TOWARD REPRODUCIBLE AND REUSABLE VISUAL ANALYSIS

by

Kiran Bhaskar Gadhave

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computing

School of Computing

The University of Utah

May 2024

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Kiran Bhaskar Gadhave**
has been approved by the following supervisory committee members:

Alexander Lex ,	Chair(s)	**3/26/2024** Date Approved
Miriah Dawn Meyer ,	Member	**3/27/2024** Date Approved
Jeffrey Phillips ,	Member	Date Approved
Marc Streit ,	Member	**3/27/2024** Date Approved
Vivek Srikumar ,	Member	Date Approved

by **Mary W. Hall** , Chair/Dean of
the Department/College/School of **Computing**
and by **Darryl P. Butt** , Dean of The Graduate School.

ABSTRACT

With data growing in scale and complexity, interactive visualizations are increasingly important in data analysis. However, interactive visual analysis lacks the reproducibility and reusability of computational analysis using code or scripts. This dissertation aims to improve the reproducibility and reusability of visual analysis, improving trust in the visual analysis process and allowing the use of the analysis in a different context.

While computational analysis using languages like R or Python is inherently reproducible and reusable, interactive visual analysis often remains ad hoc and difficult to capture, reproduce, and reuse. Hybrid approaches, such as using multiple tools, have compatibility issues and lack reproducibility within the interactive components. In computational notebooks, code and interactive visualizations have two major gaps: results of interactions cannot be used in code, and interactions with visualizations are lost upon cell re-execution or notebook restarts, hindering the reproducibility and reusability of visual analysis in notebooks.

The dissertation makes four contributions toward addressing the above issues: 1) a software library to capture and replay the interaction provenance; 2) techniques to capture the analyst's pattern-based intent and annotations to make the interaction provenance semantically meaningful; 3) techniques to reuse the semantically meaningful interaction provenance on updated datasets and curate reusable workflows that can be reused on updated datasets as well as different analysis environments; and 4) techniques to leverage the interaction provenance to bridge the gaps between code and interaction in computational notebooks. These techniques improve the reproducibility and reusability of visual analysis, therefore improving trust, reducing the gaps between computational and interactive analysis, and moving us closer to achieving a literate visual analysis framework.

To Aai, Baba, my wife Anagha, and our cats Smaug and Sauron.

CONTENTS

ACKNOWLEDGEMENTS

As I culminate this incredible journey of my PhD, I would like to express my appreciation to everyone who supported me through my Ph.D. First and foremost, I extend my deepest gratitude to my advisor and mentor, Alexander Lex. Alex gave me the opportunity and his support has been unwavering throughout my Ph.D. journey. The opportunity you provided me and the faith you placed in my abilities have been instrumental in my growth and success and I couldn't have done it without you. I would like to thank my supervisory committee — Alexander Lex, Miriah Meyer, Jeff Phillips, Marc Streit, and Vivek Srikumar. Your feedback and insights have been a great help in shaping my research.

A special note of thanks for all my co-authors whose contributions and feedback made my projects possible. I am particulary grateful to Zach Cutler for being an outstanding co-author, it was fun working with you. I would also like to express my heartfelt thanks to the members and alumni of the Visualization Design Lab. Carolina, Jimmy, Jen, Haihan, Devin, Max, Jack, and Jake for being my comrades through this journey. This dissertation is not just a reflection of my efforts but a testament to the collective support, guidance, and encouragement of each one of you. I am deeply grateful for your support. Thank you for being part of this remarkable chapter of my life.

This dissertation extends published works for which we have permission to re-use. © 2020, IEEE. Reprinted [1], with permission, from Cutler, Gadhave, and Lex, Trrack: A Library for Provenance-Tracking in Web-Based Visualizations, IEEE VIS Short Papers, 2020. ©2021 SAGE. Reprinted [2], with permission, from Gadhave, Görtler, Cutler, Nobre, Deussen, Meyer, Phillips, and Lex, Predicting Intent Behind Selections in Scatterplot Visualizations, Information Visualization, 2021. ©2022 Computer Graphics Forum. Reprinted [3], with permission, from Gadhave, Cutler and Lex, Reusing Interactive Analysis Workflows, EuroVis, 2022.

This dissertation was supported by the National Science Foundation (IIS 1751238, CNS 2213756).

CHAPTER 1

INTRODUCTION

In this dissertation, we introduce techniques to enhance the reproducibility and reusability of the visual analysis process. As we witness unprecedented growth in the scale and complexity of data, the role of data visualization in understanding this information and making it accessible has become increasingly pivotal. Interactive visualizations are intuitive, leverage human perceptual capabilities, offer a dynamic means to explore data, and are, therefore, integral to the data analysis. However, despite growing popularity, interactive visual analysis often remains ad hoc and is challenging to reproduce or reuse.

1.1 Data Analysis Methodologies

Data analysis methodologies generally fall into two broad categories: 1) computational analysis and 2) interactive visual analysis.

1.1.1 Computational Data Analysis

The computational analysis approach offers high flexibility and expressivity. Such analysis involves using code and scripts in programming languages such as Python and R. These languages, coupled with specialized libraries and frameworks, offer a powerful approach to data analysis. Code/script-based analysis is particularly suited for situations where custom analysis workflows are needed or when dealing with large datasets that require sophisticated statistical or machine learning techniques. However, despite the expressivity and flexibility of programming, it often presents a steep learning curve, posing a barrier for those without a programming background. This complexity can make even simple data analysis tasks time-consuming and outputs challenging to interpret. Moreover, maintaining a codebase requires good software development practices, adding to the learning curve.

1.1.2 Interactive Visual Analysis

Interactive visual analysis is highly intuitive. Interactive visual analysis tools represent the data visually and support interactions like selections, filters, and data transformation operations for analysis, data wrangling, and processing tasks. Examples of visual analysis tools include Tableau [4] and PowerBI [5]. Visual analysis tools are user-friendly and do not require programming knowledge, making them accessible to novice users. Visual analysis tools are valuable for exploratory data analysis to uncover patterns, relationships, and insights. However, these tools also have their drawbacks. Many specialized tools are designed with a specific end-user or task in mind, which can limit flexibility in visualizations and interactions [6]. On the flip side, generic tools often introduce complexity. Interactive tools also often struggle with large datasets and may not support custom analyses and interactions not built into the tool.

Most important for this dissertation, though, is that the two methodologies have significant differences in the reproducibility and reusability of the analysis conducted between them, which we will discuss next.

1.1.3 Reproducibility

A growing concern over reproducibility marks the current state of scientific research. A 2016 Nature survey by Baker [7] found that 52% of scientists believe we are in the midst of a "reproducibility crisis." This concern is further supported by the fact that over 70% of researchers could not reproduce the experiments of others, and more than half have struggled to reproduce their experiments. A reproducible analysis should produce the same results when repeated with the same data. To verify the reproducibility of an analysis, analysts must share not only the results and the data but also the analysis process and their rationale. Pratt points out reproducibility may not be a relevant or desirable goal for all research methodologies, especially in qualitative research where transparency is more valuable than replicable results [8]. Reproducibility is critical for establishing trust and credibility when using quantitative methods and data analysis. Therefore, these statistics call into question the reliability of scientific findings and highlight the need for more reproducible methods. Multiple factors affect the reproducibility of an analysis session. Selective reporting, publishing pressure, fraud, etc. rely on integrity of the analyst or stem

from systemic issues. There are other factors for low reproduciblity which the analyst can address like availability of analysis steps/data or lack of validation by the original analyst. Developing methods for easily recording, replaying and sharing the analysis can address some factors in reproducibility crisis.

Computational analysis is potentially reproducible. Such analysis uses code/scripts that can be run deterministically to produce the same results for the same inputs. Provenance of code development can be tracked using version control, and most computational programming environments support documentation of the analysis or code using code comments or text representations. The literate programming approach proposed by Knuth [9] exemplifies the extent to which documentation and code can be co-located, greatly enhancing reproducibility. As much of the software does, computational analysis faces challenges with long-term reproducibility. These challenges include changing or discontinued dependencies, data unavailability, lack of documentation, or incompatibility with the computing environment (such as the operating system). However, these challenges can be addressed by employing practices and technologies like robust documentation processes, long-term backup of data, version control for managing code, and containerization for archiving computing environments.

Visual analysis often falls short with respect to reproducibility. A core limitation of most visual analysis tools is their lack of support for tracking the analysis process, making it challenging to retrace the steps taken. The analysis process can be tracked by tracking the underlying interactions. Such a sequence of interactions from an analysis is called an interaction provenance. Some visualization tools offer basic provenance tracking capabilities, often limited to supporting undo/redo functionality. When available, the provenance is often a log of keyboard and mouse events that capture the *what*, but not the *why*, behind the interactions. To understand the *why*, we need to know the analyst's rationale for an interaction, which can include patterns in the data or domain knowledge. Analysts typically rely on manual notes or recordings to capture the analysis steps. Such manual note-taking adds effort for the analyst and is challenging to do consistently. Further, the lack of support for annotating the analysis steps means contextualizing their analysis rationale is not straightforward.

The statistics and concerns about reproducibility in scientific research underscore a

critical need for more robust methods. The challenge for interactive visual analysis lies in capturing and sharing the analysis process and the rationale behind it. Considering this challenge, we pose the following research question: **Can we effectively capture the interactions and the analyst's rationale for the interactions to reproduce the visual analysis?**

1.1.4 Reusability

Reusability is another area where computational and interactive visual analysis diverges. Reuse is applying the same analysis methods to different datasets or contexts. In an ever-changing data landscape — such as a business updating sales figures, a public health agency revising health reports, or a weather station collecting new meteorological data — applying a consistent, reliable analysis process to new data are invaluable, saving time and resources.

Computational analysis excels in reusability. Scripts and programs can be adapted and re-run on new datasets with minimal modifications, making code-based analysis efficient and conducive to standardizing workflows and collaboration. The emergence of open-source communities, data analysis libraries [10], [11], and platforms [12], [13] has further facilitated the sharing and reuse of code, improving collaboration.

However, interactive visual analysis often lacks such adaptability. Visual analysis has challenges reproducing the analysis; therefore, reusing the interactions presents an even more formidable challenge. Visual analysis is user-friendly for data exploration but frequently falls short in its ability to export or translate interactive steps into a reusable format. When the dataset updates, a particular data operation might need slight tweaking to be reapplied to the new data. Low-level interaction provenance based on keyboard and mouse events does not support such parameterization and reuse. Visual analysis is often nonlinear and iterative; therefore, the interaction process can be extensive and include multiple analysis paths, which might have alternate analysis approaches, dead-ends, or paths created by correcting mistakes. We often want to reuse only repetitive parts of the analysis (such as data cleaning steps) or parts that lead to valuable insights. Poor support for capturing interaction provenance makes curating one or more useful parts difficult.

The constantly changing data landscape necessitates the need for reusable analyses that can adapt to new data. Interactive visual analysis struggles with adaptability and

capturing interaction provenance. In light of these struggles, we ask the following research question: **Can we meaningfully apply (parts of) the interaction provenance of a visual analysis process to an updated dataset?**

1.2 Hybrid Approaches to Data Analysis

We are unlikely to have data analysis solutions with both the simplicity of visual analysis tools and the expressiveness of programming languages. A hybrid approach would allow 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. Such a hybrid approach can be achieved using multiple analysis tools or a single well-integrated platform supporting visual and code-based analysis.

1.2.1 Multiple Tools: Picking the Best Tool for the Job

Complex data analysis pipelines often employ multiple tools for different analysis tasks. Doing so offers a comprehensive approach to processing, understanding, and interpreting complex data. This approach leverages the strength of various tools such as SQL and databases for managing the data, statistical software like SAS and SPSS for statistical analysis, programming languages like Python and R for advanced data analysis and machine learning, and visualization tools like Tableau and Power BI for interactive dashboards and reporting. The primary advantage of this approach is versatility, allowing the analysts to use the best tool for the job.

Apart from the steep learning curve associated with multiple tools, this approach offers another major challenge — compatibility between tools. Different tools have varying levels of support for capturing the analysis steps, with many tools having none. Application programming interfaces (APIs) for software and interchange formats for data enable communication between different systems. However, there is no standard way to capture the interactions to enable sharing of the provenance between tools. A standard specification or interchange format for interaction provenance could enable tighter integration between analysis tools and enable seamless switching between tools that adhere to the standard.

A popular way to continue the analysis from one tool to another is to export/import the data between these tools. Data is usually transferred using popular data interchange

formats like CSV, JSON, XML, etc. These formats often have strict specifications on how the data can be represented; therefore, different tools can independently develop support for these formats. However, the data interchange format cannot store the analysis provenance that was used to curate the dataset. Therefore, the analysis context is often lost when switching from one tool to another using only the data. The loss of analysis context can lead to issues with misinterpretation of the data, e.g., incorrect parsing of dates due to different standards or repetition of analysis steps such as converting a column of 1s and 0s to Boolean instead of integer.

Using multiple tools in complex data analysis pipelines presents a unique set of challenges, particularly with sharing analyses between different tools. While the versatility of using the best tool for each task is advantageous, it also leads to complexities in integrating these tools seamlessly. Therefore, the pertinent research question that comes up is: **How can we reuse interactions captured from one analysis tool in another?**

1.2.2 Computational Notebooks: Best of Both Worlds?

Computational notebooks are not a novel concept, but recent advancements in cloud-based notebooks and interactive outputs have increased their popularity even more. Mike Bostock, the creator of *d*3 [14] and Observable [13] notebook platform, describes a *notebook* as "an interactive, editable document defined by code" and "a computer program designed to be easier to read and write by humans" [15]. This definition aligns closely with the literate programming paradigm proposed by Knuth[9], which emphasizes making computer programs reproducible, reusable, and easily maintainable through a mix of natural language exposition and source code.

Modern notebooks support interactive visualizations in the output either natively (as in Observable) or with the help of libraries (as in Jupyter Notebooks). In theory, computational notebooks offer the best of both worlds: Highly expressive, documentable code paired with interactive visualizations.

However, is this truly the case?

Whereas computational notebooks like Jupyter support interactive visualizations, these are often a dead end. The current integration of visualizations with code is typically unidirectional: Plots are generated for analysis or storytelling, but they do not facilitate data

manipulation. Libraries such as Vega-Altair [16], bokeh [17], and IPython widgets [18] enable the creation of complex interactive visualizations in Jupyter Notebooks. However, the results of the interactions in these visualizations, like selections and filters, are not accessible in the code. Wu et al. [19] refer to this as the **semantic gap** in computational notebooks.

Wu et al. describe another type of gap, which they refer to as a **temporal gap**: The gap between the persistence of code cells and interactions in the outputs. Changes in code cells are persistent across notebook saves and can be re-executed. However, interactions within visualizations are transient and not saved, even across cell re-executions. To make insights gained through interactive analysis in such outputs permanent, analysts must extensively document the interactions, which can be burdensome [20].

