Diff in the Loop: Supporting Data Comparison in Exploratory Data Analysis

APRIL YI WANG, The University of Michigan, USA
WILL EPPERSON, Carnegie Mellon University, USA
ROBERT DELINE, Microsoft Research, USA
STEVEN M. DRUCKER, Microsoft Research, USA

Data science is characterized by evolution: since data science is exploratory, results evolve from moment to moment; since it can be collaborative, results evolve as the work changes hands. While existing tools help data scientists track changes in code, they provide less support for understanding the iterative changes that the code produces in the data. We explore the idea of visualizing differences in datasets as a core feature of exploratory data analysis, a concept we call *Diff in the Loop* (DITL). We evaluated DITL in a user study with 16 professional data scientists and found it helped them understand the implications of their actions when manipulating data. We summarize these findings and discuss how the approach can be generalized to different data science workflows.

Visualize differences, not just "snapshots" or single data iterations

CCS Concepts: • **Human-centered computing** → **Visualization systems and tools**; **Interactive systems and tools**.

Additional Key Words and Phrases: data science programming, exploratory data analysis, data comparison

Track data changes in addition to code changes

ACM Reference Format:
April Yi Wang, Will Epperson, Robert DeLine, and Steven M. Drucker. 2022. Diff in the Loop: Supporting Data Comparison in Exploratory Data Analysis. In *CHI Conference on Human Factors in Computing Systems (CHI '22), April 29-May 5, 2022, New Orleans, LA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3491102.3502123

Data differences/comparison as a first-class citizen in EDA

1 INTRODUCTION

Forage: (of a person or animal) search widely for food or provisions

Data scientists try different transformations, aggregations, and filters until their data is in a state appropriate for the given task [26]. When producing models from their data, data scientists similarly iterate on different model features, architectures, and hyperparameters [1]. Existing tools for tracking changes typically only tackle half of the problem: differences in code. Development environments, for example, allow users to compare differences in notebook code cells between committed revisions [50], and Verdant reduces the burden of foraging code editing histories in Jupyter notebooks [25]. Yet comparing versions of *data* throughout an analysis is just as important [17]. Code differences do not always reveal data differences. For example, removing missing values from one column of a dataset may also affect the distributions of the dataset's other columns. To track the effect that different lines of code have on the data currently requires data scientists to take the initiative to write additional code to browse or plot the data.

Recent work has begun to explore ways for analysts to understand and explain data iterations. Datamations uses animation to explain data transformation pipelines [36], and Chameleon allows analysts to compare data iterations simultaneously with model performance [21]. However, these approaches explain data differences *after* analysis has been done. In this paper, we explore adding visualizations of data differences as a core feature of tools for exploratory data analysis, a concept we call Diff in the Loop (DITL). Our DITL prototype stores a snapshot of the code and runtime

Manuscript submitted to ACM

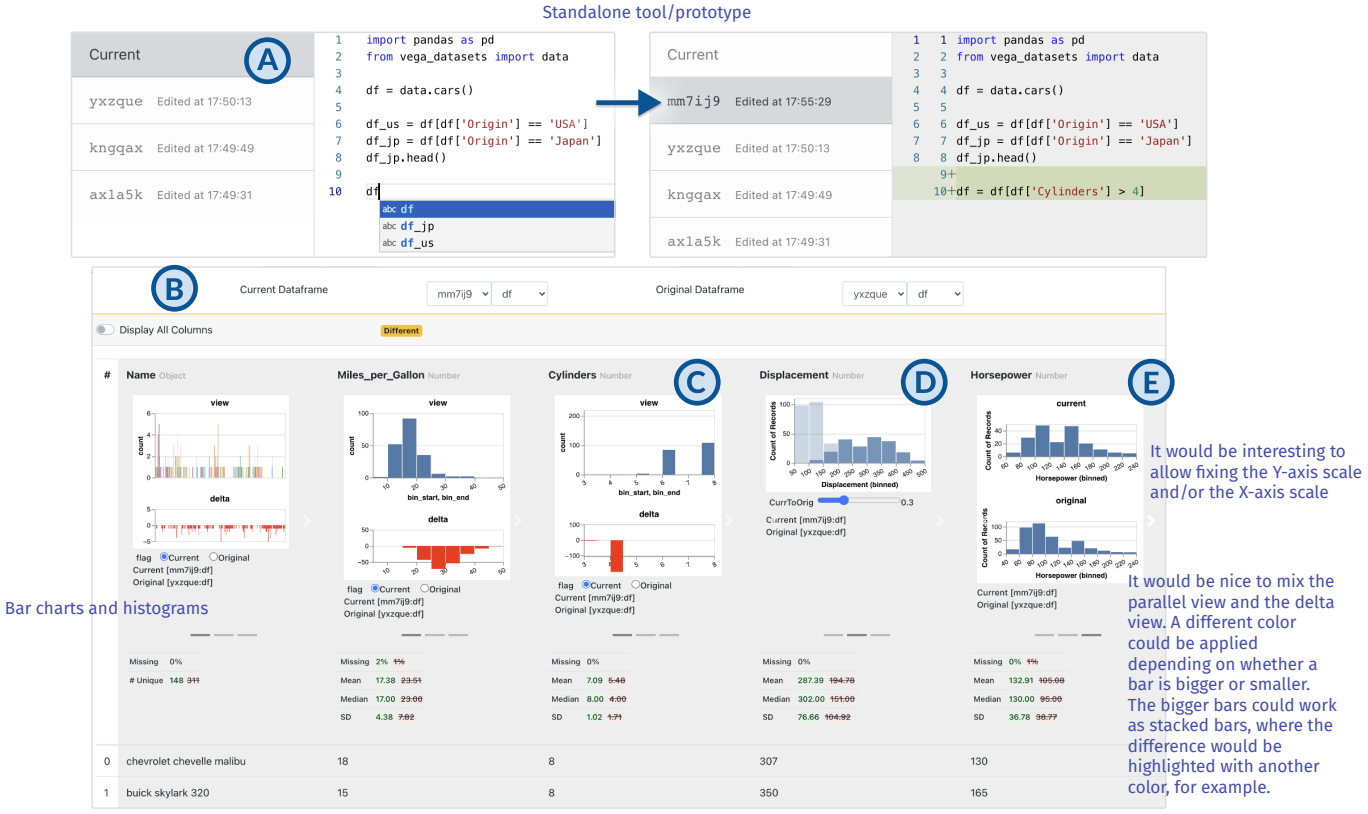

Fig. 1. As users iterate on their data during analysis, they can use DITL to compare data snapshots. Every time users successfully execute code we save a snapshot (A). Users can compare the code using traditional code diffing tools. Users can also use DITL to compare data iterations with interactive visualizations, descriptive statistics, and data preview (B). User can choose three ways to visualize the differences in each column: delta view (C), opacity view (D), and parallel view (E).

