DOI: 10.1111/cgf.15092
Eurographics Conference on Visualization (EuroVis) 2024
W. Aigner, D. Archambault, and R. Bujack
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 43 (2024), Number 3

Persist: Persistent and Reusable Interactions in Computational Notebooks

K. Gadhave[1], Z. Cutler[1], A. Lex[1],

[1]University of Utah, USA

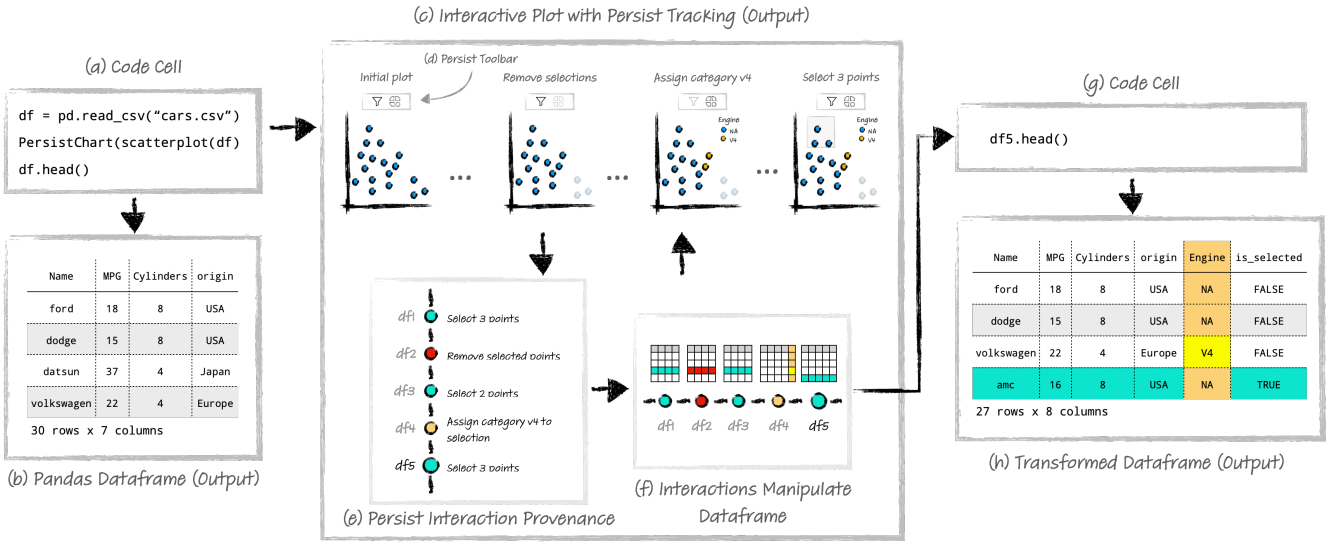

Figure 1: *Persist captures interaction provenance for visualizations embedded in computational notebooks and applies these interactions to data structures. The code cell in (a) shows that an analyst loaded a table, created a Persist-enabled scatterplot, and (b) printed the head of the dataframe. (c)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

Abstract

Computational notebooks, such as Jupyter, support rich data visualization. However, even when visualizations in notebooks are interactive, they are a dead end: Interactive data manipulations, such as selections, applying labels, filters, categorizations, or fixes to column or cell values, could be efficiently applied in interactive visual components, but interactive components typically cannot manipulate Python data structures. Furthermore, actions performed in interactive plots are lost as soon as the cell is re-run, prohibiting reusability and reproducibility. To remedy this problem, we introduce Persist, a family of techniques to (a) capture interaction provenance, enabling the persistence of interactions, and (b) map interactions to data manipulations that can be applied to dataframes. 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 interaction provenance when a notebook or a cell is re-executed, enabling reproducibility and re-use. We evaluate 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 and were able to correctly complete more tasks with Persist.

CCS Concepts

• ***Human-centered computing*** → ***Visualization;***

1. Introduction

Computational notebooks allow for narrative data analysis combining code, data visualizations, text, figures, etc., in the spirit of literate programming [Knu84] 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 [KRP*16]. 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 they lack generality: they are good at specific tasks but lack other desirable characteristics, such as broad applicability, reusability of analysis processes, reproducibility, etc.

It is unlikely that there are data analysis solutions that are simple and as 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 operations for tasks that cannot be efficiently completed with interactive tools.

Recent developments in computational notebooks have led to increased support for interactive outputs in notebooks. However, these new approaches to integrating visualizations with code are typically one-way streets: plots are generated to inform the analysts and tell a story, but they cannot be leveraged to manipulate the data. Libraries such as Vega-Altair [VGH*18], Plotly [Plo15], bokeh [Bok18], and IPython widgets [IPy15] can be used to create interactive visualizations as outputs, but typically do not support data manipulation. The code cells cannot access interactions such as selections and filters. Analysts can use interactions to explore the data but must write code to manipulate the data. 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, interactions taken in visualizations are transient and are not saved even across cell re-executions. To make insights based on interactive analysis permanent, extensive documentation of the interactions themselves is required, which is a burden to analysts [RTH18].

Recently, systems such as B2 [WHS20] and Mage [KRH*20] have introduced solutions to address the barrier between interactions and code by generating code in response to interactions. As our primary contribution, we introduce a new principle for integrating interactions with code based on provenance data. 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. Compared to generating code, using provenance associated with the visualization has several advantages: first, unlike for code, there is no "gap" between the actions taken in a visualization and the injected code that is located in another cell. Second, manual changes to generated code pose challenges to keeping the visualization and the code in sync. Third, excessive generated code can lead to clutter, and finally, provenance enables easy branching and iteration, which is important when exploring data interactively.