Although a code-based analysis can be documented, saved, and re-executed, an interactive visual analysis remains temporary and ad hoc. Consequently, because of the semantic and temporal gaps, analysis in computational notebooks is not fully reproducible or reusable, leading to the question: **Can interaction provenance bridge the semantic and temporal gap between interactions and code in computational notebooks?**

1.3 Goals and Contributions

To address the research questions posed in previous sections, we have to achieve the following goals:

G1 **Capture and reproduce interaction provenance:** To reproduce interaction provenance for visual analysis, we will develop a provenance tracking approach that can capture the interactions in a visual analysis system. Our approach will support capturing nonlinear interaction provenance. The captured interactions will be replayable, enabling reproducibility of interaction provenance.

G2 **Make interaction provenance semantically meaningful:** To make the provenance semantically meaningful, we will capture the analyst's rationale for the selection interactions. Selection forms the basis for other interactions such as filtering, labeling, aggregation, etc., making it crucial to understand the rationale for the selection. We will capture the analysis rationale by capturing the *pattern-based intent* and domain knowledge of the analyst.

G3 **Curating and reusing interactions:** To reuse the interactions, we will develop techniques for effectively applying the interaction steps to an updated dataset by leveraging the semantically meaningful interaction provenance. Since interaction provenance can have multiple branches and grow in size rapidly, we will support curating smaller *workflows* for reuse.

G4 **Porting interactions to a different environment:** To use captured interactions in a different environment, we will adapt the interaction provenance to act as a shared abstraction between different analysis tools. The shared abstraction will sit between interactions and data operations and allow the conversion of interactions from visualization to data operations in code-based tools.

G5 **Bridging the gaps between code and interactions in computational notebooks:** We will develop techniques to bridge the two major gaps between code and interactions in computational notebooks: Results of the interactions are not accessible in code and interactions are not lost on cell re-execution. Our techniques will leverage the interaction provenance from the output to act as a shared representation between the code and the interactive output. We will save the interaction provenance directly in the notebook, enabling us to replay the interactions.

This dissertation proposes capturing the interaction provenance (i.e., sequence of interactions) for the interactive visual analysis session and leveraging the captured provenance to meet the above goals. Figure 1.1 gives an overview of our contributions and how they relate to the goals. The key contributions of the dissertation are:

1.3.1 A Software Library to Capture Interaction Provenance (G1)

In Chapter 3, we discuss a web-based software library [1] — Trrack — that is designed for easy integration into existing or future visualization systems. Trrack supports a wide range of use cases, from simple action recovery to capturing intent and reasoning, and can be used to share states with collaborators and store provenance on a server. Trrack also includes an optional provenance visualization component — TrrackVis — that supports the annotation of states and aggregation of events. It is designed to support various types of provenance, including interactions, insights, and rationale. Trrack addresses the need for an approach to capture and replay interactions as described in G1.

I developed the initial versions of Trrack and TrrackVis and later advised Zach Cutler, who was an undergraduate student at the time, with further development. Trrack and TrrackVis are available for installation on npm and have collectively been installed more than 12,000 times. Links to the source code, and documentation are available at https://vdl.sci.utah.edu/publications/2020_visshort_trrack/

1.3.2 Techniques to Capture Semantics Behind Interactions (G2)

Chapter 4 introduces methods to infer analyst intent behind selections in data visualizations [2] addressing G2. We describe intents based on patterns in the data and identify algorithms that can capture these patterns. Upon an interactive selection, we compare the selected items with the results of a large set of computed patterns and use various ranking approaches to identify the best pattern for an analyst's selection. We store annotations and the metadata to reconstruct a selection, such as the type of algorithm and its parameterization, in the provenance graph captured by Trrack. We present a prototype system that implements these methods for tabular data and scatterplots. Analysts can select a prediction to auto-complete partial selections and to seamlessly log their intents. We evaluate our approach in a crowdsourced study, where we show that auto-completing selection improves accuracy and that we can accurately capture pattern-based intent.

Links to the source code for the prototype and the supplementary materials, including scripts for generating the datasets for crowdsourced study and analysis of the study, are available at https://vdl.sci.utah.edu/publications/2021_ivi_intent/

1.3.3 Technique to Curate Reusable Workflows from Interaction Provenance (G3, G4)

Chapter 5 introduces methods to capture the interaction provenance in visualization systems for different interactions such as selections, filters, categorizing/grouping, labeling, and aggregation. These interactions can be curated or directly applied to updated datasets, making interactive visualization sessions reusable. We demonstrate our approach using an interactive visualization system that tracks interaction provenance, and allows generating workflows from the recorded actions. The system can then be used to compare different versions of datasets and apply workflows to them. We also introduce a Python library that can load workflows and apply them to updated datasets directly in a compu-

tational notebook, providing a seamless bridge between interactive visualization tools and computational environments. These techniques contribute towards goals G3 and G4.

Links to the source code for the prototype and the supplementary materials are available at https://vdl.sci.utah.edu/publications/2022_eurovis_reusing_workflows/

1.3.4 Techniques to Bridge Between Interactions and Code in a Computational Notebook (G5)

In this chapter, we introduce Persist, a family of techniques to capture and apply interaction provenance to enable the persistence of interactions in computational notebooks. When interactions manipulate data, we make the transformed data available in dataframes that can be accessed in downstream code cells. Our techniques address the semantic and temporal gaps in computational notebooks addressing G5. We implement our approach as a JupyterLab extension that supports tracking interactions in Vega-Altair plots and in a custom data-table component. Persist extension can re-execute the interaction provenance when a notebook or a cell is re-executed, enabling reproducibility and re-use. We evaluated Persist in a user study targeting data manipulations with 11 participants skilled in Python and Pandas, comparing it to traditional code-based approaches. Participants were consistently faster with Persist, were able to correctly complete more tasks, and expressed a strong preference for Persist.

The Persist JupyterLab extension is available for installation on the Python Package Index and can be installed using *pip*. Links to the source code, supplementary material including example notebooks and study analysis, and the published package are available at https://vdl.sci.utah.edu/publications/2024_preprint_persist/

1.4 Organization

In Chapter 2, we discuss the background and related work on interaction provenance, analyst's intent, analysis workflows, and computational notebooks. Chapters 3 to 6 discuss techniques that make up our contributions. In Chapter 7, we discuss the broader implications of our research and briefly probe possible future work. We conclude with Chapter 8 by summarizing our contributions.

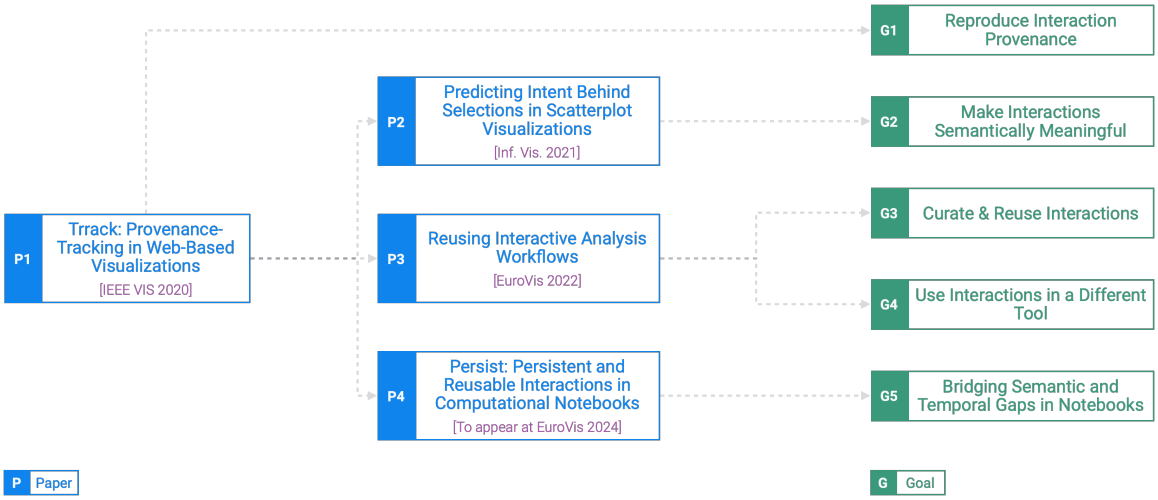

Figure 1.1: An overview of our contributions in the dissertation. Chapter 3 discusses our first contribution — Trrack — which is a provenance tracking library [1] that captures and replays the interactions (G1). Chapter 4 will discuss techniques to make the interaction provenance semantically meaningful [2] (G2). Chapter 5 will discuss techniques for curating and reusing the interactions on an updated dataset and use the interactions in a different analysis environment altogether [3] (G3, G4). Chapter 6 will discuss our contribution to bridging the gaps between code and interactions in computational notebooks [21] (G5).

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter delves into the foundational and related work underpinning the concepts and techniques discussed in this dissertation. We will discuss prior research on provenance in interactive visualizations, capturing analysis intent, data analysis workflows, and interactive visualizations in computational notebooks.

2.1 Interaction Provenance in Visual Analysis

A central aspect of the techniques discussed in this dissertation is tracking the interaction provenance in visual analysis. Interaction provenance refers to the interactions recorded during a visual analysis session. Understanding and capturing this provenance is essential for enhancing the reproducibility and reusability of the visual analysis. This section reviews interactions in visual analysis, focusing on selections, current methodologies, and technologies employed in capturing interaction provenance, underscoring the existing gaps that our research aims to bridge.

2.1.1 Interactions in Visual Analysis

Interactions are integral for the exploration and understanding of data in visual analysis. These interactions range from picking the datasets and selecting parts of the data to applying data transforms, zooming, and panning in the visualizations. Before discussing how to capture these interactions for provenance effectively, we must consider the various types of interactions commonly employed in visual analysis systems. From among the numerous interaction taxonomies (e.g., [22]–[25]), we adopt the taxonomy proposed by Heer and Shneiderman [26], which classifies interactions into three high-level categories of "interactive dynamics": 1) **data and view specification**, with the subcategories visualize (specify data and visual encoding), filter, sort, and derive; 2) **view manipulation**, with the subcategories select, navigate, coordinate (synchronize between multiple views), organize

(arrange windows and workspaces); and 3) **process and provenance** (record, annotate, share, guide).

A key interaction in visualization systems is selection [26], which is the foundation for numerous higher-level tasks in visual analysis. Brehmer and Munzner [27] classify selections as a manipulation method in the *how* part of their typology. Rind et al. [28] classify tasks along a cube using the dimensions *abstraction* (*concrete* to *abstract*), composition (*low-level* to *high-level*) and *perspective* (*how* and *why*). In their design space, a selection is an abstract, low-level task of the *how* perspective. Selections, including brushes, queries, and filters, define a subset of data items. These are typically communicated through visual alterations of the selected items. In systems with multiple coordinated views, linked brushing is commonly used to highlight the same items across different visualizations. Most selections are defined by explicit clicks on individual items, "paint-brushes" that select all elements under a brush tool, geometric brushes, such as rectangles or lassos, or textual queries. More advanced, data-driven brushes have also been proposed. For example, Fan and Hauser [29] introduce a method for fast brushing based on neural networks, where they estimate an intended selection based on simple sketches.

Data-aware selections are particularly interesting for this dissertation. These are actions defined in 'data space ' [26], [30], [31], meaning they are described by certain predicates rather than a list of items. This approach is exemplified by dynamic queries [32], which consider all items meeting specific conditions, such as those defined via sliders, as part of the query results. Certain types of brushes [33] can be realized in a data-aware way. For example, a rectangular brush in a scatterplot easily translates into the necessary predicates. Many selections (and other actions) are, however, realized by direct reference, e.g., by pointing at items, and hence, they are defined in *item space*. Selections in item space have several disadvantages: They cannot be generalized to apply to updating data or be used to explain a selection semantically. Data-aware selections, on the other hand, offer several advantages: They are robust to changes, can semantically explain a selection, and can be adapted in various ways to support an analyst, e.g., by relaxing a selection [34], or for reuse in a different context [35]. Most data-aware selections are realized by deriving rules directly from a rectangular brush. In general cases, rules for data-aware selections are more complicated to derive. However, deriving the pattern of a selection (what makes

the items in a selection belong to each other and different from everything else) is possible algorithmically. Xiao et al., for example, create "knowledge representations" of selections in communication networks [36]. This approach is similar in spirit to the work we discuss in Chapter 4. However, Xiao and coauthors' knowledge representations are limited to simple clauses and are not concerned with higher-level patterns in the data.

2.1.2 Provenance in Visual Analysis

Research on provenance capture and visualization has a long history. Provenance in data analysis refers to the history of an artifact, such as a dataset, a computational workflow, or an insight. Ragan et al. [37] discuss different purposes of provenance, including recall, replication (reproducibility), presentation, and collaboration (among others), but do not discuss reuse. Ragan et al. also characterize the different types and purposes of provenance. They distinguish the provenance of data, provenance visualization, the provenance of interaction, provenance insights (which capture analytical findings), and provenance rationales (which capture the reasoning behind any decisions made). Most provenance-tracking techniques are limited to the former three, whereas insight and rationale provenance can be achieved only using manual annotation.

Provenance tracking has two distinct approaches: 1) explicitly modeling a visualization workflow (*workflow-based*) [38] and 2) tracking the history of analysis to achieve provenance (*process-based*) [39].

2.1.2.1 Workflow-Based Provenance

Provenance based on explicitly designed workflow is typical in large-scale scientific data processing [40] systems such as SCIRun [41]. Workflow-based approaches are also common for specifying the visualization pipeline, for example, for volumetric data [42], networks [43], and tabular data [44]. A benefit of workflow-based systems is that they explicitly capture rules and thus can be reused easily. However, creating these workflows does not support freeform data exploration, and these rules do not typically capture higher level semantics.

2.1.2.2 Process-Based Provenance

An alternative to the workflow-based approach is to capture the provenance as the analyst interacts with an interactive visualization system [1], [39]. Many visualization systems support history tracking for action recovery (undo/redo), so we limit our discussion to systems that explicitly target provenance. Examples of the system that captures provenance directly from interactions include the graphical histories by Heer et al. [45] or CzSaw [46]. Various tools also represent histories as node-link diagrams [47]–[50], and some methods automatically detect key states in an analysis process [51] or retrieve prior states using search [52]. The provenance tracking in these systems is realized in an ad hoc way. However, recent papers have introduced software libraries for process-based provenance tracking. SIMProv [53] captures provenance for web-based visualizations. SIMProv uses a hybrid model that primarily stores actions, which can make switching between states slow. This dissertation presents the Trrack software library for tracking provenance in web-based applications. We will discuss the novel techniques implemented by Trrack [1] in Chapter 3.

2.2 Semantic Interaction Provenance

Understanding the 'why' behind user interactions in visual analysis is as crucial as tracking them. Interaction provenance often fails to capture the semantics or the intent behind the interactions themselves. We explore the concept of intent in data analysis, reviewing prior works that have attempted to capture and analyze these intents within a visual analysis framework.