variables as users make changes in the code editor. Using a table-based diff view (Fig. 1), users can either compare different datasets or compare the same dataset at different snapshots. When comparing datasets A and B, the user can choose three ways to visualize the differences in each column: plotting histograms of A and B side by side; overlaying histograms of A and B, with cross-fading between them; or as a histogram of the user's chosen dataset (either A or B), plus a plot of the difference in the histogram bucket counts (either A subtracts B or B subtracts A). For each column, the DITL prototype also shows differences in descriptive statistics that are appropriate for that column's data (categorical or quantitative). The DITL view is designed to support both the explicit comparison of datasets and implicitly monitoring the evolution of a dataset as the user transforms it.

For example, Fig. 1 illustrates the effect of filtering a car dataset named `df` to those rows whose `Cylinders` column is greater than 4. The summary under the `Cylinders` column directly reflects this change: the top "view" plot shows a histogram of the current values (all above 4); the bottom "delta" plot shows that this step has the effect of removing rows with values 3 and 4, but keeping rows with values 5, 6, and 8. This column's summary confirms that the filter had the intended effect. Further, the DITL view also shows the effect this filtering step had on the other columns. For example, the distribution of `Miles_per_Gallon` lost the higher end of its distribution, with its median lowering from

23 to 17. Meanwhile, the columns `Displacement` and `Horsepower` lost the lower ends of their distributions. By having these data differences shown during exploratory data analysis, the user can maintain awareness of the effects that code has on the dataset as a whole, not just on columns mentioned in the code. Today, such awareness would require both the initiative and extra effort to write the code oneself to produce the plots and summaries.

We evaluated DITL in a user study with 16 professional data scientists, where participants were asked to finish typical data science programming tasks. They found DITL to be useful for tracking and understanding data changes. Furthermore, DITL improved their awareness of the side effects of some coding activities, guided them towards insights into the data, and reduced their workload for given data science tasks. We discuss the potential to integrate DITL in various data science programming tools and to generalize this approach for tracking changes in user-generated charts. To summarize, our contribution is twofold:

- A demonstration of the benefit of elevating data differences through visualizations to a core feature in a data science programming environment;
- Insights into users' needs and uses for leveraging both code and data differences during exploratory data analytic workflows through a user study with 16 data scientists.

2 BACKGROUND AND RELATED WORK

2.1 Supporting Exploratory Programming

In his 1977 book, Tukey describes exploratory analysis as "detective work" [49, p. 1] where analysts must iteratively create, refine, and explore hypotheses about their data. This pithy description is equally apt for modern data science, where the exploration of the different facets of data is one of the defining characteristics [1, 2, 18, 20]. Today, data scientists still spend up to 80% of their time doing data wrangling tasks like cleaning, filtering, and formatting data tables before they can begin further analysis [7].

In order to support the flexible programming needs in data science, tools like Jupyter Notebook [35] allow users to interleave code with documentation to aid explanations. Observable [32] and Glinda [10] leverage live programming to provide immediate feedback and keep notebook results consistent. Moreover, online data science programming tools like Google Colab [16] and DeepNote [8] allow multiple users to edit the notebook together and execute the code in a shared environment. Other tools enhance the notebook programming environment by providing rich history interaction [24] or visualization provenance [55]. However, the ability to easily compare versions of data in notebooks or other exploratory programming environments remains a challenge.

Visual Analytics systems help data scientists explore their data visually with custom visualizations [11, 43] or recommendations [44, 53, 54]. While these systems allow programmers to visualize a single iteration of their data at a certain point in time, DITL focuses on comparing versions of data across iterations. Most similar to our work are Chameleon [21] and Boxer [15]. Chameleon focuses on the context of data iterations as it relates to model development and thus prioritizes shifts in feature distributions or train and test splits, while Boxer focuses on comparing changes for classification results. However, DITL demonstrates techniques that can be used to show data iterations for many different types of data transformations and focuses on demonstrating the benefit of including data iteration in a broader set of data analysis tasks.

2.2 Making Sense of the Changes

Data scientists often leverage version control systems to track code and output changes. However, popular tools like GitHub [12] compare notebooks as JSON files, making changes hard to read and understand. ReviewNB [37] and VS Code [50] improve the readability of the notebook diffs by rendering code diffs and output diffs side-by-side in an intelligible format. Additionally, some online data science programming tools allow code change tracking for incremental user edits [8, 16]. Our work extended the concept of diffing and versioning control from code assets to data tables.DataFrames

Various tools demonstrate the benefits of interaction with analysis histories during exploratory programming. Kery et al. [25] designed an algorithmic and visualization approach for finding past exploration paths in the long editing histories of computational notebooks; Head et al. [19] used program slicing to gather relevant editing histories into a minimal notebook; Wang et al. [52] proposed connecting team chat messages with code edits to aid in the explanation of editing logs. Additionally, animation has been used to communicate analysis pipelines and history [36], or to make the relationships between two visualizations more clear [27, 47]. DITL builds on these previous approaches by elevating data table differences as a primary concern during analysis.