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 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 for the extension and the supplemental materials including the study notebooks and analysis can be accessed at `https://github.com/visdesignlab/persist`.

We evaluate the efficacy of Persist by comparing it to the traditional code-based approach in a user study with 11 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. Participants 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 describe an analysis session following an analyst as they work on a Utah avalanche dataset [Cen23]. This analysis session is also available in the supplemental materials. The dataset contains reported instances of avalanches in the Utah 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 the analyst identify 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 tracks all the interactions in a provenance graph (Figure 2c) and automatically creates a variable that 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 the previous analysis (Figure 2d). The bar chart reveals a seasonal pattern, with avalanches peaking in February. The analyst

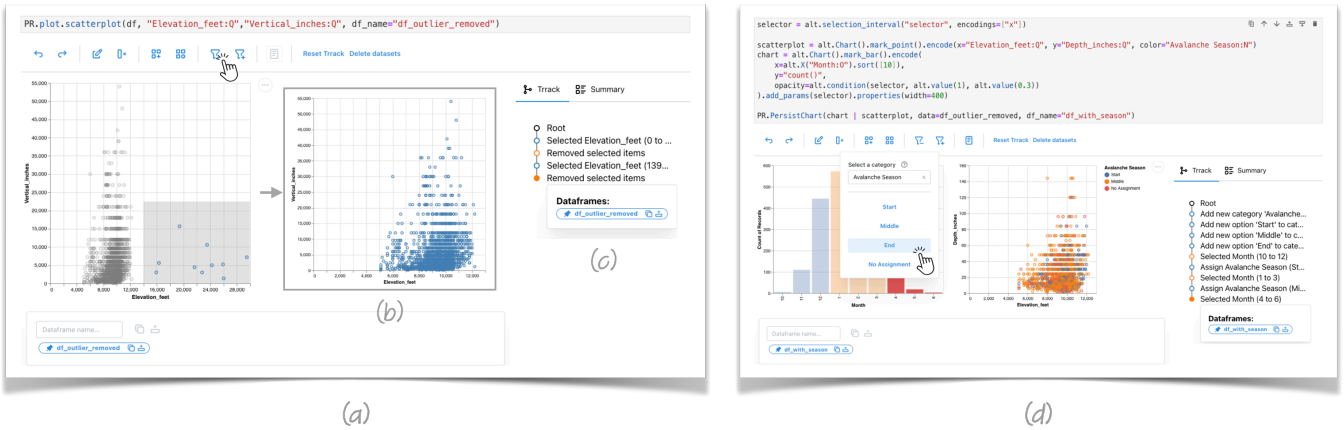

Figure 2: *Examples of how Persist is used for data manipulation. (a) An analyst creates an interactive Vega-Altair scatterplot showing elevation and depth of avalanches. They notice the outliers in the elevation and proceed to select the outliers and remove them (b). (c) The interactions are tracked in the provenance graph and Persist creates a dataframe containing the updated data. (d) In a follow-up cell, the analyst uses 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. Using the Persist UI, the analyst wants to categorize avalanches by season. The colors indicating categories were added without modification to the visualization code, and again all steps are tracked and applied to the dataframe.*

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. 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 applied in the bar chart.

To examine the data in a tabular format, the analyst employs the PersistTable (see Figure 3). The table reveals data artifacts, like stray semicolons in column headers. The analyst corrects these artifacts in the table by directly editing the affected column header.

3. Related Work

We will review prior work related to computational notebooks, interactive data analysis, and interactive visualization in notebooks.

3.1. Computational Notebooks

Computational notebooks are popular tools for data exploration and analysis. They fulfill Knuth's [Knu84] vision of literate programming by blending together code, text, figures, and data to develop a narrative data analysis. Jupyter [KRP*16] is one of the most popular literate programming tools and has support for a wide variety of data analysis and visualization libraries.

The data analysis process is iterative. The linear structure of a notebook serves well for narrating the data analysis process. Jupyter notebooks support iterations during the analysis process; however, analysts report ending up with a 'messy' notebook having 'ugly' or 'hacky' code [RTH18]. To iterate over different approaches, analysts copy code and run cells out of order, reducing the notebook's reproducibility [RTH18, HHB*19].

3.2. Interactive Data Analysis

Interactive data analysis environments such as Tableau [Tab15], Power BI [Inc19], or bespoke custom tools are popular because they support direct manipulation of 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 [GGC*21]. Replicating the interactive analysis on updated datasets is also difficult. 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 [GCL22] introduced methods of extracting workflows developed in an interactive visualization system and reusing them, e.g., on new datasets. We also demonstrated exporting workflows so that they can be used in a computational notebook. In this work, we are building on this foundation to instrument interactive visualizations directly in notebooks. Kandel et al. [KPHH11] introduced an interactive system for data transformations called Wrangler. Wrangler allows direct manipulation of visualized data, offers an interactive history for review and refinement, and can export wrangling steps to code. Guo et al. [GKHH11] 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 [mic23], described as a code-centric data cleaning tool. The extension allows for interactive data cleaning and simultaneously generates Python/Pandas code that corresponds to the cleaning operations.