Interaction provenance is often captured as a series of mouse and keyboard events and, hence, is not reusable, reproducible, or even human-readable. Interactions in a visual analysis system are guided by domain-specific analysis questions, e.g., identifying a gene that could be a drug target. The interactions themselves are based on specific patterns in the data, for example, finding a cluster or filtering out the outliers. The provenance log comprised of mouse and keyboard events does not fully reflect this pattern-based intent by the analyst. The focus of the provenance in such a case is on 'what' the analysts did rather than 'why' they did it. To meaningfully capture visual analysis sessions, we must infer the *intent* or the *why* behind the analyst's interactions during the analysis session.

This *intent* or the *why* is comprised of two parts: 1) the domain-agnostic structural patterns in the data and 2) the domain-specific knowledge that the analyst possesses.

2.2.1 Pattern-Based Intent

We can refer to the patterns in the data that analysts interact with as **pattern-based intents**. Pattern-based intent is related to insights in the data, as defined by Karer et al. [54]: "Insights affecting the viewer's knowledge about statistical and other structural information about the data." A difference in our definition of pattern-based intents is the viewpoint: Insight is about an analyst learning something, whereas intent is the reasoning behind an interaction.

In Chapter 4, we will discuss our contribution to automatically capturing intent behind selection interactions. We compare the analyst's selections with an output of different machine learning algorithms, each representing a pattern in data. We use the comparison to predict or infer the intent behind the selection. As discussed in the previous section, selection is one of the fundamental interactions found in visualization systems. However, selections also serve as the first step in more complicated actions, such as filtering, extracting, querying, aggregating, grouping, manipulating, or labeling items. Hence, knowing the intent behind a selection is also helpful in knowing the intent behind these derived operations.

Inferring an analyst's intent has been studied in various contexts. For example, Myers [55] proposes methods for inferring operations and source code from demonstrations when implementing graphical user interfaces. More specific to data analysis, Gotz and Zhou [25] study analysts' activities and model them in four tiers, from high-level *tasks*, to *subtasks*, to *actions*, to *events*. Actions, which correspond to our pattern-based intents, are composed of a type, an intent, and parameters. They represent an executable, semantic step, such as a query, that bridges the high-level human cognitive ability and the low-level user interactions. Gotz and Zhou implement this framework in the Harvest prototype that captures such actions. In contrast to our approach, however, Harvest captures *that* an action was executed, but not *why*. A related tool that also captures actions is SensePath [56]. A key difference to Gotz and Zhou's work is that SensePath is optimized to support qualitative data analysis: It is made for analysts to use the log of semantic actions in

qualitative coding. Dou et al. [57] argue that much of the reasoning process during a visual analysis session can be inferred by humans from inspecting user interactions. However, it is unclear whether a human's ability can be leveraged by automatic methods [39]. Brown et al. [58] have shown that user performance and certain personality traits can also be inferred from analyzing user interactions.

Another related thread of work is concerned with predicting future events in an analysis process to enable guidance [59]. For example, Ottley et al. [60] predict future clicks on items based on an interaction history. Steichen et al. [61], and Gingerich and Conati [62] show that predicting lower-level tasks, such as *retrieve value*, is possible using eye gaze data. This approach differs from our goal of inferring the intent of an interaction. Monadjemi et al. [63] propose a Bayesian approach to predict intents by ranking Gaussian distribution models based on user interactions. The ranked models can then predict the next interaction, detect exploration bias, and summarize the analysis process based on click patterns. Battle et al. [64] propose ForeCache, a tool for exploring large datasets. ForeCache uses Markov chains and computer vision algorithms to model analysts' future actions based on their past moves. The system uses these predictions to pre-fetch data. In contrast to our techniques, these related approaches target predicting future interactions, visualization recommendations, query generation, or preloading data subsets but fail to capture the 'why' behind an interaction. Our approach is also related to query-from-example methods developed in the databases community. For example, Dimitriadou et al. [65] infer range queries from a set of selected items.

A common goal for intent prediction is view specification, i.e., selecting data (sub)sets and suitable visual encodings. Systems such as Tableau's Show Me [66] use data properties to predict proper visual encodings. Natural language interfaces for view specification attempt to extract intents from language [67] and extract configurations for a view. Saket et al. [68] predict intents for view specification from demonstrations, such as assigning a color to a dot in a scatterplot, based on which their system infers the intent of mapping an additional variable in a dataset. Their follow-up work [69] demonstrates that analysts seamlessly switch between manual and mixed-initiative approaches. Demiralp et al. [70] compute patterns for a dataset and suggest visualizations for each pattern, but their approach is not reactive to analyst selections.

2.2.2 Domain Specific Knowledge

The analyst's domain knowledge is the other important part that informs the intent behind an interaction. A common approach to capture the domain knowledge is through **note taking and annotation**. Annotations are standard in visualizations designed for presentation but are not frequently integrated into exploratory visualization tools, with notable exceptions (e.g., [45], [50], [71]–[75]).

Manual notes, documentation, and annotations can capture analysts' reasoning and insights, but creating and maintaining them is associated with a burden on the analyst and, thus, a lack of scalability [25]. In computational notebooks, support for documentation and annotations is built with markdown and code comments. Rule et al. [20] have claimed that most Jupyter notebooks they analyzed either completely lacked or had scarce documentation of the analysis. The interviews with data analysts showed that interviewees found documenting the analysis for sharing and presentation tedious and lacked guidance. When such documentation is present, the documentation focuses on presenting the results and interpretation rather than an exposition of the analysis process. Online discussions regarding the use of literate programming [76] bring up the point that the story one tells at the start of the project is not the same as one told at the end. Documentation of the analysis exposition from the beginning will make it harder to change and adapt as the analysis evolves, whereas retroactive documentation risks missing out on how the assumptions and analysis approach changed over time.

In Chapter 4, we propose associating annotations with the corresponding provenance step and automatically capturing pattern-based intents. This approach allows analysts to elaborate on their intentions, bridging the gap between pattern-based and domain-specific, higher-level intents. We will further discuss the possible future work on capturing the domain knowledge in Chapter 7.

We can augment the provenance with semantic information by automatically capturing the 'pattern-based intent' and associating the analyst's domain knowledge. Semantic provenance can enable the goal of reproducibility and reusability of the analysis session by allowing the session to be verified by other analysts and be meaningfully applied to an updated version of the dataset.

2.3 Data Analysis Workflows

Reapplying the interactions or workflows curated from the interaction provenance to an updated dataset or in a different analysis environment is challenging. This section discusses prior work on data analysis workflows, compares visual analysis workflows with code-based workflows, and discusses existing research on creating, specifying, and reusing visual analysis workflows.

We define workflows in the context of data analysis as a sequence of steps executed to achieve some data transformation or analysis goal. We can distinguish between two types of workflows: 1) explicitly modeled workflows and 2) provenance-based workflows.

2.3.1 Explicitly Modeled Workflows

Explicit modeling of workflows is common in scientific data analysis [40]. Representative examples are systems such as Galaxy [77] for biomolecular data, SCIRun [41], and Kepler [78] for scientific/simulation data, and KNIME [79] in a machine learning context. Workflow approaches are also common for scientific visualization applications such as volume rendering. Here, VisTrails [42] is a prolific example. Notable workflow-based systems for abstract data visualization include GraphTrail [43], where each node in the workflow shows an aspect of a multivariate network, and VisFlow [44], which is tailored to tabular data. GEM-NI [80] is a system that presents a workflow-based approach for generative design. The GEM-NI approach demonstrates explicit workflows for the parallel exploration of alternative designs. Commercially available solutions like Tableau Prep support the creation of workflows for preparing and cleaning data. The interactions supported by Tableau Prep are combining data, filling in missing values, etc. However, Tableau Prep does not support creating a workflow interactively by manipulating the data directly or exporting the modeled workflow to another environment, including Tableau Desktop.

Explicitly modeled workflows are designed to be easily reused. At the same time, the definition of these workflows is similar to the explicit code-based specification of visualizations, and thus, the associated interaction cost [81] is high. The spontaneity and rapid exploration associated with interaction patterns such as direct manipulation [82] is lost. They are easier to learn than writing code but have a steeper learning curve than interactive systems.

2.3.2 Provenance-Based Workflows

An alternative approach to explicit workflow modeling is tracking user actions provenance [39], [83] and using this information to extract workflows later. Although several visualization systems track provenance [45], [47], [48], [84] and a few dedicated libraries to making tracking provenance easier to implement exist [1], [53], most tools do not explicitly curate workflows based on provenance. A notable exception is the Vistories tool [50], which enables analysts to curate interaction steps into data stories. However, these data stories cannot be reused on different datasets. Chen et al. proposed a parametric symbolic approach to support analytic provenance in their CZSaw system [85]. CZSaw enables analysts to reuse parts of the analysis process based on a previously created parametric model. The system does not support autodetection and application of patterns, and the analysis has to be done in the same system.

2.4 Analysis with Multiple Tools

In many scenarios, analysts frequently switch between interactive and computational tools [86]. Recent surveys and interviews with data practitioners [86]–[89] support the idea that data analysis rarely takes place within a single environment. Data scientists frequently switch between code-based environments (Jupyter, R, or MATLAB) and graphical analysis environments (Tableau or PowerBI) during the data analysis. 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 [89]. Data analysis workflows created in one tool cannot be directly or easily shared with another. Data stored in a widely supported format such as CSV or JSON are commonly used when switching between environments. Using only data instead of the actual analysis process frequently requires repeating specific analysis steps in the new environment, e.g., converting columns with only *ones* and *zeros* from integer to boolean. Further, the history and context of analysis steps from one tool are not transferred to the new tool, complicating the reproducibility of the entire analysis process. Therefore, despite the advantages of interactive visualization, analysts frequently stick to code environments to avoid the extra work involved with switching between tools [86]. Batch and Elmqvist discuss the need for interactive visual analysis systems to export the interactive action performed in them [88].

Alspaugh et al. [87] propose developing integrated tools that combine code environments' expressiveness with direct manipulation available in interactive visualizations.

In Chapter 5, we present a hybrid approach to capturing workflows. Analysis sessions are automatically captured in a provenance graph with support for branching the analysis processes. The analyst can later use the provenance graph to curate explicit workflows, which can be applied to an updated dataset. Our approach supports storing the analysis provenance and the curated workflows in an abstract representation, which can be reused in different environments by building tooling around it.

2.5 Bridging the Gaps Between Code and Interactions in Computational Notebooks

This section discusses relevant prior work to notebooks and interactive data analysis, followed by interactive visualization within notebooks.

2.5.1 Computational Notebooks

Computational notebooks are a popular tool for data exploration and analysis. They fulfill Knuth's [9] vision of literate programming by blending code, text, figures, and data to develop a narrative data analysis. The most popular examples are the Jupyter notebooks [90], Observable notebooks [13], and R Markdown [91]. Computational environments enable sophisticated documentation and narration, incorporating markdown text, figures, and other media. These computational environments inherently support capturing a reproducible and reusable analysis process. Analysts can use markdown features in notebooks to externalize their analysis rationale and add context to their methodologies. Structuring data analysis in parameterized functions allows for code reuse on updated or new datasets. The popularity of data analysis libraries such as Pandas [11], scikit-learn [92], [93], tidyr [94], and ggplot [95] further support the reusable nature of computational environments.

2.5.2 Code and Interactive Data Analysis

Kandel et al. [96] introduced an interactive system for data transformations called Wrangler. Wrangler allows direct manipulation of visualized data and offers an interactive history for review and refinement, but it can also export wrangling steps to code. Guo et

al. [97] 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 [98] described as a code-centric data cleaning tool. The extension allows interactive data cleaning and generates Python/Pandas code corresponding to the cleaning operations.

2.5.3 Interactive Analysis in Notebooks

Interactive data analysis is rare to see in notebooks. Schmidt and Ortner [99] cite limited interaction capabilities native to the environment as one of the reasons for the lack of interactive data analysis. However, support for interactive outputs in notebooks is becoming more common. Wrex [100] 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 [90] is the most popular computational notebook and supports various data analysis and visualization libraries. Jupyter Notebook can be used to create interactive outputs with libraries like JuptyerWidgets [18], Bokeh [17], and Streamlit [101].

However, the notebooks' interactive outputs and code-cells have significant gaps. First, native visualization libraries, such as Matplotlib [102] in Jupyter, have only basic interactive capabilities and cannot feed back actions from visualizations to code. Complex visualizations, such as custom tools, can be embedded in Jupyter notebooks but typically cannot manipulate data in the notebooks. Libraries such as Vega-Altair [16] support interactive visualizations, but the interactions primarily coordinate between multiple views and do not natively manipulate data. Second, the notebook interactions are transient and lost on re-executing the notebook or restarting the kernel. In contrast, the code-cells are persistent and can be replayed repeatedly. Some recent works try to address these gaps using different techniques, the most common of which is generating code from interactions.

Wu et al. [19] have introduced 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. The Mage API extension for notebooks [103] also aims to support

fluid movement between code and interactive outputs. Mage detects the interactions in the output, maps it 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 non linear analysis using code snippets can become complex.

In Chapter 6, we present Persist, a new principle for integrating interactions with code. Persist shares the vision of a seamless transition between code and interactions with B2 and Mage. However, despite the shared vision, Persist approaches the problem of bridging between code and interactions differently. Similar to B2, Persist employs the idea of using a shared abstraction to bridge between code and interactions. Persist leverages the rich interaction provenance that is captured from the interactive outputs as the shared abstraction. 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. Persist directly saves the interaction provenance to the notebook and replays it when required to address the issues with interactions being transient. 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.

CHAPTER 3

TRRACK: PROVENANCE TRACKING FOR WEB-BASED VISUALIZATIONS

This work is based on our previous work published as a short paper [1]. © 2020 IEEE. Reprinted, with permission from Z. Cutler, K. Gadhave, and A. Lex, "Trrack: A library for provenance-tracking in web-bsed visualizations," in *IEEE Vis. Conf.*, Oct. 25–30, pp. 116–120.

At least since Shneiderman argued that we cannot expect users to get it right immediately, and visualization systems need the ability to correct a sequence of actions and replay it, the value of undo/redo and replay [22] (or action recovery) has been undisputed. Even though professional desktop applications commonly support action recovery, many web applications, especially academic visualization prototypes, do not. For example, the authors of both Lyra [104] and Data Illustrator [105], two popular web-based data visualization authoring tools, mentioned the lack of undo in their original designs as a cause for concern for users, which resulted in less exploration. Yet, action recovery is only one of the purposes of collecting provenance data. Provenance data have many other uses, ranging from recalling an analysis process to reproducing it, collaborating, and logging for evaluation or meta-analysis [37].

Designers and developers of web-based visualization tools and prototypes frequently develop custom software to capture, store, and utilize provenance information. Doing so, however, can be tedious and is often not in immediate service of the goals of a visualization prototype. Home-grown solutions are also unlikely to leverage the full potential of provenance, such as history visualization or easy state sharing among remote collaborators.