Comparing and visualizing differences has previously been explored in context of dashboards or static tabular datasets [13, 14, 29–31, 42]. Niederer et al. proposed interactive comparison tools to visualize changes in tabular datasets [31]. Srinivasan et al. found that explicitly visualizing the differences between two bar charts was most effective for comparison [42]. This explicit difference visualization inspired our delta view presented in Section 4.4.3. Furthermore, Gleicher presents three design strategies to support comparison between two datasets through visualizations: juxtaposition of the datasets, superposition of the datasets, and explicitly encoding the relations [13, 14]. We employ all three of these designs in our visualizations of data table differences.

Finally, work from the databases community has proposed the Data Diff problem as finding the best transformation from one dataset to another, and present a tool to find such transformations [45]. Our work differs from this approach in that we assume that an analyst has full access to the code that produced two datasets. Therefore we do not focus on finding a transformation function but rather on presenting the changes in data points affected by these transformations.

3 DESIGN MOTIVATIONS Debugging, insights, and collaboration

To motivate the problem and guide the design, we present three typical usage scenarios that demonstrate how showing both code and data differences would be useful during exploratory data analysis.

3.1 Understanding the Impact of Code Changes in Debugging

In exploratory data analysis, data scientists write code to replace values in data tables, transform and combine data tables, or query subsets of data tables. Debugging data science code involves both ensuring that code changes compile, but that they also produce expected results [38]. However, existing data science code debuggers provide limited support for probing into the impact of code changes [4, 28]. Data science programmers often need to formulate temporary code queries to inspect data tables, which are likely to become stale or commented code that reduces the readability of the analysis scripts or notebooks [39]. Manual inspection is often performed on demand so analysts may miss unexpected impacts if they do not thoroughly explore the effects of code changes. Therefore, it is critical to inspect differences in both code and data throughout analysis. We believe that showing both code and data differences in data science programming environments can directly aid debugging.

Fig. 2. We integrate DITL into a simplified data science programming environment that allows data scientists to edit code, inspect data tables, and compare different data tables. This interface shows that the user is browsing a snapshot tagged 1k9i8j where the edit took place at 12:18:17. (A) Users are able to navigate among saved snapshots, compare code differences and output differences, or switch to the current code editor; (B) Users can edit code in the current code editor which automatically saves a new snapshot upon successful execution, or view code changes in a snapshot; (C) Users can switch between the output panel, the data panel panel, and DITL.

3.2 Gaining Insights in Data Through Comparisons

Data scientists must make decisions throughout exploratory data analysis. Which features should be taken into consideration? How should null values be filled in? Does it matter if this part of the data is dropped? These decisions require making *comparisons* between whether or not a certain step improves the analysis. As opposed to comparing other types of variables, comparing data tables is exploratory and open-ended. Data scientists often need to tailor the comparison strategy according to the task. When tuning hyperparameters, data scientists must formulate a scoring function to compare the quality of the generated data tables. In model development data scientists must consider shifts in feature distribution, train test splits, and model performance when comparing data iterations [21]. In addition, understanding differences in model performance often requires data scientists to consider beyond simply aggregating performance statistics. Comparing between data tables of the results can give them new insights on regions of impact on model changes. These examples demonstrate how comparison is an inherent task within exploratory data analysis.

3.3 Improving Awareness in Collaboration

Lastly, comparing data tables improves data scientists awareness of each others' work in collaborative settings. Data scientists rely on collaboration to improve the quality of their work [56]. Tracking and managing versions of scripts, artifacts, and documentation can help data scientists improve the efficiency of collaboration, reduce duplicated work, and avoid interference with each other [51]. Code versioning and editing sharing mechanisms (e.g., Git) in traditional software engineering can help data scientists managing code iterations when handing off work. However, it is not

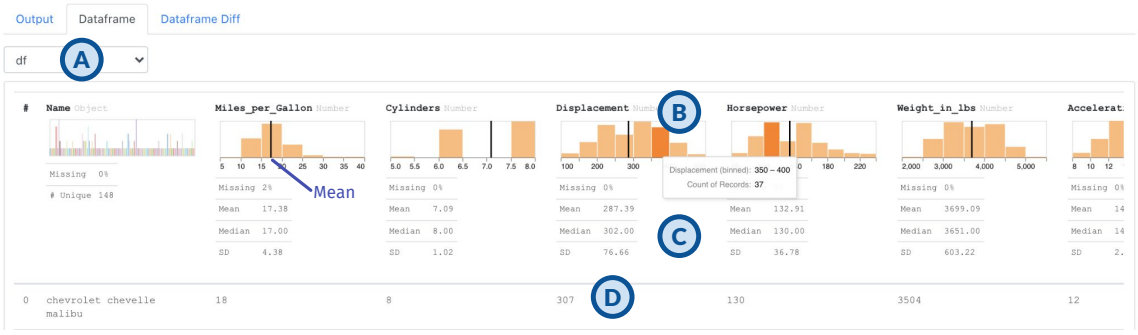

Fig. 3. The data panel allows users to inspect a single data table. (A) Users can select saved data frames from the current code snapshot; (B) The data panel shows the distribution of each column; (C) The data panel shows the summary statistics for each column; (D) The data panel shows a sample of rows from the selected data frame.

straightforward for data scientists to interpret the impact of code changes unless they execute various versions of code and inspect the data tables thoroughly. We believe that showing and tracking both code and data changes can augment the existing data science collaboration tools by improving awareness of changes.

4 SYSTEM DESIGN

To address these use cases, we present DITL, a process of inspecting and comparing versions of data tables using interactive visualizations. We integrate the design into a simplified notebook experience so that we can examine how data scientists use it for comparing data tables. We choose to implement the simplified data science programming environment to highlight the utility of incorporating data table differences during exploratory data analysis without the distraction of other programming features included in existing tools.