In contrast to 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 switch between interactive and computational tools [WLH19]. 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 [CPH*20]. Therefore, despite the advantages of interactive visualization, analysts frequently stick to code environments to avoid the extra work involved with switching between tools [WLH19]. Batch and Elmqvist discuss the need for interactive visual analysis systems that can export the interactive action performed in them [BE18]. Alspaugh et al. [AZL*19] propose developing tools that combine the expressiveness of code environments with direct manipulation available in interactive visualizations.

Support for interactive outputs in notebooks is becoming more common. Wrex [DBG*20] is a Jupyter Notebook extension that 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 synthesizes code from these interactions, which the analyst can modify and use subsequently. Wrex does not persist the interactions beyond the generated code. Jupyter Notebooks can be used to create interactive outputs with libraries such as JuptyerWidgets [IPy15], Bokeh [Bok18] and Streamlit [Tea]. However, interactive data analysis is still rarely seen in notebooks. Schmidt and Ortner [SO20] 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 [Hun07], have basic interactive capabilities and cannot feed back actions from visualizations to code. Complex visualizations, such as custom tools embedded in Jupyter notebooks, typically cannot manipulate data in the notebooks. Libraries such as Vega-Altair [VGH*18] support interactive visualizations, but the interactions primarily serve the purpose of coordinating multiple views and do not manipulate data.

Wu et al. [WHS20] introduce the B2 system, which attempts to address the gaps between code and interactions. B2 uses data queries or selections as the shared abstraction between the code and interactive visualizations. The B2 approach expresses the interactions as selections of data or query predicates. Persist also employs the idea of using a shared abstraction to bridge between code and interactions. Persist supports capturing and replaying complex interactions such as categorization by capturing the interaction provenance instead of data queries. Results of interactions in Persist can be accessed directly as Pandas dataframes, which are kept in sync with the interaction provenance.

The Mage API extension for notebooks [KRH*20] also aims to support fluid movement between code and interactive outputs. Mage detects the interactions in the output, maps them to equivalent code using code templates, and injects the code into the cell. Code templates offer more flexibility compared to the B2 approach of using selections in the data. Mage achieves the persistence of interactions by injecting the filled-in code templates in the cell. B2 adopts a similar approach of injecting selection predicates in the code cell to persist the interactions. Both these approaches risk cluttering the code cells with unused code snippets. Keeping track of nonlinear analysis using code snippets can become complex. Persist directly saves the interaction provenance to the notebook and replays it when required. 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.

4. Persist Principles

Persist can be used with any supported interactive output in Jupyter with minimal changes to code. 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. Next, we lay out the implementation-independent design principles of Persist before discussing concrete design and implementation in the next section

Injecting Operations. Persist listens to native operations of a component, such as selection, and 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 the component supports them.

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 2c. The provenance graph documents all the steps taken and can be used to navigate back in history and 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, e.g., create two separate dataframes with different subsets of rows that both can then be used in the notebook.

Updating Interactive Components Operations can change how data are best displayed in the interactive components, illustrated by the arrow connecting Figure 1f to the scatterplots. For example, a chart should update after a filter is applied (removing or graying out the data points) or after a category is assigned (changing its color or shape). We distinguish between two scenarios: (1) The change is to the data, but no additional "channel" has to be encoded, which is the case, for example, when filtering. In this case, we can just use the original chart specification and re-render with the new data. (2) The change in the data has created an additional "channel" that should be encoded. For example, when a category is assigned to a data point, that category could be shown with color. The label