To remedy this problem, we developed the Trrack library (the name derives from Reproducible TRACKing), our primary contribution, which provides provenance-tracking for the purpose of action recovery, reproducibility, collaboration, and logging. Trrack can

benefit existing and future systems widely and support data collection in quantitative and longitudinal evaluations. Trrack is designed to support a wide variety of types of provenance, including provenance of interactions, insights, and rational. Trrack is accompanied by TrrackVis, which provides optional provenance UI elements, such as simple undo/redo buttons or a sophisticated provenance visualization that can be customized and supports annotation and aggregation of states. Finally, Trrack enables sharing states through a URL and provenance data management on a server. Trrack comes with well-documented examples and uses tests to ensure code quality. Trrack is available under a permissive open-source license at https://github.com/visdesignlab/trrack. The npm packages for Trrack and TrrackVis have been collectively installed more than 12,000 times. The latest version of Trrack is used in multiple places, including industry applications. The prototypes we will discuss in the next chapters use Trrack for capturing and replaying the interaction provenance. Trrack is used to implement the latest version of popular set visualization — UpSet [106]. Trrack is instrumented for managing the provenance of two network visualizations and the study they were used in. [107]. reVISIT study, a framework for developing web-based user studies uses Trrack to enable provenance tracking of the user studies [108].

Our contribution aims to inform the visualization research community of a robust, well-tested, easy-to-integrate technology of broad interest. We believe Trrack is consistent with the renewed calls for applications and systems-oriented research that is necessary to tackle the increasing complexity of datasets and analysis challenges. Figure 3.1 summarizes the contribution of Trrack to this dissertation, which is to enable reproducing interaction provenance by capturing and replaying the interactions.

3.1 Design Goals

We designed the library based on our experience developing a provenance graph for a storytelling application [50]. The overarching design goals are 1) versatility of use — the library should support all different purposes of provenance-tracking; and 2) ease of use — developers should be able to track and visualize provenance with minimal effort. Here, we list our specific design goals.

1. **Allow Developer Agency:** Application developers should have the flexibility to

decide which actions to track. For example, what is sensible to track might be vastly different between a production visual analytics system and a prototype used in a user study.

2. **Support Action Recovery:** Undo/redo are important in all user interfaces, so mistakes can be quickly recovered. In addition to undo/redo, we want to make it possible to quickly browse to any prior state, so analysts might be willing to investigate paths they otherwise would not have if they can easily recover.
3. **Support Reproducibility:** The reproducibility of analysis processes is critical in data analysis. However, unlike analysis done in computational notebooks, interactive visualizations are difficult to make reproducible. We strive to capture not only interactions but also annotations so that user intent and reasoning can be captured as well.
4. **Support Collaboration:** Analysts rarely work in a vacuum: They need to either share and communicate their results or collaborate with other analysts on the same project. A provenance-tracking library needs to support both. To give developers flexibility, we envision two ways of collaborating: A lightweight approach of sharing the state of an application by copying the browser URL and a more sophisticated approach to sharing the full analysis provenance.
5. **Support Meta-Analysis:** We want to design our library to support collecting information about usage, either for analysis of long-term use in the field or for controlled studies. The requirements for action-recovery/reproducibility are different from those for meta-analysis; usually, the latter requires more fine-grained logs, which can be a hindrance for the former. We aim to design our system so that both can be supported simultaneously. Meta-analysis also requires that the provenance data be exported in a format that is amenable to further processing. Finally, unlike traditional logging, we want to be able to jump into any recorded session so that the context of, e.g., a performance problem, can be investigated.
6. **Support Annotation, Highlighting, Bookmarking:** As already alluded to when discussing reproducibility, users need to be able to bookmark states so that they can be quickly found and retrieved and annotate states so that users or systems can provide context, including insights or the rationale for a specific action. More generally, a

developer might want to store a variety of meta-information with individual states, which a provenance-tracking library should support.

7. **Provide Feature Rich Provenance Visualization:** In addition to provenance-tracking, we also intend to provide user interface elements that can be used to navigate provenance data if desired by the developer. A provenance visualization should be able to manage the whole feature set of the library, including branching states, annotations, and bookmarks. Also, as provenance data can quickly grow, it needs to be able to manage large interaction graphs.
8. **Efficient Storage and Retrieval in Large Provenance Graphs:** A long visual analysis session can lead to a very large provenance graph if every interaction is tracked or the tracking continues for long periods of time. A library, therefore, needs to be designed with efficient storage in mind. A provenance library also should support the quick recovery of a particular state.

3.2 Trrack Design

Here, we describe the design decisions that went into the development of the Trrack library, as motivated by our design goals. We also provide a brief architectural overview and describe how Trrack interacts with a visualization application (see Figure 3.2).

Trrack uses a provenance graph approach where each recorded action results in a new node in the graph. Nodes can be attached at any point in the graph, following the branching model of provenance. To utilize Trrack, developers must define a state that fully describes their application. Developers may then add observer functions to individual keys in the state. These observers will be triggered if that particular part of the state changes, either by adding a new state or changing the current node in Trrack, e.g., through undo. We recommend updating only the front-end application within these observers. The only other interaction developers must implement is creating and applying an action when a user interacts with the visualization (see Figure 3.2). This action is what will then create a new node with a modified state in the graph. We provide a web app with examples of different complexity to illustrate this approach at https://vdl.sci.utah.edu/trrack-examples/.

3.2.1 Trrack Architecture

The common storage types for action recovery systems are state-based and action-based [45]. **State-based** systems store the user-defined state of the application at every node, allowing for instant jumps to any node in the history. The downside of this approach is the inefficient use of memory or disk space. **Action-based** systems store the action required to get from one node to the next, which can lead to slow performance when jumping between states since actions must be applied sequentially to maintain proper recovery. The advantage of this approach is that it minimizes the storage/memory overhead. Heer et al. [45] also discuss **hybrid approaches**, where action-based systems periodically store a state to increase performance when loading a previous state.

In Trrack, we use a different approach — **differential states**. To ensure quick loading of arbitrary nodes of the provenance graph, we store states and require developers to define a state for use in Trrack. However, we do not store the state at every node to address the size problems common with such an approach. Instead, we store a difference between the current node's state and the last node that stored the entire state. We track how many keys in a state object have changed, and if a heuristic threshold is surpassed, we store a full state. This approach ensures that the stored differences do not grow larger than the original state and minimizes the size of differences of ensuing nodes. Developers may also specify that a particular action should always store the entire state if desired. This mechanism is completely abstracted from developers using the library. A developer only has to request a particular state, e.g., through the undo function, to receive a corresponding state, as illustrated in Figure 3.2. In addition to the state information, we also store meta-information about the actions, which is useful for visualization and to maintain a user's mental map.

3.2.2 Ease of Integration

Trrack is a JavaScript/TypeScript library developed with the goal of making the integration of the library as seamless as possible. There are two ways to use Trrack: It can take over the state management in the application, which is most useful when designing a new web-based visualization. Alternatively, Trrack can interact with any existing state management solution to track changes to the state without affecting the UI. Trrack is

designed to be framework agnostic. It can work with vanilla JavaScript, UI frameworks like React, and state management libraries like Redux.

3.2.3 Sharing State

Visualization applications do not commonly provide the ability to share the state of an application. Trrack allows for easy sharing by sending the current URL to a collaborator. The current state of an application is encoded and added to the URL as a URL parameter. Whenever the state of an application changes, the URL is updated. When a page with a matching URL parameter is loaded, the Trrack library parses it and returns the desired state to the application.

3.2.4 Persistence

By default, the provenance graph maintained by Trrack is stored only in memory. We must make the provenance graph persistent to ensure reproducibility and collaboration beyond just sharing states. Trrack has functionality for exporting provenance graphs so that they can be managed how a developer desires. We also provide a default implementation based on Google Firebase. Users can connect a Firebase database to Trrack during setup and automatically store every node in the graph in Firebase every time a change is made.

3.2.5 Reproducibility and Capturing Intent

An important aspect of reproducibility is understanding the reasoning behind steps taken by the user. For this purpose, each node in the provenance graph stores multiple types of metadata. Every node has an annotation property and a "type," a user-defined string meant to identify the purpose of the state change at that node. Additionally, a generic object may be defined by the developer and stored with the node. This metadata can be used to capture and later interpret actions taken at each node, which can range from simple bookmarks to written annotations to complex cases involving algorithmically predicted intents [2].

3.2.6 Logging for User Studies

Due to Trrack's ability to reproduce entire sessions and store generic metadata, we believe it is especially useful for user studies. Typically, analysis of individual actions in a user study would require manual annotation of every user, a time-intensive process. By storing labels, event types, time stamps, and generic metadata on every action, most meta-analyses of a study using Trrack can be captured automatically. This functionality can save time, reduce human error, and allow for more diverse analysis. Additionally, our differential states architecture allows for convenient analysis of stored graphs since all of the relevant information is contained in the provenance data, independent of the tracked application. Trrack provides dedicated export functionality tailored to post hoc data analysis needs. Specifically, it can export details about each action that are not stored directly on the graph and are not required for action recovery. Finally, Trrack makes it straightforward to jump into a specific analysis session of a user, allowing analysts to understand their context.

3.2.7 Logging versus Action Recovery

Some actions developers want to track (e.g., for the purpose of logging) may be too frequent or inconsequential to include in the undo/redo chain. An example is a hover action that highlights an item when the mouse rests on it. Users would not expect to undo/redo a hover, but seeing when a hover was used can be important when analyzing logs. For these cases, Trrack allows developers to label actions as ephemeral. By default, the undo/redo functions in the library will skip over ephemeral nodes, and ephemeral nodes are also treated specially in the provenance visualization.

3.3 TrrackVis

TrrackVis is a separate library complementing Trrack to visualize the provenance graph, as shown in Figure 3.2. The purpose of this library is to allow for easy navigation in the provenance graph and to provide a way to create and view annotations and metadata. The graph is shown as a node-link tree. Clicking on any node will change the application's state to the one associated with that node. TrrackVis is designed to be highly customizable, allowing the user to choose to integrate features, such as tooltips or annotations. Custom

icons can be added to nodes to match user-defined event types. To address the scalability of the visualization, consecutive nodes can be defined as a group, which can collapse the constituting elements. For example, groups can be used to identify specific sections of the analysis process as related and organize them as such for additional annotation. The "Insight" nodes in Figure 3.3 contain nested actions that are associated with a particular insight captured in this application. By default, we also use groups to collapse nodes that are labeled as ephemeral.

3.4 Implementation, Testing, and Documentation

We developed Trrack over the course of 18 months, constantly refining and adapting the library to a variety of changing needs. The library has roots in an integrated application we designed for visual storytelling [50] and in a prototype library based on the concepts from that paper [109].

To encourage the adoption of the library, we created a series of basic examples that demonstrate the core features of the library. The repository contains additional documentation and other examples. Trrack also includes a comprehensive suite of unit tests to ensure previous versions of the library do not break with future additions.

3.5 Usage Examples

We have used the Trrack library in multiple projects, spanning prototypes developed for a technical visualization paper, applications used by medical professionals, and user studies. Figure 3.3 shows some of the examples. Here we provide details on two of them.

We used Trrack to capture data on participants in a crowdsourced study [107] for evaluating multivariate network visualization techniques. The study used two levels of provenance: A study-level provenance to track the progression of the study and a task-level provenance to track the interactions in a particular task. A combination of these two provenance graphs allowed us to collect data about interaction patterns in both conditions while participants completed the tasks. To explore the logged data, we developed a custom visualization, which also allowed us to jump into individual analysis sessions. The study stimulus is available at https://vdl.sci.utah.edu/mvnv-study/.

We also leveraged Trrack in our prototype system designed for predicting analysis

intents when brushing in scatterplots [2] to capture user interactions such as selections, brushes, and adding/removing plots. The system uses this provenance trail to calculate a set of predictions for possible intents (clusters, outliers, etc.) and ranks them. We used Trrack's ability to store metadata for capturing these predictions along with relevant action nodes. Users can select one of the predictions to mark it as their intent and provide annotations stored in the provenance graph. Additionally, we captured data when we ran a crowdsourced evaluation of this project with Trrack. The evaluation involved 128 participants and used the Firebase integration. The screenshot for TrrackVis shown in Figure 3.3 is taken from this project. A demo is available at http://vdl.sci.utah.edu/predicting-intent/.

3.6 Discussion and Limitations

The library most related to Track is SIMProv [53]; we discuss the main differences next, followed by a brief discussion of limitations.

3.6.1 Performance

Compared to SIMProv, Trrack's differential states storage model provides benefits over SIMProv's hybrid model. An action-based model can be slow when jumping between nodes that are far from each other, and due to the use of differential states, the storage requirements of Trrack are mitigated.

3.6.2 Easy-to-Use Exports

For user studies and log-file analysis, easily analyzing exported provenance data is critical. Because SIMProv uses an action-based model, the exported data refers to context-specific information, such as function names, which do not exist in the export. This context-specificity makes it difficult to analyze the data outside the original application. In contrast, Trrack's exported data are not application-specific and are human-readable. Every node has a state that can be used for analysis without knowing anything about the architecture of the application.

3.6.3 Ease of Integration

Trrack abstracts away as much of the storage model as possible. Developers simply register observer events on each property of the state. These observers then handle forward and backward navigation. In contrast, to ensure efficient navigation, SIMProv requires users to define checkpoint rules, forward changes, inverse changes, state changes, and inverse state changes, thereby increasing the complexity of the application.

3.6.4 Data Provenance

Although Trrack is well suited to track interaction data, it is not designed to store the provenance of datasets. For example, interactively running a normalizing procedure would create a new dataset. This dataset could either be stored directly, as an attribute of a node, or written to a separate file, and that file could be referenced as the output. Both solutions have disadvantages: The former mixes actions and data and creates a large provenance graph; the latter makes the association between data and an application state brittle. We plan on investigating the integration between data and action provenance in the future.

3.6.5 URL-Based Sharing

Sharing state through URLs may become problematic if the state is large and exceeds URL size limits. However, we have successfully tested this approach with over 150 keys describing a state. We believe most applications will be able to store a state much smaller than this as long as data are stored separately.

3.7 Future Work

In the future, we plan to strengthen the collaborative aspects of our method. We currently enable asynchronous collaboration but do not track contributions by separate users. We also would like to allow synchronous collaboration so multiple users may view and interact with the same visualization session, similar to the functionality available in Google Docs. As an immediate next step, we want to add story-editing and story-viewing capabilities to our library, similar to the approach demonstrated by Gratzl et al. [50].

With our hybrid strategy for storing states and diffs, the primary question to answer is how frequently states should be stored. We hope to do more investigation into ideal times

to store states. We also hope to add functionality that optimizes the graph's size when exported by iterating over the graph and re-storing nodes as appropriate.

3.8 Conclusion

In this chapter, we introduced Trrack, a library to track provenance in web-based visualizations. Our goal is to provide an option to easily track provenance in existing or future visualization systems. The companion library TrrackVis allows for visualization of and interaction with the provenance graph. Tracking provenance with Trrack allows action recovery, collaboration, reproducibility, annotation of analysis steps, and post hoc meta-analysis of the interaction sequences; and provides persistent storage with Firebase or other custom storage solutions.

Trrack and TrrackVis are open-source and published in the npm registry. We provide extensive usage examples and documentation and hope our library will contribute to increased provenance-tracking in more visualization tools. Also, although Trrack was designed primarily with visualization tools in mind, it can also be used with general-purpose web applications.