4.1 Overview of DITL Study Apparatus

Figure 2 shows an overview of DITL study apparatus. As opposed to Jupyter notebooks, it has a single code editor for editing and running code. Users can make changes in the code editor (Figure 2B) or view the historical contents in previous edits. A snapshot (Figure 2A) is saved upon successful execution, which tracks the code content, output, runtime variable values, and a timestamp. Each snapshot is marked with a unique hashtag to aid in history navigation. Below the code editor, users can switch between the output panel, the data panel, and DITL. The output panel shows the results of users' consoles. The data panel (Figure 3) allows users to inspect the value of a single data table, using a design inspired by existing data table inspectors [34, 46]. As shown in Figure 3A, users can select saved data tables across different code snapshots. For each column, the data panel displays a visualization of the distribution (Figure 3B), summary statistics (Figure 3C), and sample rows (Figure 3D). Next, we elaborate on DITL and explain how the diff views are generated.

4.2 Tracking Runtime Variables

The programming prototype we built is able to collect runtime variable values upon every successful execution. The web-based interface executes Python code and stores the names and values of variables that are dataframes. This approach allows us to create snapshots during each code iteration which are later used to create the dataframe diff

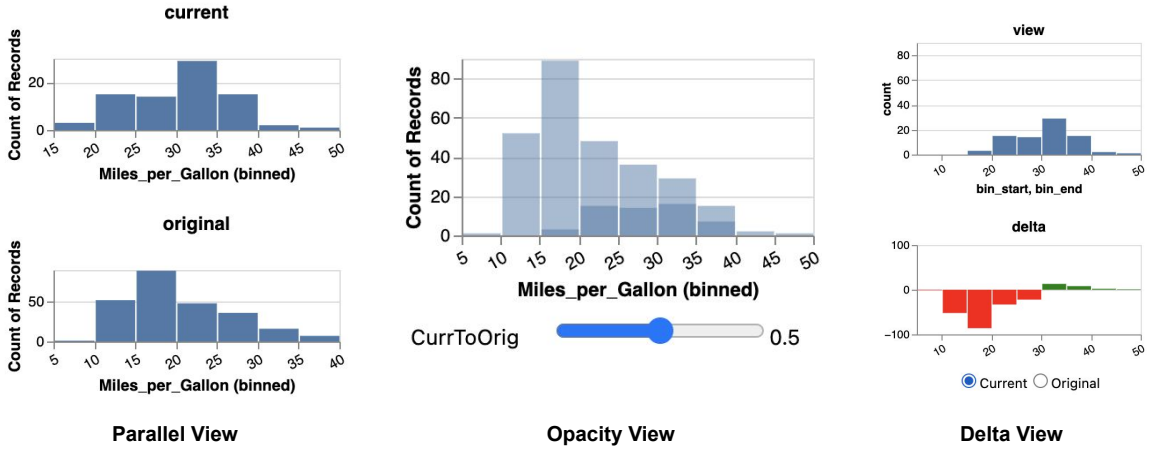

Fig. 4. DITL uses three approaches for rendering data differences: parallel view, opacity view, and delta view.

visualizations. This approach to tracking runtime variable iterations could be easily generalized to other data science programming tools like Google Colab [16], DeepNote [8], or Jupyter Notebooks [22].

4.3 Comparing Changes in Data Frames

In order to visualize the differences between data tables, we must first identify correspondences between two tables. Our notation and method for calculating data table correspondences is inspired by the notation used in the visualization library D3 [6]. In order to align the two data tables, we use the heuristics of comparing data frame index provided by the Pandas package [33]. Given an original (old) data table and a current (new) data table, we use five labels to describe their correspondences. **Both** corresponds to a point that is exactly the same in both the original and current data table. Updates occur when a point has the same primary key but some other column value has changed. **Update-Enter** refers to the newly updated point in the current data table, while **Update-Exit** refers to the old point in the original data table. Lastly, **Enter** corresponds to new points while **Exit** refers to deleted points. The labels are appended as an additional column in the joint data table.

4.4 Rendering Data Frame Diffs

As shown in Figure 1, we use this correspondence information to visualize the differences between two data tables. To eliminate information overload, by default, we only display the columns that have been changed. Users can click on a toggle button to display all columns. Through early pilot testing, we decided on three views for DITL: parallel view, opacity view, and delta view (Figure 4). The design of these views follows the best practices from the visualization community for supporting comparisons [14]. We implemented the interactive visualizations in Vega-Lite.

4.4.1 Parallel View. The parallel view shows the original and current distributions of the column side-by-side. We enable the tooltip to show detailed information about the distribution. While we purposely chose to implement the parallel view in a fashion consistent with the way these are currently displayed in comparison views in ReviewNB [37] or VSCode [50], the parallel view can be easily augmented with the option to display on common axes scales. We are aware of the potential issues with not unifying the axes and we included the view to validate these issues.

4.4.2 Opacity View. While the parallel view allows users to directly inspect and compare distributions of the columns, it is hard to visually compare the shape of the distributions since they may come in different scales. Thus, we designed an opacity view which overlays the distributions on the same axes. This design implements Gleicher's guideline [14] that correspondences can be easier to track when the data is overlaid. We then use the opacity channel to map the "diff-label" information. Users can move the opacity slider to cross-fade between the current and original distributions.

4.4.3 Delta View. In our pilot user testing, users demonstrated the desire to visualize not only the distributions of the current and original data tables, but also the distributions of their differences. Thus, we introduce the delta view to explicitly show the subtraction results. As shown in Figure 4, the delta view contains two parts. The top view shows either the current or original distribution. On the bottom, the delta shows how the current distribution differs from the original by computing $N_{delta} = N_{enter} + N_{update\_enter} - N_{exit} - N_{update\_exit}$. Red bars represent negative values of delta, indicating the decreased counts of the data points falling under the bin, while green bars represent positive values of delta, indicating the increased counts of the data points falling under the bin. We purposely tweak the scale for the delta distribution to help amplify small changes.

5 USABILITY STUDY

We conducted a 60-minute long virtual usability study with each of 16 professional data scientists to understand the support that DITL can provide for common data science programming tasks. In particular, we sought to answer the following research questions:

- Do data scientists find DITL useful for comparing data tables?
- How might DITL provide them with additional insights into the differences between programming iterations?