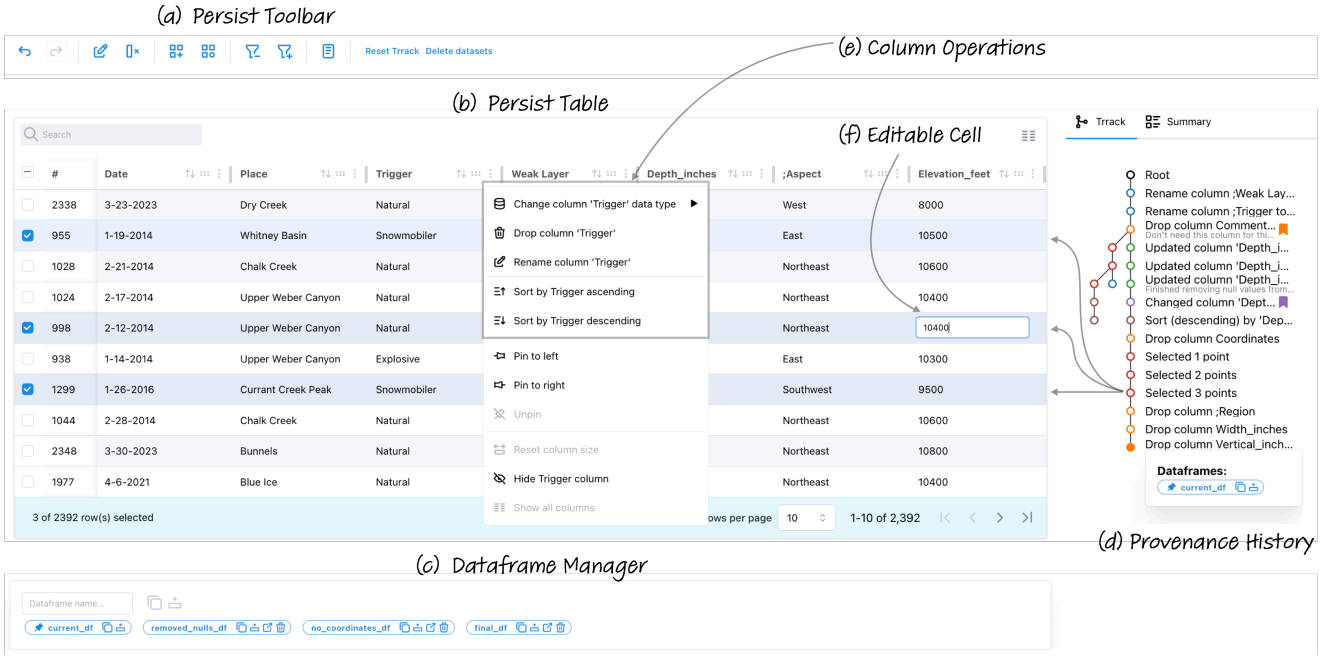

Figure 3: *The PersistTable is an interactive data table that can be used for manipulating dataframes. (a) The Persist toolbar that is also injected into Vega-Altair charts. (b) The paginated table. Analysts can interact with the (e) headers, rows, or (f) individual cells. (c) The Dataframe Manager serves as the interface between the dataframes 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 active operations.*

should be displayed in the chart when a data point is labeled. We label the point by updating the chart specification to encode these additional attributes, either as a visual channel or as a tooltip.

Re-Execution. When a Persist-enabled cell is re-executed, 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. Persist interfaces with the output visualization to track interactions and update it in response. Therefore, the types of interactions and complexity of interactions Persist can handle are limited by the visualization and the interface it provides. However, in some scenarios the operations may not be compatible with the changed data or changed visualization. If either of them is incompatible, Persist will raise an error similar to Python. If, e.g., an interaction deletes the column that the updated Vega-Altair chart uses, the chart breaks and may be unusable. An analyst can use the interaction history to undo such an action.

5. Persist Design

The previous section described the principles behind Persist. Here, we describe our concrete implementation of these ideas in the Persist prototype, 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 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. In addition, the table enables operations through direct manipulation by either interacting with the column headers (Figure 3e) or with individual cells (Figure 3f). These actions include *sorting items*, *renaming columns*, *editing values*, and *reordering columns*. Additional operations could be easily added, such as *find and replace*.

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 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 that uses 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. Since Persist saves the interaction provenance directly in the notebook, the interactive analysis can be shared along with the notebook. Persist uses the Trrack [CGL20] provenance tracking library, which we developed previously. Persist can be installed with `pip install persist_ext`. Persist is developed using Python, JupyterWidgets [IPy15], anywidget [Man23], TypeScript, and React.

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 the Persist extension made the data analysis faster and more accurate and reproducible, and to collect preferences and opinions from participants with experience in data analysis. Our hypotheses were:

- **H1-Speed:** That participants would perform the tasks faster in the Persist condition.
- **H2-Correctness:** That most participants would 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 would have lower subjective workload using Persist.
- **H6-Helpfulness:** That participants would 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: undergraduate (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 (00167331).

7.1. Procedure

The study employed a **within-subject design** using two datasets: records about avalanches from the Utah Avalanche Center [Cen23] and Video Games sales data from the Corgis Dataset Project [KWR*23]. 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) for which 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 the participant's notebook would be analyzed, after which we obtained their consent. This introduction was followed by a survey in which participants reported their experiences with Python, Pandas, and data wrangling. Following the survey, 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 such as 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 completed the NASA TLX [HS88] questionnaire to assess their subjective workload. Upon completing the tasks, participants filled out a post-study survey and completed a semi-structured 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 [HS88]). We also recorded 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

