interactive Visualization in Computational Notebooks. In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology, UIST '20, pages 152–165, New York, NY, USA, Oct. 2020. Association for Computing Machinery.