5.1 Method

5.1.1 Recruitment. We randomly selected 200 data scientists at a large software company based on their job titles and sent them recruitment emails. To be eligible for the study, participants had to self report at least basic experience with Python programming. We recruited 16 participants altogether (3 females, 12 males, 1 prefer not to say). Three of our participants had less than 1 year of professional data science experience, 11 had 1-5 years of professional experience, and two had more than 5 years of professional experience. Their job titles included data scientist (9), senior data scientist (4), principal data scientist (1), research scientist (1), and senior machine learning scientist (1). We compensated participants with a US$25 Amazon gift card.

5.1.2 Study Setup. The usability study was conducted remotely with participants sharing their screens over a video conferencing tool. Since the DITL study apparatus is a web-based programming environment, participants were able to use the tool on their computers within their own choice of browsers and configurations. Each study consists of three sessions — a training session and two experiment sessions. After a brief walkthrough of the prototype, we gave participants a trial task to get familiar with the tool. We presented them with an ongoing code session to explore a dataset about cars [3]. The trial task is scaffolded into four activities: using the data panel to inspect a given data frame; using DITL to compare the differences between two data frames; understanding historical edits to the code and the data frame; and, modifying the current code to include an additional step for exploratory data analysis.

After the training session, we gave participants two existing data science tasks modified from online data science challenges. One task is about customer satisfaction (noted as T1), which is modified from Kaggle [23]. The other task is

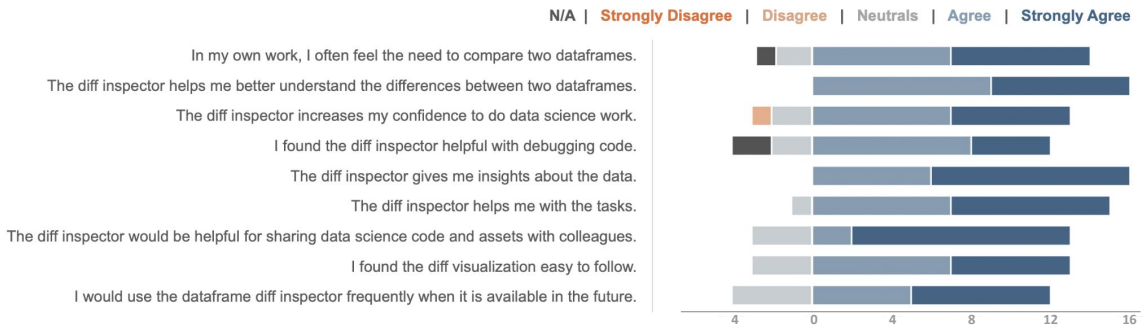

Fig. 5. Participants' responses to the likert scale questions in the post-task questionnaire.

about salary analysis (noted as T2), which is modified from TidyTuesday projects [48]. We chose these two tasks because they are shared on popular data science communities [5, 41] and are perceived to be representative of real-world data science tasks. Since the original challenges are open-ended and time-consuming, we scaffolded the tasks into three subtasks: one for cleaning duplicate presentations in data (noted as S1), one for exploring subsets of the data (noted as S2), and one for evaluating two model prediction results (noted as S3). To maximize the time on experiencing DITL, we provided participants hints and code cheatsheets for the given tasks, and allowed them to ask API-related questions. We counterbalanced the order of the tasks between subjects. For each task, participants are randomly assigned to solve it with or without the DITL. We encouraged participants to think aloud throughout the tasks.

Lastly, we asked participants to fill out a post-study questionnaire and reflect on their experience with the tasks. Two members from the research team observed each study session and took notes. We recorded the screen sharing of the study sessions and transcribed the audio recording.

5.2 Results

5.2.1 The need to compare data tables. Our evaluation showed that comparing data tables is a common activity in various data science tasks. During the study tasks, we frequently observed participants comparing data tables for various purposes. For S1, many participants inspected and compared the data tables before and after changes to validate whether the code edits worked as they expected (14/16). Comparing data tables is also an essential step for generating insights for exploration purposes. For example, for S2, all participants (16/16) compared the subset of the dataset with the original dataset in order to understand the side effects of the filtering query and come up with their next step. Participants also reported the need to compare data tables to make decisions between solutions. For S3, all participants (16/16) evaluated the performance of the models by comparing the model prediction results either using DITL or writing code for inspections. In the post-task questionnaire, most participants (14/16) agreed that they often need to compare two data tables in their own work (Figure 5). For example, P14 mentioned that their work involved collecting new data during model deployment: "One thing we do is comparing the original training data with the scoring snapshots of the weekly changing data. This is something [comparing data iterations] we should do but we did not do as often."

5.2.2 DITL makes comparison easier. We observed several different strategies for comparing data tables. When DITL was not available, participants wrote code to manually understand and compare data tables. For instance, they printed summary statistics, previewed the first five rows of the data tables, manually created distribution plots, or formulated

Two-sample *t*-test: a method used to test whether the unknown population **means** of **two groups** are equal or not

https://www.jmp.com/en_ch/statistics-knowledge-portal/t-test/two-sample-t-test.html

customized queries for examining a specific attribute (e.g., the ratio of female respondents to male respondents). Most participants used DITL (15/16) when it was available during the analysis. As shown in Figure 5, participants agreed (15/16) that DITL helps them with the tasks. They explained why DITL makes comparison easier.