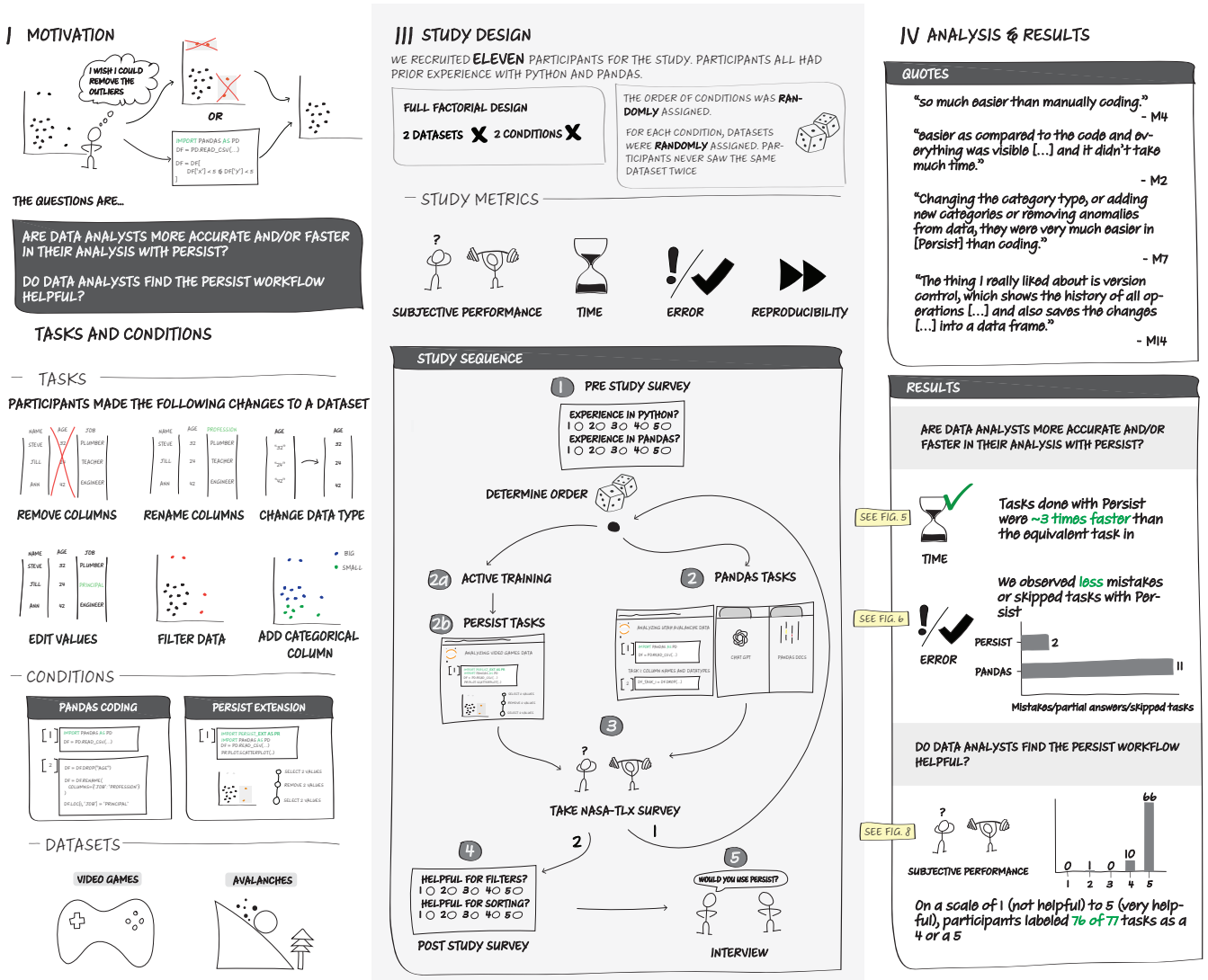

Figure 4: *Illustration of study tasks and conditions, design, analysis, and results.*

on best practices for fair statistical communication in HCI [Dra16] 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. We also supplement our analysis for the time values 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 objectives 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 excluded the task 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 [Saw09]. 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 and another was incorrect. No tasks were

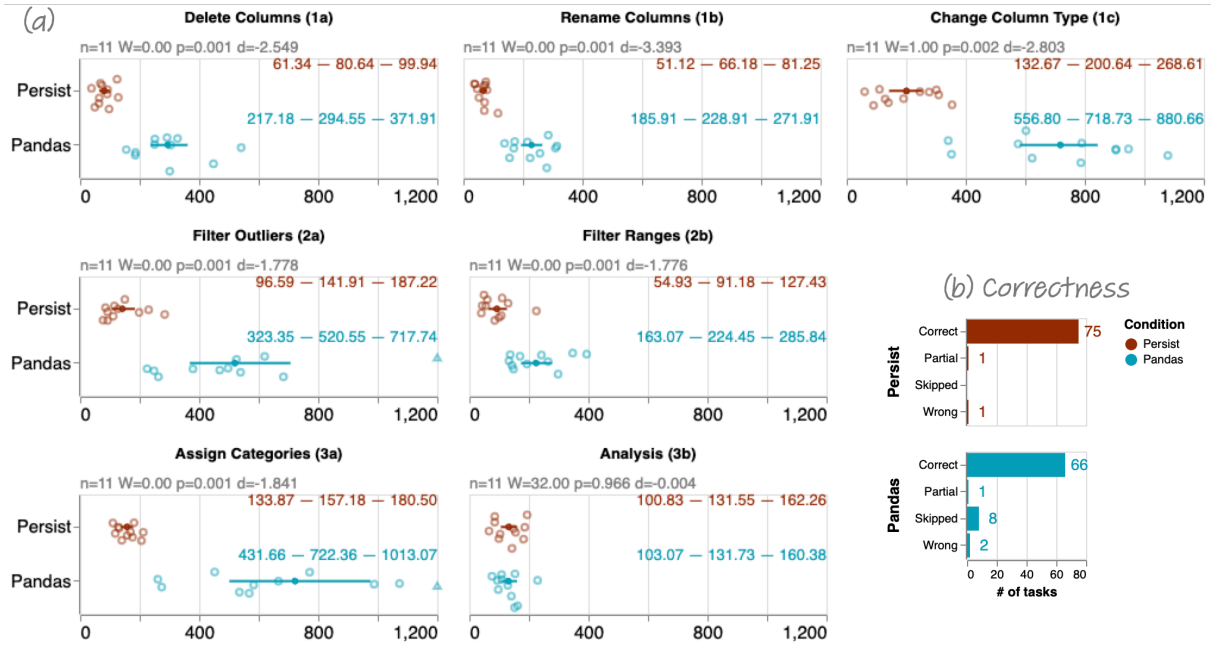

Figure 5: *Time and Correctness Results (a) Overview of task completion times for both conditions in seconds. Raw values are shown in jittered dot plots; the solid dot and lines show the mean and the 95% confidence intervals. The colored numbers show the lower and upper bounds of the confidence interval 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 for which 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 [Saw09]. (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.*