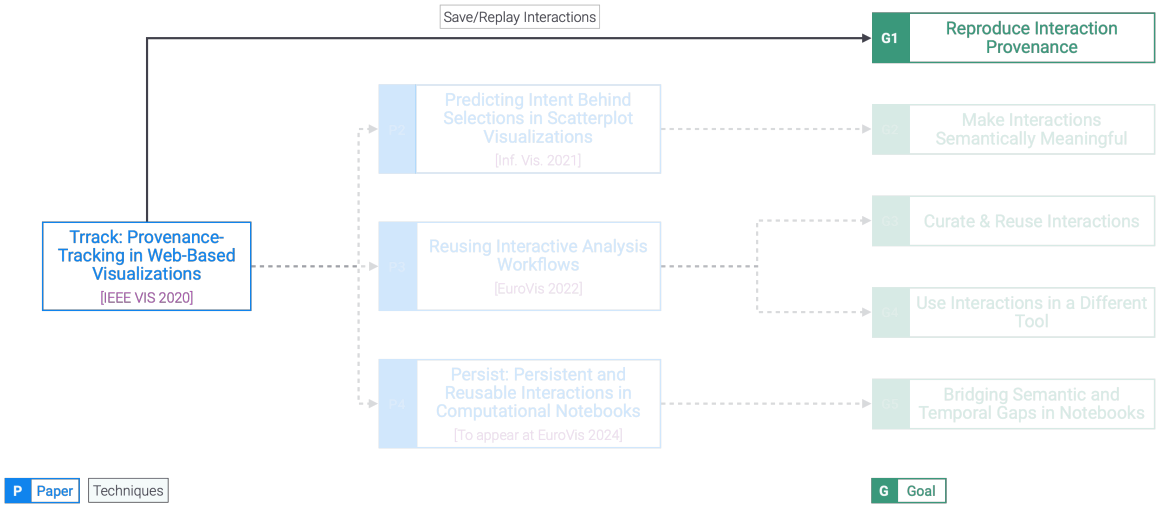

Figure 3.1: Contribution of Trrack to reproduce the interactions. Provenance tracking using Trrack can be instrumented to capture the interactions in the web-based visualization. Trrack can replay the interaction and reproduce it.

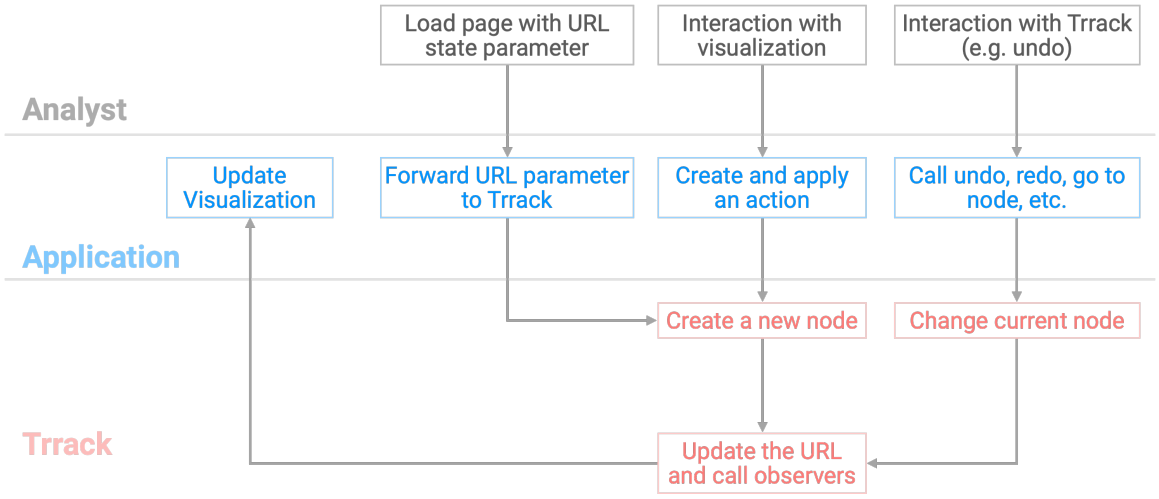

Figure 3.2: The relationships between the analyst, the application, and the Trrack library for storing provenance.

Figure 3.3: Four example applications using Trrack, our provenance-tracking library, and TrrackVis, the associated provenance visualization library, for different purposes, ranging from action recovery to logging for user studies. TrrackVis, shown on the right, utilizes custom icons, annotations, and grouping of nodes.

CHAPTER 4

PREDICTING INTENT BEHIND SELECTIONS IN SCATTERPLOT VISUALIZATIONS

In this chapter, we introduce techniques to achieve goal G2 — making interaction provenance semantically meaningful. Towards this end, we will introduce techniques to semiautomatically capture the analyst's pattern-based intent. Figure 4.1 shows the contribution of this chapter toward this dissertation. The chapter is based on our previously published work [2]. K. Gadhave *et al.*, "Predicting intent behind selections in scatterplot visualizations," *Inf. Vis.*, vol. 20, no. 4, Aug. 2021, pp. 207–228.

4.1 Motivation and Overview

When experts interact with a visual analysis system, they are frequently guided by a domain-specific analysis question, such as identifying a gene that could be a drug target. To answer this question, they execute a series of intermediate tasks, such as selecting a set of correlated items for detailed analysis. In contrast to the high-level goal of answering a domain-specific question, these intermediate tasks are based on patterns in the data: For example, selecting outliers, clusters, or correlations. Such a carefully constructed selection of items based on a domain-agnostic structure reflects a reasoning process — an **intent** — by the analyst. We refer to the motivation behind these actions as the *pattern-based intent* of an analyst. Pattern-based intents are distinct from higher level intents in that they are free of context and based solely on the data. They are also distinct from low-level intents, such as hovering over an item to read its label. In this chapter, we introduce methods to infer these pattern-based intents for brushes in scatterplots. We define *pattern-based intents* as the reasoning behind selections based on statistical patterns or structures in a dataset. These selections can then serve as the basis for more sophisticated actions, such as filtering, querying, aggregating, or labeling so that semantic knowledge about the purpose of the selection can be applied to these actions.

Why is capturing pattern-based intents important?

First, inferring intents based on partial selections can be used to **autocomplete** selections. To select outliers, for example, analysts would have to brush only a few examples and could then autocomplete the selection instead of painstakingly brushing the examples individually. autocomplete can also be used to correct a selection. For example, if an analyst intended to select a cluster, reviewing the predicted cluster might reveal points that should be added to the selection.

Second, making pattern-based intents available in provenance data improves the **recall and reproducibility** of analytic processes conducted with visualization tools. By capturing such processes at a higher level of abstraction than just low-level interactions, they become more transparent when revisited either by the original analyst or a collaborator. Hence, analysis sessions that capture intents are more justifiable and likely to increase trust in the process.

Down the line, such rich provenance data also have the potential to enable **reusing visual analysis sessions** on modified or updated data. For example, when an analyst first removes outliers before proceeding with an analysis, that action could be translated into a rule, which then could be used to automatically remove outliers from an updated dataset.

We use scatterplots and tabular data as common and important representatives of visualization techniques and data types to demonstrate the feasibility of predicting intents from selections. To identify the types of patterns that map to these pattern-based intents, we conducted formative interviews with scientists who regularly use scatterplots in their research.

Our primary contribution is **a set of methods to detect and capture these pattern-based intents** for brushes and selections. We select data mining algorithms that are suitable to detect patterns in a dataset and compute a large set of potential patterns for a dataset. We introduce methods to address the potentially large space of dimensions and parameters. Finally, we develop three approaches to score and rank the output of the algorithms relative to an analyst's selections.

Our secondary contribution is **an implementation of these methods in an interactive visualization technique**, thereby demonstrating how they can be leveraged for autocomplete and provenance tracking. By showing ranked predictions of patterns for a selection,

we create a mixed-initiative approach that lets analysts easily capture their pattern-based intent by verifying a prediction. We provide the means to annotate these intents to tie them to higher level domain goals and capture this information in a provenance graph.

We demonstrate the usefulness of our approach in a set of examples. We also show that we can successfully predict pattern-based intents in a large, crowdsourced quantitative study.

4.2 Patterns for Selections

When analyzing data, analysts have intentions at different levels of abstraction. We are specifically interested in the pattern-based intents behind brushes or selections of data items in scatterplots, which are still semantically rich but domain agnostic [110]. To define a set of patterns that map to these intents, we first developed an initial classification based on the literature [111], [112] and our own experiences working with scatterplots. We then validated and extended the initial classification through interviews with six scientists at the University of Utah who regularly use scatterplots in their data analysis. We used a convenience sample of domain experts we had interacted with professionally. Our inclusion criteria were: 1) regular use of scatterplots, and 2) and a willingness to share scatterplots or data used in scatterplots. The interview protocol was reviewed by the Institutional Review Board and classified as exempt from full review. We identified six participants from nursing, astrophysics, chemical engineering, psychiatry and population health, and surgery. The participants included one graduate student, one research scientist, and four faculty members. We provided participants with paper printouts of scatterplots of their own data and asked them to describe and highlight the kinds of patterns they found interesting. The goal of the interviews was to validate our initial classification of patterns based on the literature and to identify patterns we might have missed. The interviews were video recorded and then transcribed. The transcriptions were coded by two independent coders using a seeded codebook developed from the initial classification of patterns: Outliers, clusters, categories, multivariate optimization, and range queries. A table in the supplementary material shows the code frequencies from both coders for each interview. Both coders identified many instances of outliers, clusters, categories, and range queries. Only one of the two coders identified two cases of multivariate optimization. Both

coders frequently identified correlation analysis, which we originally had not included in our set of patterns. Based on this process, we identified the following data patterns that match the analyst's intents when analyzing data in scatterplots.

4.2.1 Correlations

Correlations are associations between two or multiple dimensions. Five of our six interviews with domain experts mentioned them as a target pattern. Frequently, analysts were looking to identify correlations in the overall datasets or parts of the data but also attempted to find points that did not fit the correlations. They had the intent to **identify subsets of data that correlate**, but also **identify items that do not fit the correlation**. In several interviews, these points were identified as "bad data". We found that participants did an approximate visual regression analysis, identifying linear and nonlinear trends.

4.2.2 Outliers and Inliers

Outliers are data points that differ significantly from other items. They were brought up as a pattern of interest in all six interviews. Frequently, analysts wanted to understand what causes the data points to be outliers, relying on their background knowledge. Outliers are also related to, but distinct from, the points that do not fit a correlation: For example, an item can be an outlier in its magnitude but perfectly fit the correlation. Outliers were also mentioned as bad data that should be filtered out. We consider both outliers and "inliers," i.e., the set of points that are not outliers, as target patterns.

4.2.3 Clusters and Groups

Clusters or groups of data points are items that are similar to each other but distinct from the rest of the dataset. They were mentioned as a pattern analysts look for in three of six interviews. Clusters were frequently not well defined in the data the experts we interviewed analyzed.

4.2.4 Multivariate Optimization

One goal when analyzing data is to find data points that are dominant over multiple dimensions. A typical example is to find a hotel that is both close to the city center and affordable. The set of such points is often called a skyline [113]. Hotels in the skyline are

such that no other hotel is cheaper and closer to the center. Skylines were brought up in two of our six interviews and hence are the least frequently mentioned pattern.

4.2.5 Categories

An observed pattern can sometimes be traced back to the items being of distinct categories. Four of our six expert participants mentioned they intend to select elements by category. For example, one expert wanted to separate data points based on categories, where one category corresponded to an experimental condition, and the other category was made up of unmanipulated controls.

4.2.6 Ranges

Four of the six experts mentioned they select data based on numerical ranges. Several experts stated these ranges can be based on domain conventions for setting cut-offs. We observed range selections based on single or multiple dimensions, implying that ranges can be combined for more complicated queries.

4.2.7 Discussion

We believe that the described patterns cover a broad range of use cases, but we do not argue that our list of patterns is exhaustive. For example, domain-specific patterns might be meaningful in certain contexts. Sarikaya et al. [112] describe tasks for analyzing scatterplots. Each of our patterns can be mapped to one or multiple tasks from their work. For example, they mention tasks like *identify anomalies*, *identify correlation*, and *search for known motif*, which can be mapped to the *outlier*, *correlations* and *cluster* pattern, respectively. Their list of tasks, however, goes beyond patterns, including, for example, *explore data*, and they do not explicitly mention some of our patterns, such as multivariate optimization.

Our pattern classification is limited to tabular data in scatterplots. We expect other patterns, such as rankings, would be common in different representations. Finally, we have sometimes included a pattern and its antipattern, such as outliers and nonoutliers, as separate patterns, but we have not done so consistently for all patterns. We have included anti-patterns for those cases where they were explicitly mentioned in our interviews (outliers and correlation). However, anti-patterns could also be considered for other cases.

4.3 Mapping Patterns to Intents

Most patterns that we identified in our formative study are also commonly targeted in data mining, which implies that various algorithms can be used to identify them. We leverage this diversity to calculate a broad set of patterns using different algorithms, combinations of dimensions, and parameters. We then compare the computed patterns with analysts' selections and rank them according to that match. Whereas our initial step creates a large set of patterns, the subsequent ranking makes these patterns manageable. We explain the details of the algorithms used and our ranking approaches in this section. Figure 4.2 gives an overview of our method.

Up to this point, we have implicitly assumed that the patterns we discussed appear in two-dimensional space. In practice, however, many datasets have much higher dimensionality. Hence, a key question we have to answer is: *For which dimensions should we calculate predictions?* We considered calculating patterns for all pairs of dimensions, all dimensions that are actively brushed in the system, all dimensions that are visible in the system, all dimensions in the dataset, and any combination of these options. Calculating all possible options is computationally expensive, if not prohibitive, but also not necessary. As we aim to predict the intent of analysts interacting with (possibly multiple) 2D scatterplots and not to reveal high-dimensional patterns, we decided to limit predictions to: 1) pairwise dimensions; and 2) the dimensions that are actively brushed. We believe that predicting patterns on pairs of dimensions is the most appropriate choice for 2D scatterplots, as these patterns match what is visible in the plot. This restriction to pairs of dimensions is also supported by the fact that we did not encounter examples where experts wanted to select items based on more than two dimensions. However, we also do not want to exclude the possibility of analysts selecting higher dimensional patterns. Hence, we also calculate all patterns for all dimensions that are actively brushed, as the brushes indicate that an analyst is explicitly interested in a combination of these dimensions. Consequently, in a set of two 2D scatterplots visualizing dimensions A/B and C/D, and with active selections in both scatterplots, we calculate and predict patterns in two dimensions for A-B and C-D and patterns in 4D space for A-B-C-D.

For example, if an analyst would like to select the species in Fisher's prominent Iris flower dataset, a selection based on 2D combinations of dimensions would be difficult

as the features are not well separated in any combination of 2D plots. In our system, they could start by plotting sepal width and sepal length and brushing a group of similar plants. They can further narrow the selection down by brushing in a second plot showing petal length and petal width. This combination of selections triggers a prediction considering all four dimensions. They can then select the cluster prediction that best matches their intended selection, leveraging patterns computed on higher dimensional space.