One advantage of DITL is that besides reducing the amount of code that participants wrote, it also eliminated code that was there just for verification or validation purposes. When DITL was not available, participants wrote code for logging and querying attributes. This process would produce additional code, reduce the readability of the analysis, and potentially lead to the rabbit hole of debugging code that was not part of the primary analysis efforts. For example, P4 suggested that DITL helped her maintain a cleaned code space: "This is useful for quick visual inspection across data frames. I find this helps to avoid intermittent logging and debugging during the development process." We counted and compared the total lines of code that participants produced for completing the tasks. Unsurprisingly, participants using DITL wrote significantly fewer lines of code (17.08 lines vs 25.38 lines, $p < 0.001$ two-sample t-test). Some participants mentioned that this tool could be helpful for novice data scientists who are less familiar with relevant APIs (P9) or for explaining changes to people who are not on the technical side (P10): "That [DITL] is way easier; The diff is really helpful for analytical purpose; I think this would help people like me to show the changes to other people who might not know the technical side." (P10)

Next, participants perceived that DITL helps them discover insights about the data (16/16). Participants described the tool "directly explains what is going on" (P10), "allows me to instantly look at the differences" (P16), "gives me a big picture" (P7), and "is helpful for formulating the next steps" (P8). In addition, participants mentioned that the visualizations helped them understand the side effect of code edits: "I was not aware of the changes in column 'Horsepower' when I applied a filter on the column 'Cylinders' until I used the tool." (P6)

To obstruct or interfere with something

5.2.3 Feedback on the Visualizations. Overall, 13 out of 16 participants found the visualizations easy to follow (Figure 5). Two participants mentioned that their lack of familiarity with interactive visualizations "is getting the way" (P4) and wished to "have more practice to use the visualizations" (P9). Participants also made comments on the usefulness of the three different approaches for visualizing the changes. As expected, there was not a single "best" view. Rather, the three views are complementary depending on the task: "Not necessarily every view was useful for different tasks. It is hard to say which one is the best for all. It kind of depends on the task. (P6) "

The parallel view is perceived to be "straightforward" (P1, P6). One participant described the parallel view as the default approach they would use when manually comparing two distributions (P6). This corresponds to our rationale to include the parallel view — to simulate the go-to approach for comparing the distributions by plotting them side-by-side. However, other participants critiqued that this approach "seems not that helpful" (P13) and even "misleading" (P3, P13). They raised concerns that this view was not intuitive for understanding changes and could be misleading due to the inconsistent axes and scales (P3, P13).

Participants had split attitudes towards the opacity view and the delta view. Five participants (P1, P2, P5, P8, P14) were in favor of the opacity view most, and described it as "intuitive" and "easy to understand", particularly for observing shifts in distributions. For the delta view, six participants (P4, P7, P9, P13, P15, P16) explicitly mentioned it being most helpful. They found it particularly useful upon slicing and selecting subsets (P4), providing the exact differences in counts in the tooltip (P15). Yet, some participants reported that it requires more time for them to understand the delta view than the two other views (P6, P8).

5.2.4 Preferences for integration. Participants' feedback on future integration helps us validate the design motivations. Overall, 12 out of 16 participants responded in a positive manner that they would frequently use DITL if it were available

in the future (Figure 5). For debugging and understanding the impact of code changes, 12 out of 16 participants were positive on the usefulness of DITL. Participants explicitly mentioned future usage of "debugging customers' data over time" (P5), "validating results of cleaning" (P15), "time travel debugging" (P2), and "debugging during the dev process" (P4). For supporting decision making, all participants agreed that DITL gives them insights about the data. In particular, P3 described how DITL can be useful to compare A/B experiments: "Looking for distributional shifts between A/B experiments. Where the distributional information is hard to summarize into a neat hypothesis test, the visual chart really helps." Lastly, 13 out of 16 participants agreed that DITL would be helpful for sharing data science code and assets with colleagues.

Participants described how they see DITL working in their own data science workflows. They mentioned integrating DITL in existing data science IDEs like PyCharm (P15), Jupyter notebooks (P6, P14), RStudio (P10, P13), and VS Code (P7, P8, P9) for tracking and comparing data tables. Some participants mentioned data science collaboration tools, for example, integrating DITL as part of the git versioning experience (P1), or augmenting real-time collaborative editing tools like Google Colab (P16) with DITL.

In addition, participants provided suggestions to further improve the comparison feature. Participants wanted tailored comparisons over certain data types. For example, P15 suggested adding visualizations to demonstrate text attributes, such as word length, number of characters or character sets, different topics. Participants also mentioned the need to compare visual outputs beyond data tables: "Maybe in the future, users can compare other kinds of graphs than distribution plots." (P16)

6 DISCUSSION AND FUTURE WORK

6.1 Towards a Design Space for Visualizing Data Comparisons

Since the goal of our project is to investigate the idea of data comparison in exploratory data analysis, we did not extensively explore the design space for these visualizations nor did we evaluate these possible designs. What we learned at this stage suggests empirical evidence for the utility of using DITL in exploratory analysis. Participants felt that the effectiveness of the visualizations themselves greatly depended on the task at hand and that there was no single visualization that fits all circumstances. For example, the opacity view might be more suitable for observing the trend of shifting in distributions (e.g., correcting skewed distributions through log-transforms); while the delta view might be more suitable for showing slicing and filtering to highlight the changes on individual data bins. In addition, our approach of encoding the diff information in additional channels can be extended to create other types of visualizations, for example, a grouped bar chart rendering the current and original distributions along the same axes, or a facet view showing the distribution of data points marked as "new" or "absent". Future work can continue to explore this design space and evaluate the usefulness of the views for various data science tasks.

6.2 Generalizing from Comparing Data Tables to Comparing Arbitrary Charts

DITL demonstrates the idea of tracking and visualizing changes in data tables in data science programming environments. We further argue that the same techniques used for visualizing the differences in iterative changes of data tables can be generalized to visualizing changes on a wide variety of charts. Typically, data scientists make two types of changes on charts: changing the underlying data or changing the visual representations. If the visual representation and data schema remain the same while only the underlying data changes, a similar approach can be used to first combine the original and current data tables to encode the diff information for each data point. This diff information can subsequently

be rendered with an unused channel (e.g., opacity, color, facet, or z-axis) in the visual representations. Interactions such as sliders, selections, or brushes can be used to switch between original and current charts. In addition, we can filter the combined data table and explicitly render the subtractions. For example, the delta view can be generalized in charts to represent the visual differences in the visualization. On the other hand, if the visual representation or the data schema changes (e.g., table pivoting), there is an opportunity to combine the stateful interactive visualizations with animations (e.g., SandDance [40], Datamation [36], Gemini [27]) to explain the transitions. Lastly, if the changes in charts are multifold, future work can look into ways to break the changes into the combination of data changes and visual representation changes.