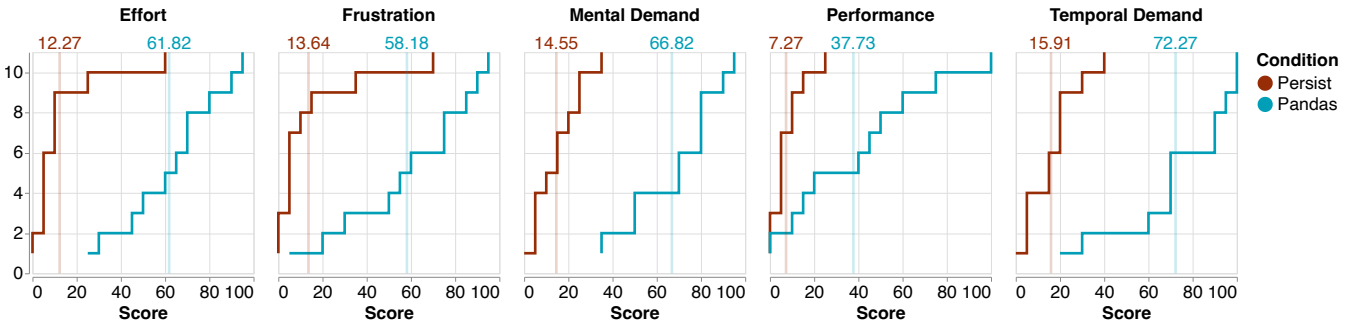

Figure 6: *Subjective workload as measured by the NASA TLX shown as an empirical cumulative distribution function (eCDF), where the index of participants is on the y-axis, and the score is on the x axis. Low values are "good" in all cases (low effort, low frustration, etc.) For performance, a low value is on the scale "Good (0)" to "Poor (100)". Averages for both conditions are shown with 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.*

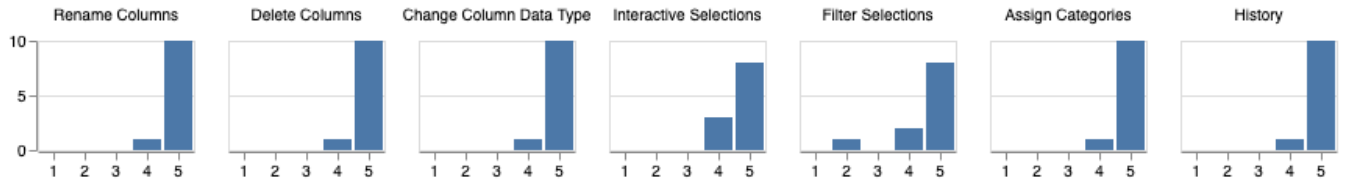

Figure 7: *Histograms of ratings for helpfulness of Persist for tasks on a 5-point Likert scale, where 1 corresponds to "not helpful", and 5 corresponds to "very helpful". Participants find Persist helpful or very helpful across tasks. For filters, one user expressed a preference for entering precise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

skipped. Conversely, in the Pandas condition, two were incorrect, one was partially correct, and eight tasks were skipped, lending some support to H2-Correctness and H3-Completeness.

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** and lending support to H4-Reproducibility.

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 poststudy survey, participants assessed the helpfulness of Persist in completing various tasks such as renaming columns; deleting columns; changing column data types and 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 icons and tooltips helped 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 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. 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."* All participants preferred Persist for most of the tasks, but some had concerns with interactive selections. These concerns stem from the task instructions giving precise numbers, and hence participants were required to accurately select certain points with a mouse; whereas in code they could put in exact values. P6 said, *". . . I'm just not comfortable with visual selections. Because, as you know, there are edge cases with human errors."* P11 (who had worked as data scientist in industry) 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 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 dataframe 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 hypotheses. We have not 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 dataframe operations. Similarly, our survey data and the qualitative interviews validate H6-Helpfulness. Looking closer at the data for completion times, the Pandas condition has a higher variance in completion times, while the Persist condition has minimal variation. This difference could indicate that Persist is easy to learn and can be consistently applied, whereas the ease of using Pandas operations depends on the analysts' experience. Almost all participants had to look up syntax for most Pandas operations. However, even the more proficient participants in our study said they found Persist helpful. We conclude that a tool like Persist can significantly speed up the workflow of most somewhat proficient data scientists, while

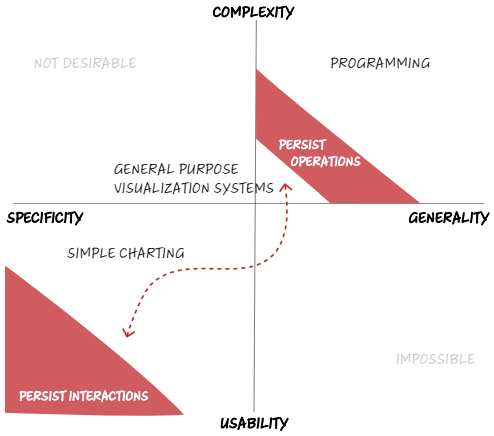

Figure 8: *Conceptual trade-offs of data analysis systems.*

still being an appreciated tool for experts, thereby contributing to making computational data analysis accessible to a wider audience.

In a comparison of the completion time of the two filter tasks (2a and 2b), the difference in the Persist condition is negligible, but 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 Task 2b. This observation makes us consider whether we should provide functionality to copy workflows created with Persist [GCL22].

One critique of our study design could be that we tested tasks and operations that we expected would perform well with Persist, but did not test tasks that cannot be completed with Persist alone, and that, hence, the benefits of Persist shown in the study are not surprising. We counter this by stating that we did test a representative set of operations, but also acknowledge some operations 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 and 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 are 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, whereas 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. Therefore allowing some *operations* that would usually be in the domain of programming languages to be executed with easy-to-use interactive systems, without reducing the overall generality of the data analysis approach.