4.3.1 Algorithms

Many algorithms can extract the patterns we describe. In our system, we deliberately rely on standard algorithms that are robust and simple, although more sophisticated versions might exist. One reason for this is generality: Many data mining algorithms require careful choices of hyperparameters, but choosing good parameters requires expertise and trial and error, which is not acceptable for our use case. Instead, we choose parameters for these simple algorithms by sampling the parameter space or relying on defaults. For example, we run k-means with a k of 2-7 but use defaults for all other parameters. We do not use evaluation approaches for the quality of the outputs; instead, we let our ranking approach reveal the most suitable results. We also assume that the visualization uses linear scales. However, an extension to logarithmic or power scales would be straightforward. We use algorithms provided by *Scikit-learn* [92] unless noted otherwise and normalize the data before the analysis.

We use two algorithms for **clusters**, DBSCAN and k-means, that have complementary strengths since different algorithms pick up different kinds of clusters, such as circular clusters or concave-shaped clusters. DBSCAN is based on a (parameterized) measure of density (clusters are clouds of dense points of arbitrary shape), whereas k-means assumes roughly spherical clusters and requires the cluster number as a parameter. If no clusters are present, DBSCAN considers the whole dataset as one cluster (except for outliers), whereas k-means always provides a segmentation of the dataset. We solve each formulation multiple times with different parameterizations.

For **outliers**, we use two algorithms: The local outlier factory and the outliers identified by DBSCAN. We treat inliers provided by the local outlier factory as a separate prediction named *nonoutliers*.

Multivariate optimization is used to find values that are optimal across multiple dimensions. Although a general optimization would require weighting the value of each dimension, **skylines** [113] are a generic approach that determines the items that are not dominated by other points. As a skyline requires a definition of what is considered "good" in each dimension (e.g., a *low* price, but a *high* customer rating is considered good for a hotel), we compute skylines for all high/low permutations of the 2D cases. We limit predictions to all-low or all-high for higher dimensional cases because calculating all possible permutations would be computationally expensive.

The patterns that we have described so far use the output of an algorithm as a reference against which we can compare an analyst's selection. The **range-based query** pattern differs from the other patterns in that we do not have such a reference because the values used in range selections are typically external to a dataset. Our interviews have shown that ranges can be the result of domain-specific knowledge or can be used to select all the high or low values. We also found that analysts create complex queries by combining multiple simple brushes and selections. The traditional approach to storing range-based queries is to store the extent of brushes. However, this method is not general: It does not work for other selection types, like point or paintbrush selections, and it is defined only on spatial representations. To address this problem, we introduce a method that is based on learning a decision tree from the input. We formulate the problem as a binary classification problem, where the decision tree is used to separate the selected from the not-selected points. As the decision tree uses information-theoretic measures, it learns a compressed representation of the brushes made by the analyst. For example, the red, rectangular brush in Figure 4.3(a) could be stored with four coordinates identifying each side of the rectangle. A range query based on a decision tree, shown in Figure 4.3(b), stores a generalized and simplified version with only two rules. We can also generalize a selection, similar to query relaxation [34], based on this idea. By pruning the decision tree by one level, we extract the most important components of the selections, as illustrated in Figure 4.3(c), which can be useful to correct imprecise selections.

If a dataset contains categorical values, we treat each **category** as a separate pattern. Individual categories could conceivably be shown in the scatterplot, but predicting an overlap between a selection and a category is especially important if a dataset has many

categories that cannot be shown at the same time.

Finally, we use regression as a framework to analyze **correlations** in the data. To identify the sets of points that are part of a linear or quadratic correlation pattern, we run the following algorithm on linear and quadratic regression datasets, where X is the entire dataset, I are points marked as inliers, R is the regression model built with Scikit-learn, r_i is the residual for a point x_i calculated from R, and *iters* is a constant for the maximum number of iterations we execute if the algorithm does not converge earlier.

1. First we assume all the points in the dataset X are inliers I and build a Scikit-learn regression model R on the I.
2. Then we calculate residuals r_i using R for all points x_i in I.
3. Next we define $\bar{r} = median(r_i | x_i \in I)$.
4. Then we redefine I as all the points $x_i | x_i \in X$ where $r_i < 2\bar{r}$.
5. We repeat points 1 to 4 for a predefined number of iterations *iter*, stopping early if inliers do not change between iterations.

The pseudocode for the above algorithm is expressed in Algorithm 1.

4.3.2 Ranking Predictions

All the described patterns result in classifications for each item in the dataset. To rank the predictions in our system, we compare these patterns with a binary classification representing an analyst's selection. Figure 4.4 shows an overview of our method. Some algorithms, like clustering, produce a multiclass prediction, which we first transform into a set of binary classifications using one-hot encoding. We can then use a similarity metric to rank each of the predictions. We use a preprocessing step to remove duplicate predictions for the same pattern from the set of predictions to rank. Duplicate predictions occur frequently if a pattern is robust to different parameterizations of the same algorithm.

Algorithm 1 Calculate inliers for a correlation pattern.

$I \leftarrow X$ ▷ initially mark all points as inliers
while $iters > 0$ and I is changed **do**
  $R \leftarrow Regression(I)$ ▷ building regression model
  $\bar{r} \leftarrow median(r_i | x_i \in I)$ ▷ median residual over I
  $I \leftarrow x_i | x_i \in X \wedge r_i < 2\bar{r}$ ▷ update inliers
  $iters \leftarrow iters - 1$
end while

In the following subsection, we discuss three ways to rank the predicted patterns that are optimized either to infer intent for an existing selection or to predict intent of a partial selection, plus a special case for ranking range queries.

4.3.2.1 Ranking for Inferring Intent

Our baseline metric is the Jaccard index, which is a measure of similarities between sets. We consider the set of selected items S and the set of items in a candidate pattern C. The Jaccard index $J(S, C)$ between those two sets is then defined as

$$J(S, C) = \frac{|S \cap C|}{|S \cup C|} = \frac{|S \cap C|}{|S| + |C| - |S \cap C|}.$$

Here, a value of 1 corresponds to a perfect match, and a value of 0 indicates no overlap. The Jaccard index is well suited to infer the intent of an existing, complete selection.

4.3.2.2 Ranking for Autocomplete

The tasks of autocompleting and inferring intent differ with respect to ranking a possible pattern: In the case of inferring intent for a completed selection, finding the best match overall is necessary. In contrast, for autocompletion, the selection is partial, as the goal of the task is to complete the selection. Hence, we needed to develop a ranking approach that does not penalize incomplete selections. To do this, we rank the predictions using a modified Jaccard index J_m. We define the similarity between sets S and C as

$$J_m(S, C) = \frac{|S \cap C|}{|S \cap C| + |C \setminus S| + w \times |S \setminus C| + r \times |X|}.$$

Here, X is the complete dataset. The modified similarity metric reduces the penalty for points in S that are not present in C by down-weighting $|S \setminus C|$ using a factor $w < 1$, reflecting the goal of a partial selection to be automatically completed. The metric also adds a regularization parameter of r to prevent boosting ranks in cases where few correct points are selected. Empirically, we found that $w = 0.2$ and selecting r such that $r \times |X| = 3$ gives good results for datasets that are suitable to be visualized in scatterplots. Due to the regularization, the metric never reaches 1, but 0 still indicates no overlap.

4.3.2.3 Ranking Ranges

Our range-based queries rely on a decision tree of arbitrary depth; hence, the pattern captured by that decision tree is always a perfect match to the selection. Consequently,

the range query would always rank at the top if we ranked it using the Jaccard index. However, this ranking is inconsistent with what humans perceive as a good prediction of their intent: When analysts create complex selections, they tend not to think of them as long lists of rules. Instead, they likely select a pattern based on a higher level relationship in the data. To address this inconsistency, we assign a score R to the range-based query using a heuristic based on the depth d of the decision tree: $R = \frac{1}{d^2}$. Our heuristic relies on the assumption that simpler queries are more likely to match an analyst's intent than complex queries that require deep decision trees to represent them. The resulting score is on the same scale as the Jaccard index and, hence, can be easily integrated.

4.3.2.4 Probabilistic Ranking

The Jaccard index considers each possible pattern independently. However, an analyst's intent is rarely independent, and some predicted patterns are more likely than others. To address this, we propose a probabilistic framework that models these effects. We denote predicted patterns with $C_i \in \mathbf{C}$ and the Boolean vector representing the analysts' selection as S. Finding a probabilistic ranking of the predicted patterns is the same as determining the conditional probability $P(C|S)$ for each pattern. Framing the problem using probabilities also gives us more interpretability as it relates the different intents to one another: The probabilities for each intent add up to one:

$$\sum_{C_i \in \mathbf{C}} P(C_i|S) = P(C|S)$$

$$\sum_{C_i \in \mathbf{C}} P(C_i|S) = P(C_{\text{Outliers}}|S) + P(C_{\text{Clusters}}|S) + \ldots = 1.$$

To compute $P(C_i|S)$ we can use Bayes theorem. $P(S|C_i)$ models how a particular intent explains the current selection of the analyst. It is scaled by the term $P(C_i)$, which is called the *prior*. It describes the probability of each intent without considering additional information. Finally, $P(S)$ acts as a normalizing constant that ensures that the result is a probability. To make this equation computationally tractable, we make use of two observations. First, if we do not consider the order of selections, the problem that we are trying to solve is very similar to text classification. Our description of the analysts' selection is almost identical to a *bag-of-words (BOW)* model, which is often used in this domain. The difference is that typically, in text classification, the bag-of-words model describes the frequency of each

word. In our method, the BOW model simplifies to a constant frequency of one if a point is part of the selection. Second, by assuming that each feature (selected point) is independent of another, we can compute $P(S|C_i)$ using the *naive Bayes* method. In particular, we use a *multinomial naive Bayes* classifier to compute the conditional probabilities. For each selection by the analyst, we train such a classifier on the output vector of each of the intents C_i. Given a selection S as an input, the classifier yields the corresponding probability. Our prediction is then the intent that maximizes this probability. Sometimes, selected points are not part of any of the training samples, which leads to zero probabilities for the intent. This is a common problem when using naive Bayes classifiers. We use *Laplacian smoothing* to avoid this effect.

4.4 Visualization and Interaction Design

In this section, we describe how we implemented our methods in an interactive visualization system and explain our visualization design decisions. The interface allows analysts to add scatterplots as desired. Categories can be visualized using glyph types (see Figure 4.5). We provide a paint-brush feature [33] with three brush sizes, rectangular brushes, and individual, click-based selections. The items in multiple rectangular brushes can be treated as unions or as intersections within or between multiple plots. Points that are selected individually or using the paintbrush are always treated as part of the intersection. The labels of the items in a selection are shown in a separate view (see Figure 4.6), where we also break down the number of items in the union and intersection of multiple brushes.

We designed the **prediction interface**, shown in Figure 4.6, as a ranked table with scores shown as bar charts [114]. Each predicted pattern is a row. Hovering over a prediction shows a preview, and clicking it replaces the selection with the prediction. The different scores are shown as bar charts in the columns "Intent Rank" (the Jaccard index), "autocomplete Rank" (the Jaccard index modified to be sensitive to partial selections), and "Probability." The table can be sorted based on these scores. Other columns denote the "Matches (M)," i.e., the number of points that the prediction and selection share; the "Not Predicted (NP)" items, i.e., the number of items in the selection but not in the prediction; and the "Not Selected (NS)" items, i.e., the number of items in the prediction but not in the selection. Combined with the similarity scores, these numbers give analysts a sense of

how well each prediction matches the selection. Hovering over each of the M, NP, or NS numbers highlights the corresponding items in the scatterplot in green (see Figure 4.6).

Each prediction also shows on which dimension it was calculated (and their order) in the "Dims" column. We use short labels, which we replicate on the axes of the scatterplots to identify the dimensions. For range queries, we display the dimensions that are used in the decision tree.

When using **autocomplete**, analysts can sort by the autocomplete score. In addition, a pop-up appears right next to a selection in the scatterplot (Figure 4.7) showing the top three predictions for the current selection according to the autocomplete score. This popup can be used as a shortcut to complete selections.

To enable **reproducibility and recall**, a provenance graph is visualized in the history view (Figure 4.6) [1]. Every persistent action, such as adding a plot or making a brush, is logged in the interface and can be retrieved at a later time. The provenance graph supports branching analysis histories. A prediction can be logged as a semantically meaningful insight, which can be supplemented with an annotation (see the annotation interface in Figure 4.5). Textual annotations are designed to connect the pattern-based intent in the data to the high-level, domain-specific goals. We use insights to group and aggregate the provenance graph: All actions that were in service of a particular insight are grouped together and can be collapsed. This grouping allows us to show a concise and semantically meaningful analysis history while still storing a complete history of interactions. The example in Figure 4.6 shows one expanded group, indicated with an orange frame, and one aggregated insight in the inactive branch on the left.

The provenance graph contains all the information that is necessary to reconstruct the semantics of a selection, which means that a selection is not just a list of IDs but contains, for instance, the explicit range query or the cluster centroid and the algorithm configuration that can be used to reproduce a specific pattern on updated data. In the future, we plan to export the intents into machine-readable form so that an interactive analysis and filtering session can be used, for example, in computational notebooks.

We chose to use scatterplots and point/brush-based selections for our prototype because Scatterplots are a commonly used and widely understood visualization technique and are well-suited for brushing items. Combined with highlighting, scatterplots allow

us to demonstrate analyst selections and system predictions clearly. We chose to focus on selections because they are not only important by themselves but also are frequently precursors to more complex interactions like filtering, grouping, labeling, or segmentation. Our goal was to demonstrate the capturing of intent at the selection stage.

The visualization system described here is a technology demonstration that we developed with the goal of showing and validating the methods to detect and capture intent. We expect that a production system using our approach will use a simplified user interface for ranking intents, potentially closer to the simplified selection interface we used for our study (shown in Figure 4.7).

4.5 Results

We have implemented our prediction approach in an open-source prototype and have also provided a variety of real and simulated datasets. An online version of the tool is available at https://vdl.sci.utah.edu/predicting-intent/, and the source code is available at https://github.com/visdesignlab/intent-system.

We demonstrate our results through examples of brushes and the matching prediction. Figure 4.5, for example, shows a partially selected cluster that is also predicted as a cluster. Figure 4.6 shows a brush that closely matches a category and a range. Figure 4.7, which shows the study stimulus, gives an example of how our system can be used to autocomplete complex brushes. The plot, overall, shows a strong linear correlation between X and Y. Here, a participant has selected four points in a dataset (the four points in the top-left corner) and intended to select outliers. Our system recommended a list of predictions on the right and shows the top three predictions on the plot itself. Selecting a prediction in the pop-up or the ranked table on the right reveals the points recommended for autocompleting the selection in green. Here, the first pattern matches the outliers above the main trend, and the third pattern matches all outliers, including those above and below the main trend. We provide further examples for all patterns in the supplementary material and refer to our prototype for an interactive demonstration.