6.3 Integrating DITL in Data Science Programming Environments

In this paper, we demonstrate the idea of DITL in a customized data science programming tool. To integrate DITL in existing data science programming environments, both scalability and task complexity must be considered. In particular, the timescale for creating snapshots of the data iterations should be tailored to the context. For programming IDEs that allow execution of script files (e.g., PyCharm), tool designers can leverage built-in debuggers to track variable values upon each execution and map versions of variable values to the snapshot of the scripts. For REPL-based programming environments that allow interactive execution of code snippets (e.g., Jupyter Notebook), mapping the versions of variable values with the execution orders and the state of the notebooks can be a challenging task. Future work can explore how approaches used in foraging code versions (e.g., Verdant [25], Gather [19]) can be extended for foraging data iterations. In collaborative data science programming environments, the timescale for creating snapshots should be tailored towards tracking data iterations and hand-offs between collaborators. For example, versioning tools like Git or real-time editing tools like Google Colab can support the diffing of the data tables and charts when synchronizing collaborators' edits. Lastly, the idea of comparing data changes can be helpful in live programming environments. Live programming is a programming paradigm recently emerging in data science communities (e.g., Observable Notebook [32], Glinda [10]). Compared to REPL-based programming, live programming updates the execution immediately upon editing [9]. Although live programming is favored for providing a responsive and consistent experience for exploratory data analysis, the live experience hides history and may result in mismatched expectations for the automatic execution [9]. Future work can explore the idea of showing both code and data iterations in live programming environments for browsing and resurrecting histories.

6.4 Limitations

6.4.1 Limitations of DITL. DITL is tailored towards comparing data tables with changes to the column values without altering the schema. DITL is able to detect small changes to the schema such as adding, deleting, and renaming column names, while not able to handle full schema transformations like data pivoting. Recent work [36] has used animations to explain operations such as pivoting that might be incorporated into future work. In addition, DITL only compares two data tables. Future work can explore ways to make comparisons between multiple data tables.

6.4.2 Limitations of the Evaluation. Our user study has several limitations. First, in order to control the complexity of the tasks and the duration of the study, we gave participants data science tasks modified from online challenges instead of evaluating the tool with their own tasks. Second, we scaffold the tasks to ensure that novice data scientists were capable for performing the required tasks. To further prevent diluting the focus of the study, we provided immediate, verbal assistance to them when they got stuck on the programming tasks. We did not evaluate performance in terms of

Priming effect: it occurs when an individual's exposure to a certain stimulus influences his or her response to a subsequent stimulus, without any awareness of the connection

https://thedecisionlab.com/biases/priming

time as we expected this might be affected by participants' familiarity with the tool. Most statements in the post-task questionnaire are positively framed, which could cause a priming effect. Future work should consider long-term deployment to further examine the usefulness of the tool in open-ended, real-world data science tasks.

7 CONCLUSION

This paper presents the idea of Diff In The Loop (DITL), integrating data differences through visualizations as a first class citizen in data science programming environments. We illustrate the usage of comparing data tables in three usage scenarios grounded in prior literature. We implement a prototype that incorporates DITL and show how comparing data tables through visualizations can help in exploratory data analysis. The evaluation of this system confirmed the needs and benefits of showing both code and data differences during exploratory data analytic workflows. In particular, DITL helps data scientists understand the implications of their actions when manipulating data.

ACKNOWLEDGMENTS

We thank all of our participants for their help in the study, and the anonymous reviewers for their valuable feedback.

REFERENCES

[1] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 291–300.

[2] Saleema Amershi, M. Cakmak, W. B. Knox, and T. Kulesza. 2014. Power to the People: The Role of Humans in Interactive Machine Learning. *AI Mag.* 35 (2014), 105–120.

[3] Cars 2021. http://lib.stat.cmu.edu/datasets/

[4] Souti Chattopadhyay, Ishita Prasad, Austin Z Henley, Anita Sarma, and Titus Barik. 2020. 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*. 1–12.

[5] Ruijia Cheng and Mark Zachry. 2020. Building Community Knowledge In Online Competitions: Motivation, Practices and Challenges. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW2 (2020), 1–22.

[6] D3.js 2021. https://d3js.org/

[7] Tamraparni Dasu and Theodore Johnson. 2003. *Exploratory data mining and data cleaning*. Vol. 479. John Wiley & Sons.

[8] Deepnote 2021. https://deepnote.com/

[9] Robert DeLine and Danyel Fisher. 2015. Supporting exploratory data analysis with live programming. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 111–119.

[10] Robert A DeLine. 2021. Glinda: Supporting Data Science with Live Programming, GUIs and a Domain-specific Language. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–11.

[11] Katarina Furmanova, Samuel Gratzl, Holger Stitz, Thomas Zichner, Miroslava Jaresova, Alexander Lex, and Marc Streit. 2020. Taggle: Combining overview and details in tabular data visualizations. *Information Visualization* 19, 2 (2020), 114–136.

[12] GitHub 2021. https://github.com/

[13] Michael Gleicher. 2017. Considerations for visualizing comparison. *IEEE transactions on visualization and computer graphics* 24, 1 (2017), 413–423.

[14] Michael Gleicher, Danielle Albers, Rick Walker, I. Jusufi, C. Hansen, and Jonathan C. Roberts. 2011. Visual comparison for information visualization. *Information Visualization* 10 (2011), 289 – 309.

[15] Michael Gleicher, Aditya Barve, Xinyi Yu, and Florian Heimerl. 2020. Boxer: Interactive comparison of classifier results. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 181–193.

[16] Google Colab 2021. https://colab.research.google.com

[17] Julien Gori, Han L Han, and Michel Beaudouin-Lafon. 2020. FileWeaver: Flexible File Management with Automatic Dependency Tracking. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 22–34.

[18] Philip Jia Guo. 2012. *Software tools to facilitate research programming*. Stanford University.

[19] Andrew Head, Fred Hohman, Titus Barik, Steven M Drucker, and Robert DeLine. 2019. Managing messes in computational notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–12.