However, we recognize that Persist has limitations. Whereas data operations on pandas and similar tools are scalable to large datasets with millions of rows, Persist is limited with respect to scalability by what can be plotted in a reasonable way. While scalable visualization solutions for large amount of data are available, they are not implemented in our prototype. Scalability of the interaction provenance is also a concern. Long iterative analysis can result in unwieldy provenance graphs. We can develop more sophisticated approaches for managing large interaction provenance, such as query-based retrieval of states of interest [SGP*19], automatically chunking multiple interactions as a higher-level interaction, or features such as undo-as-delete [HMSA08] to avoid short stray branches when recovering from mistakes. 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 Plotly. Similarly, Persist is currently limited to pandas dataframes and does not yet implement all reasonable operations for dataframes. We believe an abstraction to SQL would open up compatibility with other data structures such as DuckDB and other databases. Whereas poststudy interviews indicate that Persist is easy to pick up after just a short tutorial, we cannot discount the possibilities of non-experts facing certain hurdles. Interactive analysis is not yet common in notebooks, so an interface like Persist can add to the learning curve of notebooks for novice analysts.

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. One aspect that Persist does not simplify is chart creation. Combining Persist with the chart creation technology shown by others [WHS20, EGMP23] would be desirable. 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 the dataframe. Finally, we have compared Persist to traditional analysis, we are interested in comparing Persist to alternative code-generating approaches, such as B2, so that we can develop a better understanding of the trade-offs of both approaches.

10. Acknowledgements

We thank Jake Wagoner for helping with engineering aspects and the Visualization Design Lab for fruitful discussions and feedback. We gratefully acknowledge funding from the National Science Foundation (IIS 1751238, CNS 213756, and CNS-2313998).

References

[AZL*19] ALSPAUGH S., ZOKAEI N., LIU A., JIN C., HEARST M. A.: Futzing and Moseying: Interviews with Professional Data Analysts on Exploration Practices. IEEE Transactions on Visualization and Computer Graphics 25, 1 (2019), 22–31. doi:10.1109/TVCG.2018.2865040. 4

[BE18] BATCH A., ELMQVIST N.: The Interactive Visualization Gap in Initial Exploratory Data Analysis. IEEE Transactions on Visualization and Computer Graphics 24 (2018), 278–287. doi:10.1109/TVCG.2017.2743990. 4

[Bok18] BOKEH DEVELOPMENT TEAM: Bokeh: Python library for interactive visualization. https://docs.bokeh.org/en/1.0.1/docs/citation.html, 2018. 2, 4

[Cen23] CENTER U. A.: Utah Avalanche Center. https://utahavalanchecenter.org/observations, 2023. 2, 6

[CGL20] CUTLER Z. T., GADHAVE K., LEX A.: Trrack: A Library for Provenance Tracking in Web-Based Visualizations. In IEEE Visualization Conference (VIS) (2020), pp. 116–120. doi:10.1109/VIS47514.2020.00030. 6

[CPH*20] CHATTOPADHYAY S., PRASAD I., HENLEY A. Z., SARMA A., BARIK T.: 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 (New York, USA, 2020), CHI '20, Association for Computing Machinery, pp. 1–12. doi:10.1145/3313831.3376729. 4

[DBG*20] DROSOS I., BARIK T., GUO P. J., DELINE R., GULWANI S.: 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 (New York, NY, USA, Apr. 2020), CHI '20, Association for Computing Machinery, pp. 1–12. doi:10.1145/3313831.3376442. 4

[Dra16] DRAGICEVIC P.: Fair Statistical Communication in HCI. In Modern Statistical Methods for HCI, Robertson J., Kaptein M., (Eds.). Springer International Publishing, Cham, 2016, pp. 291–330. doi:10.1007/978-3-319-26633-6_13. 7

[EGMP23] EPPERSON W., GORANTLA V., MORITZ D., PERER A.: Dead or Alive: Continuous Data Profiling for Interactive Data Science. arXiv preprint arXiv:2308.03964 (2023). arXiv:2308.03964. 10

[GCL22] GADHAVE K., CUTLER Z., LEX A.: Reusing Interactive Analysis Workflows. Computer Graphics Forum 41, 3 (2022), 133–144. doi:10.1111/cgf.14528. 3, 10

[GGC*21] GADHAVE K., GÖRTLER J., CUTLER Z., NOBRE C., DEUSSEN O., MEYER M., PHILLIPS J. M., LEX A.: Predicting intent behind selections in scatterplot visualizations. Information Visualization 20, 4 (Oct. 2021), 207–228. doi:10.1177/14738716211038604. 3

[GKHH11] GUO P. J., KANDEL S., HELLERSTEIN J. M., HEER J.: 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 (New York, NY, USA, Oct. 2011), UIST '11, Association for Computing Machinery, pp. 65–74. doi:10.1145/2047196.2047205. 3

[HHB*19] HEAD A., HOHMAN F., BARIK T., DRUCKER S. M., DELINE R.: Managing Messes in Computational Notebooks. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (New York, USA, 2019), CHI '19, Association for Computing Machinery, pp. 1–12. doi:10.1145/3290605.3300500. 3