4.6 Evaluation

We explored various approaches to evaluate our methods. First, we decided to focus our evaluation on the primary contribution of this chapter: A set of methods to detect and capture analysts' pattern-based intents behind selections. We do not evaluate details of our visualization design, as they are meant to be technology demonstrations in service of our primary contribution. To validate our primary contribution, we have to determine how well our predicted intents match the mental models of analysts. To do this, we considered qualitative evaluation with experts using our system, case studies, usage scenarios, and quantitative evaluation. We ultimately chose a two-pronged approach: A demonstration of our results through a prototype with many preloaded datasets illustrated by a discussion in the previous section and the supplementary material and a quantitative evaluation. For our quantitative study, we had to make trade-offs among ecological validity (realism), internal validity (isolating factors), and external validity (generalization), and we opted for a controlled crowdsourced study with a simple interface to have control over the factors (internal validity) and to include diverse participants, beyond just experts (external validity) [115].

We validate our predictions using autocomplete as an application scenario (see Figure 4.8-I). This choice is pragmatic: Even though our ability to capture mental models is at least equally important to autocomplete, evaluating autocomplete is significantly easier because it enables us to run a large-scale study and cover various types of patterns. We argue that success in predicting correct pattern-based intents in an autocomplete scenario is transferable to capturing pattern-based intents for the purpose of reproducibility and reuse.

In our study, participants were instructed to select a specific pattern in two conditions: Either using only brushes or using brushes and autocomplete based on our prediction system (see Figure 4.8-II). The study is designed to test the validity of our approach using two primary measures: **Accuracy** and **match between intended and predicted pattern**.

We chose a subset of our patterns: Correlations (linear and quadratic), outliers, clusters, and multivariate optimization. We excluded ranges since they cannot be used for autocomplete, and categories since selecting elements belonging to categories would be tedious in our system without autocomplete, and yet an alternative user interface design that enables

participants to explicitly select categories would solve that problem trivially.

We also describe the study in a data comic, shown in Figure 4.8, because data comics are an effective way to communicate the complex procedures of a study [116].

4.6.1 Procedure

We used a within-subjects design for two conditions: **User-driven**, using only manual brushes, and **computer supported**, which adds a simplified version of our prediction interface. Participants were instructed to select points that belong to a specified pattern. The interface, shown in Figure 4.7, was simplified to show only the rankings tailored to autocomplete. The names of the predicted patterns were not shown to avoid biasing participants. To counter-balance any learning effects, the conditions were assigned in random order, the task order in each condition was randomized, and the datasets were randomized. We recruited 128 participants for the study on Prolific, a crowdsourcing platform with a research focus. Based on the completion times of pilot experiments, each participant was paid $ 6.25 USD for an estimated duration of 25 minutes, resulting in an hourly rate of about $ 15 USD. All participants viewed and agreed to an IRB-approved consent form. To be eligible, participants had to use a laptop or desktop device and either the Chrome or Firefox browsers.

Our procedure consisted of five phases (see Figure 4.8-III) and followed guidelines on training participants for complex analysis tasks [107]: *Passive training*, in the form of an 8-minute video introducing the types of patterns and the interface; *active training*, where they had to complete representative tasks, but could use a help-feature to reveal the answer; *trials* in the two conditions; and a short *poststudy survey*. The full study with all phases is available online.

Our initial study had a negative result for outlier detection. We found that our outlier prediction algorithms did not perform adequately after investigating the reasons behind this result. The algorithm we used at that time was the Local Outlier Factor (LOF), which compares the local density of the object to the density of its neighbors. The k-nearest neighbors algorithm gives this local density. The algorithm is good at detecting local outliers, i.e., data points that are some distance from a dense cluster are considered outliers, whereas a point that is far from a sparse cluster might be considered a part of the cluster

due to the cluster being sparse. We added another outlier detection algorithm, DBSCAN, to the interface. DBSCAN clusters the given data into density-based clusters and marks the points lying in the low-density regions as outliers, which ensures that points that are outliers with respect to the entire dataset are correctly marked as outliers most of the time. We re-ran the study only for the outlier task. We edited the instructional video to remove explanations on other tasks; otherwise, the procedure remained the same. We recruited 130 participants for the revised outlier-only study. Participants were paid $ 3.52 for an estimated study duration of 14 minutes. For transparency, we report on both our original outlier results (referred to as "outlier old" going forward) and the revised outlier condition ("outlier revised").

4.6.2 Data and Tasks

We generated synthetic 2D datasets with between 200 and 222 items for 1) linear correlations, 2) quadratic correlations, 3) outliers combined with a linear correlation, 4) outliers combined with a single cluster, 5) clustered datasets with three or four clusters, and 6) datasets for multivariate optimizations, each in three levels of difficulty: Easy, medium, and hard. The levels of difficulty were driven by how apparent a pattern is. For example, an easy clustering dataset had fully separated clusters, whereas a hard dataset had clusters that significantly overlap. We generated two variations of each combination (to be used in the different experimental conditions) for a total of 36 datasets for the study and 6 datasets for training tasks. For each dataset, we **generated ground-truth through human labeling**. Patterns such as clusters or outliers can be ambiguous, and our goal was to match the human perception of those patterns. Hence, we chose to ask expert coders to label the datasets. Our coders were five doctoral students in visualization not involved with this work, with experience analyzing these patterns. We instructed them to carefully label each dataset for a specific pattern, with no algorithmic support. We then treated all points that four to five of our coders selected as correct, the points that two to three coders selected as ambiguous (neither correct nor incorrect), and the points that only a single or no coder selected as incorrect. The supplementary material contains images of the datasets, the ground-truth labels, and the code used to generate them.

The **tasks** instructed participants to select one of the patterns they learned about during

training. As an example, for outliers, the prompt was: "Select the points that are outliers, i.e., that are not following the main pattern you see in the data" (see Figure 4.7). For clusters, a specific cluster was marked in the plot with a red cross, and the prompt was "Select the points which belong to cluster centered around the cross."

4.6.3 Measures

We **measured accuracy, time to completion, the type and rank of a predicted pattern chosen by a participant, and survey responses**. After each question, we also elicited confidence and perceived difficulty on a 5-point Likert scale and asked for comments. We also logged detailed interactions in a provenance graph. We **calculated the accuracy** of the participant's responses by using the **Jaccard index** of the response overlapping with the ground-truth, where we first removed the ambiguous points (hence, selecting ambiguous points neither benefits nor penalizes a score). For the time measures, we subtracted the times when the browser window showing the study was inactive. The final survey asked about satisfaction with different features and experience with visualization and statistics. Demographic data are provided through Prolific participant profiles.

4.6.4 Pilots, Analysis, and Experiment Planning

We conducted several tests and pilots to evaluate tasks, system usability, data collection modalities, measures, and our procedure. We estimated the number of participants required to uncover effects based on a pilot run on Prolific with 10 participants. We used a power analysis to estimate the variance in our measures, which we combined with our observed means to estimate the number of trials required. Due to the limitations of null hypothesis significance testing, we base our analysis on best practices for fair statistical communication in HCI [117] by reporting confidence intervals and effect sizes. We compute **95% bootstrapped confidence intervals** [118] and **effect sizes** using Cohen's d to indicate a standardized difference between two means. For the accuracy values, we also supplement our analysis by including p-values from Wilcoxon signed-rank tests (given the non-normal distributions of our data and the within-subjects design). We consider a Bonferroni-corrected threshold for significance of $p = 0.0083$.

4.6.5 Expectations

We expected that accuracy would be higher using computer-supported mode for the medium and hard datasets and that accuracy would be about the same and consistently very high with the easy datasets. We assumed that the value of the prediction system would be greater on ambiguous patterns and that obvious patterns would be easy to select manually, given the brushing tools we provided. We also expected that participants would perceive predictions as accurate and the interface as user-friendly, and they would prefer the computer-supported mode. Finally, we initially also expected computer-supported mode to be faster, but we realized during testing and pilots that this would unlikely be the case.

4.6.6 Results

The original study had 128 participants, and the follow-up outlier study had 130 participants. After reviewing the provenance data using the reVISit system [119], we realized that participants sometimes chose not to use predictions in the computer-supported condition. Since our goal was to measure the effects of using predictions, we removed trials that were not completed using predictions in the computer-supported mode. To avoid biasing our data by removing low-effort results in one condition, we also always removed the equivalent trial in the user-driven mode. We include data for all trials in our supplemental material. Based on these criteria, we retained 1381 of 2268 trials in each of the computer-supported and user-driven conditions (826 of 1560 for the second study). Hence, when autocomplete was available, participants chose to use it in 61% of cases (53% for the second study). We argue that the removal of these trials is justified and even necessary, but this argument leads to the question of why predictions were not used in many cases. We do not have definitive answers for that question since conducting follow-up interviews with crowd participants is not possible. We do believe, however, that skipping the prediction interface is a sign of a crowd participant minimizing their effort, which is well known to be a challenge in crowdsourced studies [120].

We analyze easy, medium, and hard tasks together, resulting in 1785 valid trials across both studies. Figure 4.9 summarizes our main results. Accuracy and speed for every task are shown individually in Figure 4.10 and Figure 4.11. We also break down results by

levels of difficulty (see Figures 4.12 to 4.14). Accuracy was fairly high in both conditions for clusters, linear regression, and quadratic regression (median of 84-98%), with a **small to medium, significant effect showing higher accuracy in the computer-supported condition**. The computer-supported clustering condition shows a small "bump" at an accuracy of around 0.5. Analysis of provenance data has revealed that this bump is due to one of the clustering predictions aggregating two ground-truth clusters into one.

Overall accuracy for the multivariate optimization task was lower, with **accuracy in the computer-supported condition being significantly higher, with a small to medium effect size**. Interestingly, many of our coders omitted points that are contained in the formal definition of a skyline, resulting in a "bump" of accuracy scores at around 0.85, representing participants who have selected the formally correct skyline as recommended by the algorithm.

The accuracy for our original outliers condition ("outlier old," in Figure 4.9) was significantly lower in the computer-supported condition than in the user-driven condition. Inspection of the provenance data revealed that, in many cases, applying a prediction for outliers made user selections worse. As previously discussed, we re-ran our study using a different outlier prediction algorithm (outliers as reported by DBSCAN). In the second study, we saw **significantly higher accuracy for computer-supported mode**, although overall accuracy had gone down. The reduced overall accuracy could be caused by reduced learning effects in the study with fewer tasks.

The difference in accuracy in favor of the computer-supported condition was more pronounced in medium and hard tasks. The accuracy in easy tasks was similar for both conditions (see Figures 4.12 to 4.14).

Our exit survey revealed that participants generally found predictions accurate (average score of 3.6 on a 5-point Likert scale) and helpful (average score of 3.8 on a 5-point Likert scale). In terms of the interaction choices for selections, the paintbrush selection was rated more helpful (average 4.5/5) than the rectangular brush (average 2.3/5) and individual point selection (average 3.3). Every task was followed by a mini-survey in which participants reported their confidence in their selection and self-reported the difficulty of the task. Confidence and difficulty were reported on a 1-5 scale, with 1 being confident, 5 being not confident, 1 being easy, and 5 being difficult. Confidence was

higher, and difficulty was reported lower for the computer-supported condition for all tasks except outliers, where they were about the same, suggesting that participants trusted the predictions when they matched their mental model.

We also analyzed whether the **type of predictions chosen by participants matched the patterns they were instructed to select**, which is a useful metric to judge the quality of our predictions and rankings. We see a strong overlap between prediction and target pattern (see Table 4.1); **participants selected the right type of pattern 70% of the time from our predictions**, and used the correct pattern or a reasonable substitute (e.g., "outside linear regression" instead of "outlier") 77% of the time. There were variations between patterns: Clusters were almost perfectly matched, whereas regression patterns were less frequently correctly matched. Notably, quadratic and linear regression were frequently substituted, and nonoutliers were frequently chosen for regression tasks as well.

Time to completion was generally slower by 3–12 seconds (with completion times ranging from 21-37 seconds on average) for the computer-supported condition (see Figure 4.15). Given the higher accuracy — overall, the median accuracy for computer-supported mode was 3–9% higher (excluding the old outlier condition) — but the slower response times, it is worth asking whether **this a trade-off worth making.** In retrospect, the longer response times for computer-supported work make sense. Previous work by Saket et al. has shown that task completion times in multiparadigm interfaces can be higher compared to a single paradigm interface [69].

However, Saket et al. also argue that optimizing efficiency is not a suitable goal in many contexts and that multiparadigm tools can make analysts think more carefully. How meaningful are 10 seconds of an analyst's time when trying to understand an important dataset? We argue that accuracy by itself is much more important than time, when the time difference is a few seconds, for most analysis scenarios.

Furthermore, for our study specifically, we were able to show not only that the accuracy in computer-supported mode was higher but also that we were able to correctly predict patterns based on participant selections in most cases, which has the benefit that this data is now available as structured information that can be leveraged for reproducibility and reuse.

4.6.7 Study Discussion

Overall, our expectations were verified: Accuracy was consistently higher in the computer-supported condition, and we were able to correctly predict a large percentage of patterns. In terms of predicted patterns, quadratic and linear regression showed lower accuracy in predicting correct patterns, even when including nonoutliers as a reasonable substitute. This result is likely due to the linear and quadratic regression algorithms using our thresholding being quite similar. Creating an umbrella intent "regression" would be one possibility to address this problem.

Although we were able to show that we can successfully predict pattern-based intents for autocomplete, the question remains how useful real-life analysts will find the ability to track semantically meaningful pattern-based intents. To answer this question, we plan to develop our prototype system into a visual exploration tool that enables actions derived from selections, such as filters and groupings and provides features such as sharing, replaying, and exporting the analysis process into other pipelines or tools such as Jupyter Notebooks. We can then design a more comprehensive evaluation strategy that can validate the efficacy of this system with regard to reproducibility and reusability.

4.7 Discussion

In this work, we demonstrate a method for semiautomatically detecting and capturing analysts' pattern-based intents. Detecting intents is useful for two scenarios: To autocomplete selections and to be able to semiautomatically record semantically rich insights in provenance data and, therefore, make visual analysis processes reproducible and justifiable. By capturing pattern-based intents, we can, for example, more easily create curated analysis stories by leveraging ideas from prior work on using provenance information to create interactive data stories [50]. The capability to capture pattern-based intents opens up numerous other prospects as well.

4.7.1 Integration in Computational Workflows and Analysis Reuse

Our interviews show that analysts frequently use scatterplots in combination with statistical modeling tools and computational notebooks, such as R-Markdown or Jupyter notebooks. Having semantically meaningful intents available means that we can generate

robust analysis scripts based on interactive visualization, supporting more automatic computational workflows. For example, if an analyst uses our tool to select a specific cluster for downstream analysis, we will be able to generate code that will select this cluster even for updated data.

4.7.2 Learning from Interaction

Through large-scale capturing of intents, we can empirically learn patterns that analysts select to further improve our predictions. Such a system could dynamically "autocorrect" analysis and allow large-scale feedback on the usefulness and effectiveness of various features within complex tools. For instance, a software tool with a diverse set of users and skill levels would allow intent to be trained on experienced users so that novices are guided quickly toward effective strategies [59].