[20] Charles Hill, Rachel Bellamy, Thomas Erickson, and Margaret Burnett. 2016. Trials and tribulations of developers of intelligent systems: A field study. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 162–170. https://doi.org/10.1109/VLHCC.2016.7739680

[21] Fred Hohman, Kanit Wongsuphasawat, Mary Beth Kery, and Kayur Patel. 2020. Understanding and visualizing data iteration in machine learning. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

[22] Jupyter 2021. https://jupyter.org

[23] Kaggle Starbucks Satisfactory Survey 2021. https://www.kaggle.com/mahirahmzh/starbucks-customer-retention-malaysia-survey?select=Starbucks+satisfactory+survey.csv

[24] Mary Beth Kery, Amber Horvath, and Brad Myers. 2017. *Variolite: Supporting Exploratory Programming by Data Scientists*. Association for Computing Machinery, New York, NY, USA, 1265–1276. https://doi.org/10.1145/3025453.3025626

[25] Mary Beth Kery, Bonnie E John, Patrick O'Flaherty, Amber Horvath, and Brad A Myers. 2019. Towards effective foraging by data scientists to find past analysis choices. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–13.

[26] Mary Beth Kery and Brad A Myers. 2017. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 25–29.

[27] Younghoon Kim and Jeffrey Heer. 2020. Gemini: A grammar and recommender system for animated transitions in statistical graphics. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2020), 485–494.

[28] Sam Lau, Ian Drosos, Julia M Markel, and Philip J Guo. 2020. The Design Space of Computational Notebooks: An Analysis of 60 Systems in Academia and Industry. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–11.

[29] Po-Ming Law, Rahul C Basole, and Yanhong Wu. 2018. Duet: Helping data analysis novices conduct pairwise comparisons by minimal specification. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 427–437.

[30] Po-Ming Law, Subhajit Das, and Rahul C Basole. 2019. Comparing apples and oranges: Taxonomy and design of pairwise comparisons within tabular data. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–12.

[31] Christina Niederer, Holger Stitz, Reem Hourieh, Florian Grassinger, Wolfgang Aigner, and Marc Streit. 2017. TACO: visualizing changes in tables over time. *IEEE transactions on visualization and computer graphics* 24, 1 (2017), 677–686.

[32] Observable 2021. Observable: the magic notebook for exploring data and thinking with code. https://observablehq.com/

[33] pandas - Python Data Analysis Library 2021. https://pandas.pydata.org

[34] PandasGUI 2021. https://github.com/adamerose/PandasGUI

[35] Jeffrey M Perkel. 2018. Why Jupyter is data scientists' computational notebook of choice. *Nature* 563, 7732 (2018), 145–147.

[36] Xiaoying Pu, Sean Kross, Jake M Hofman, and Daniel G Goldstein. 2021. Datamations: Animated Explanations of Data Analysis Pipelines. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–14.

[37] ReviewNB 2021. https://www.reviewnb.com/

[38] El Kindi Rezig, Ashrita Brahmaroutu, Nesime Tatbul, Mourad Ouzzani, Nan Tang, Timothy Mattson, Samuel Madden, and Michael Stonebraker. 2020. Debugging large-scale data science pipelines using dagger. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2993–2996.

[39] Adam Rule, Aurélien Tabard, and James D Hollan. 2018. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[40] SandDance 2021. https://microsoft.github.io/SandDance/

[41] Nischal Shrestha, Titus Barik, and Chris Parnin. 2021. Remote, but Connected: How# TidyTuesday Provides an Online Community of Practice for Data Scientists. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1 (2021), 1–31.

[42] Arjun Srinivasan, M. Brehmer, Bongshin Lee, and S. Drucker. 2018. What's the Difference?: Evaluating Variations of Multi-Series Bar Charts for Visual Comparison Tasks. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (2018).

[43] Chris Stolte, Diane Tang, and Pat Hanrahan. 2008. Polaris: A System for Query, Analysis, and Visualization of Multidimensional Databases. *Commun. ACM* 51, 11 (Nov. 2008), 75–84. https://doi.org/10.1145/1400214.1400234

[44] Guo-Dao Sun, Ying-Cai Wu, Rong-Hua Liang, and Shi-Xia Liu. 2013. A survey of visual analytics techniques and applications: State-of-the-art research and future challenges. *Journal of Computer Science and Technology* 28, 5 (2013), 852–867.

[45] Charles Sutton, Timothy Hobson, James Geddes, and Rich Caruana. 2018. Data Diff: Interpretable, Executable Summaries of Changes in Distributions for Data Wrangling. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 2279–2288. https://doi.org/10.1145/3219819.3220057

[46] Sweetviz 2021. https://pypi.org/project/sweetviz/

[47] John R Thompson, Zhicheng Liu, and John Stasko. 2021. Data animator: Authoring expressive animated data graphics. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–18.

[48] TidyTuesday - Weekly Challenge 2021. https://github.com/rfordatascience/tidytuesday/blob/master/data/2021/2021-05-18/readme.md

[49] John W Tukey et al. 1977. *Exploratory data analysis*. Vol. 2. Reading, Mass.

[50] VS Code 2021. https://code.visualstudio.com/

[51] April Yi Wang, Anant Mittal, Christopher Brooks, and Steve Oney. 2019. How data scientists use computational notebooks for real-time collaboration. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–30.

[52] April Yi Wang, Zihan Wu, Christopher Brooks, and Steve Oney. 2020. Callisto: Capturing the" Why" by Connecting Conversations with Computational Narratives. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

[53] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2015. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE transactions on visualization and computer graphics* 22, 1 (2015), 649–658.

[54] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, A. Anand, J. Mackinlay, Bill Howe, and J. Heer. 2017. Voyager 2: Augmenting Visual Analysis with Partial View Specifications. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (2017).

[55] Yifan Wu, Joseph M Hellerstein, and Arvind Satyanarayan. 2020. B2: Bridging code and interactive visualization in computational notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 152–165.

[56] Amy X Zhang, Michael Muller, and Dakuo Wang. 2020. How do data science workers collaborate? roles, workflows, and tools. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW1 (2020), 1–23.