[HMSA08] HEER J., MACKINLAY J., STOLTE C., AGRAWALA M.: Graphical Histories for Visualization: Supporting Analysis, Communication, and Evaluation. IEEE Transactions on Visualization and Computer Graphics (InfoVis '08) 14, 6 (2008), 1189–1196. doi:10.1109/TVCG.2008.137. 10

[HS88] HART S. G., STAVELAND L. E.: Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In Advances in Psychology, Hancock P. A., Meshkati N., (Eds.), vol. 52 of Human Mental Workload. North-Holland, Jan. 1988, pp. 139–183. doi:10.1016/S0166-4115(08)62386-9. 6

[Hun07] HUNTER J. D.: Matplotlib: A 2D Graphics Environment. Computing in Science Engineering 9, 3 (May 2007), 90–95. doi:10.1109/MCSE.2007.55. 4

[Inc19] INC M.: Microsoft Power BI. https://powerbi.microsoft.com/de-de/, 2019. 3

[IPy15] IPYTHON WIDGET TEAM: Jupyter Widgets — Jupyter Widgets 8.1.1 documentation. https://ipywidgets.readthedocs.io/en/stable/, 2015. 2, 4, 6

[Knu84] KNUTH D. E.: Literate Programming. The Computer Journal 27, 2 (1984), 97–111. doi:10.1093/comjnl/27.2.97. 2, 3

[KPHH11] KANDEL S., PAEPCKE A., HELLERSTEIN J., HEER J.: Wrangler: Interactive Visual Specification of Data Transformation Scripts. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (New York, NY, USA, 2011), CHI '11, ACM, pp. 3363–3372. doi:10.1145/1978942.1979444. 3

[KRH*20] KERY M. B., REN D., HOHMAN F., MORITZ D., WONGSUPHASAWAT K., PATEL K.: Mage: Fluid Moves Between Code and Graphical Work in Computational Notebooks. Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology (Oct. 2020), 140–151. arXiv:2009.10643, doi:10.1145/3379337.3415842. 2, 4

[KRP*16] KLUYVER T., RAGAN-KELLEY B., PÉREZ F., GRANGER B., BUSSONNIER M., FREDERIC J., KELLEY K., HAMRICK J., GROUT J., CORLAY S., IVANOV P., AVILA D., ABDALLA S., WILLING C., JUPYTER DEVELOPMENT TEAM: Jupyter Notebooks – a publishing format for reproducible computational workflows. In 20th International Conference on Electronic Publishing (01/01/16) (2016), Loizides F., Scmidt B., (Eds.), IOS Press, pp. 87–90. doi:10.3233/978-1-61499-649-1-87. 2, 3

[KWR*23] KAFURA A. C. B., WHITCOMB R., RIDDLE J., SALEEM O., TILEVICH D. E., DR. CLIFFORD, SHAFFER A., DR. DENNIS: CORGIS Datasets Project. https://corgis-edu.github.io/corgis/, 2023. 6

[Man23] MANZ, T: Anywidget. doi:10.5281/zenodo.10078842. 6

[mic23] MICROSOFT: Data Wrangler Extension for Visual Studio Code. Microsoft, Nov. 2023. 3

[Plo15] PLOTLY T. I.: Collaborative data science, 2015. URL: https://plot.ly. 2

[RTH18] RULE A., TABARD A., HOLLAN J. D.: Exploration and Explanation in Computational Notebooks. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (Montreal, Canada, 2018), ACM, pp. 1–12. doi:10.1145/3173574.3173606. 2, 3

[Saw09] SAWILOWSKY S.: New Effect Size Rules of Thumb. Journal of Modern Applied Statistical Methods 8, 2 (Nov. 2009). doi:10.22237/jmasm/1257035100. 7, 8

[SGP*19] STITZ H., GRATZL S., PIRINGER H., ZICHNER T., STREIT M.: KnowledgePearls: Provenance-Based Visualization Retrieval. IEEE Transactions on Visualization and Computer Graphics (VAST '18) 25, 1 (2019), 120–130. doi:10.1109/TVCG.2018.2865024. 10

[SO20] SCHMIDT J., ORTNER T.: Visualization in Notebook-Style Interfaces. In Proceedings of the Workshop on the Gap between Visualization Research and Visualization Software (VisGap) (May 2020). doi:10.2312/visgap.20201104. 4

[Tab15] TABLEAU SOFTWARE: Tableau Software. http://www.tableau.com/, Dec. 2015. 3

[Tea] TEAM S.: Streamlit • a faster way to build and share data apps. https://streamlit.io/. (Accessed on 02/28/2024). 4

[VGH*18] VANDERPLAS J., GRANGER B. E., HEER J., MORITZ D., WONGSUPHASAWAT K., SATYANARAYAN A., LEES E., TIMOFEEV I., WELSH B., SIEVERT S.: Altair: Interactive Statistical Visualizations for Python. Journal of Open Source Software 3, 32 (Dec. 2018), 1057. doi:10.21105/joss.01057. 2, 4

[WHS20] WU Y., HELLERSTEIN J. M., SATYANARAYAN A.: B2: Bridging Code and Interactive Visualization in Computational Notebooks. In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology (New York, NY, USA, Oct. 2020), UIST '20, Association for Computing Machinery, pp. 152–165. doi:10.1145/3379337.3415851. 2, 4, 10

[WLH19] WONGSUPHASAWAT K., LIU Y., HEER J.: Goals, Process, and Challenges of Exploratory Data Analysis: An Interview Study. ArXiv (2019). 3, 4