4.7.3 Generalization to Other Visualization Techniques and Data Types

We chose to limit ourselves to scatterplots and tabular data because we believe that these are important cases that can be used to demonstrate the feasibility of our approach. There are numerous extensions and generalizations of our work, ranging from implementing more brushing tools, such as lasso selections, to allowing analysts to filter datasets. We argue that our framework could be extended to other visualization techniques, such as parallel coordinates, histograms, or tabular visualizations [121] with small adaptions. Other visualization techniques could also provide additional clues we could use for predicting intents. For example, in a tabular visualization, the action of sorting a table is likely important to understand the intent of a subsequent selection. Other data types, such as time series or network data, are likely amenable to the same approach but would require identifying appropriate patterns and the corresponding algorithms.

4.7.4 Higher Dimensionality

Although we allowed analysts to explore multiple two-dimensional views, building a mental model of high-dimensional data can be difficult. A potential solution to this problem is dynamic dimensionality reduction. That is, given points already selected, the system could dynamically adjust a linear projection (e.g., PCA) to best capture those

datasets in a 1-, or 2-dimensional subspace. Alternatively, given more complex selections, like clusters of relevant points, dimensionality reduction can use techniques such as Latent Discriminant Analysis to find the best linear projection to separate the clusters. Another approach is to label pairs of points that should be close (or far). Using these pairs, a similarity learning method could provide the best linear projection that satisfies those constraints. An intent-driven tool could suggest the most informative point-pairs to label.

4.7.5 Scalability

Our current system recalculates the predictions every time an analyst interacts with the system. This delay in the prediction mechanism can be prohibitive for large datasets, large combinations of dimensions, or more parameterizations for algorithms. The prediction mechanism's two main phases are to run the machine learning algorithms on the datasets and to rank the results based on an analyst's selection. Only the second step has to be done in real time. The first computationally expensive step can be done once, on data upload, as an offline step. Precomputing would also allow us to include a larger combination of dimensions and add more algorithms and parameterization without substantially adding to the prediction time.

4.7.6 Active Pattern Exploration

Instead of just passively suggesting which pattern matches a selection, we could also suggest various patterns in the data set as possible aspects to explore at the beginning of an analysis. In this way, analysts could, for example, see all computed skylines without ever using selections. The downside of such an approach is the potential for increased complexity in the UI: Analysts would be confronted with many different analysis choices, and rankings or suggestions would be difficult to achieve without prior input from the analyst.

4.7.7 Multiverse Analysis

As in any multistep analysis process, analysts must make choices about their analysis paths, leaving other reasonable paths behind. As we capture analysis paths explicitly, it would be intriguing to also explore different analysis paths from the multiverse and visualize the results of these alternatives using a framework like Boba [122].

4.8 Limitations

Even in a simple scatterplot environment, our work has identified numerous complexities. When more than four dimensions of a dataset are relevant in the exploration, the combinatorial complexity of all the possible intents we model is significant. One potential solution is to automatically filter entire classes of intents so that not all of them need to be explicitly explored.

Showing many scatterplots also raises the problem of fitting them visually on the screen. Predicting intents in higher dimensions based on selections in 2D scatterplots is tricky because scatterplots do not provide a good visual representation of high-dimensional patterns. We plan on addressing this problem by providing additional visualizations for visualizing high-dimensional data, such as parallel coordinate plots or heat maps, or by using dimensionality reduction.

Our tool currently does not handle missing data. When working with our collaborators, we frequently encountered datasets that were generally well-suited to our approach but contained invalid or missing cells. On the front end, we plan to provide separate views for items with missing data. On the back end, appropriate interpolation and fitting strategies could be a solution.

Our current approach to parameter space exploration is naive. We could improve our prediction by evaluating our classifications using methods such as silhouette analysis for clustering and varying the parameters accordingly.

In some cases, meaningful selections might not correspond to predicted patterns, yet our ranking system will still recommend a pattern, although with low scores. We considered including a "no pattern" prediction in the ranking but ultimately decided against it since it would be difficult to rank in either of our ranking frameworks. However, analysts can explicitly record "custom insights" that are not based on any ranked pattern to account for the same.

As analysts interact with a visualization over a longer period of time, the provenance graph keeps growing. The Trrack library [1] can demonstrably handle a large number of actions, but the provenance visualization will become hard to navigate. As a partial remedy, we group the actions in a provenance graph when an insight has been generated, which allows the collapse of the provenance visualization to give a higher level overview.

We can improve this approach by allowing analysts to manually group provenance actions and collapse the tree further. Finally, search functionality on the provenance graph may be a scalable solution to the problem [52].

4.9 Conclusion

In this chapter, we introduce the first approach to predict, capture, and annotate pattern-based intents of analysts as they interact with data in a scatterplot. We use a mixed-initiative approach, leveraging data mining methods to identify patterns in datasets, ranking potential matches based on selections, and allowing analysts to specify which (if any) of the predicted intents fit their actual intents. We discuss two application scenarios: Autocompleting selections and increasing reproducibility. We believe that our work will form the foundation of many future projects. The immediate next step is the application of different visualization techniques and data types. Other prospects include learning from interactions and integrating the output of interactions in visualizations into computational workflows.

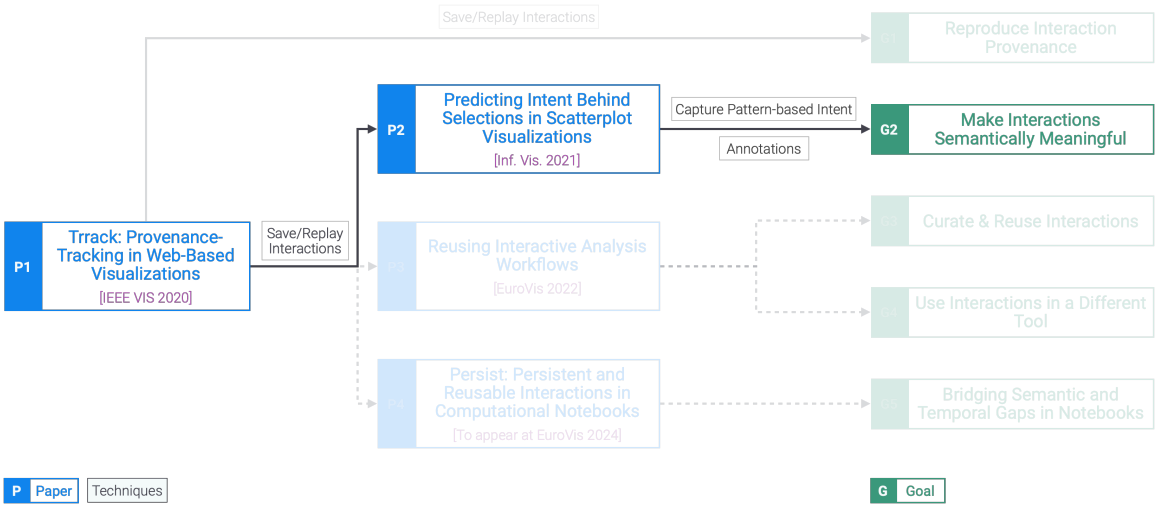

Figure 4.1: Techniques for making interaction provenance semantically meaningful. Interaction provenance captured by Trrack is used to semiautomatically infer the pattern-based intent of the analyst for an interaction. Analysts can also annotate any interaction in the provenance. Augmenting the interaction provenance with the analyst's intent and the analyst's annotations allows us to more meaningfully reproduce the interactions.

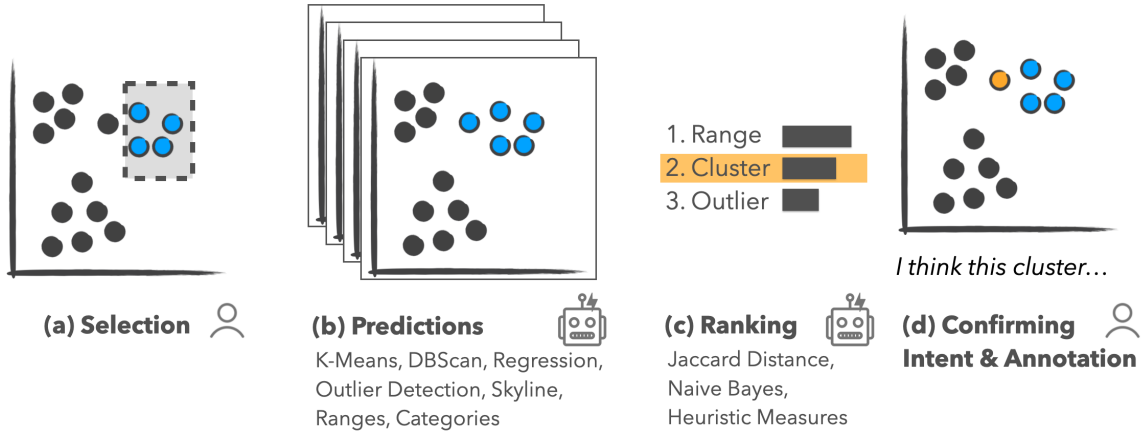

Figure 4.2: Overview of our method for mapping patterns to intents. (a) An analyst makes a selection in a scatterplot. (b) The system calculates many different patterns using various algorithms that we use for prediction. (c) We rank how well the analysts' selection matches the predicted patterns. (d) The analysts select their intended pattern and provide annotations to capture their thoughts.

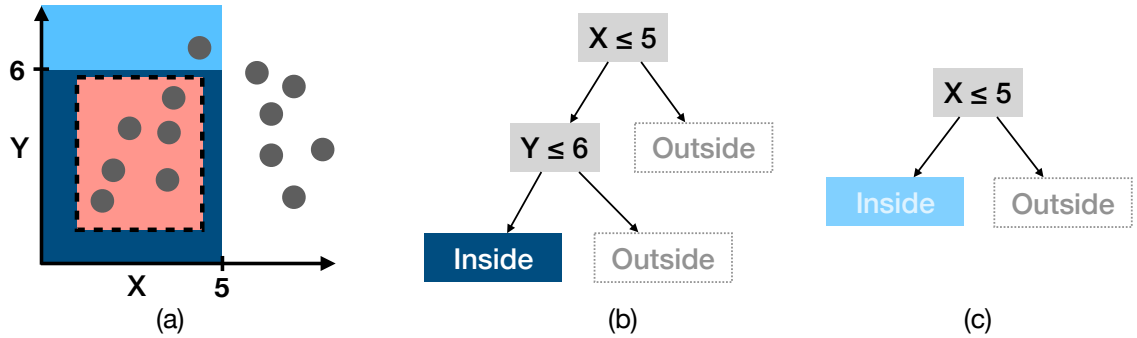

Figure 4.3: Using decision trees to capture range-based queries. (a) A brush is shown in red. The brush geometry can be described with four rules. (b) The decision tree simplifies the brush to two rules, illustrated in dark blue in (a). (c) A simplified decision tree where one level has been removed. The result is a simple rule, which also includes a point that was not in the original selection, contained in the light-blue area in (a).

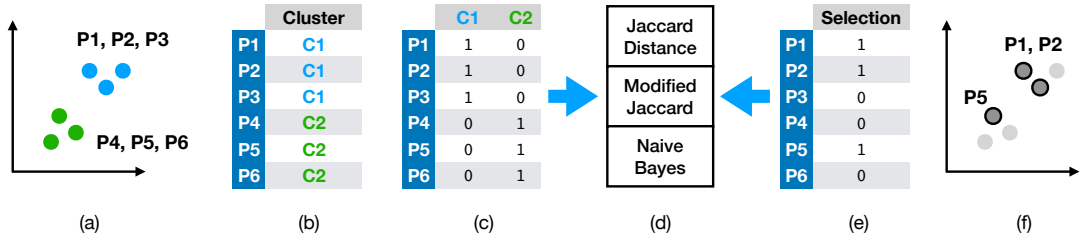

Figure 4.4: Overview of our method for ranking patterns for an analyst's selection based on a clustering example. (a) A dataset exhibits two clusters, shown in blue and green. (b) A clustering algorithm detects the clusters and assigns labels to the points. (c) We use one-hot encoding to transform the output of each algorithm into disjoint Boolean vectors. (f) An analyst's selection results in (e) another Boolean vector. (d) These Boolean vectors act as inputs to compute Jaccard indices and the naive Bayesian classifier, which are then used as scores for ranking.

Figure 4.5: Scatterplots showing three dimensions of a dataset. An analyst has brushed points in the right scatterplot based on a pattern they see (orange points). Our system predicts possible intents and ranks them by their match to the current selection. The points in green show a cluster that is recommended by our system based on the selection. When an analyst accepts this suggestion, a semantically rich log entry is stored in a provenance graph, shown on the right.

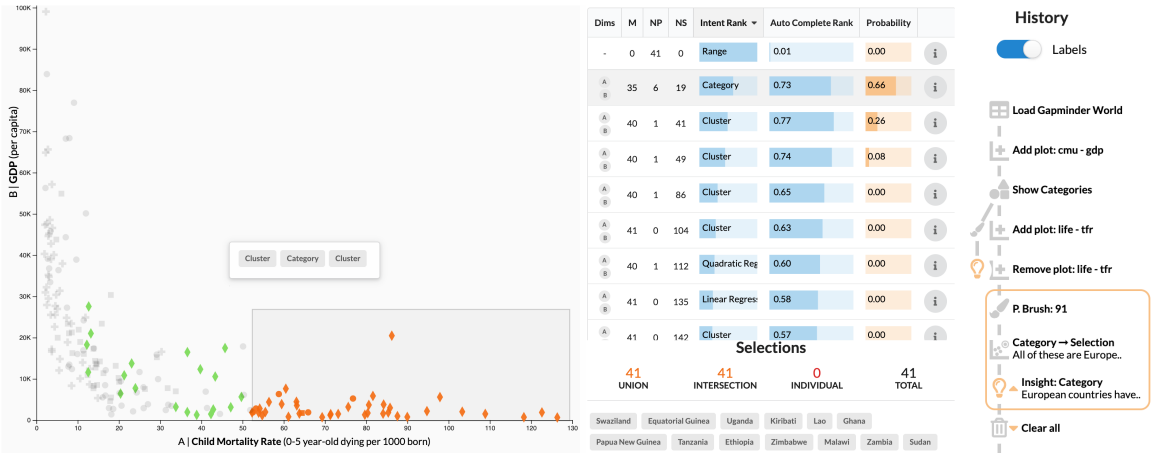

Figure 4.6: The prediction interface shows ranked patterns based on the three scores. The "Category" prediction for a selection (orange points, rectangle brush) is shown in green in the scatterplot. Hovering over a row in the prediction interface shows a preview of the prediction. Clicking the row replaces the current selection with the predicted selection. The M, NP, and NS columns show the number of matching items (M), not predicted items (NP), and predicted but not selected items (NS). Hovering over a cell highlights the corresponding items in the scatterplot in green. The "Dims" column displays the dimensions considered for calculating a pattern. The "Probability" column encodes the probabilistic ranking. The provenance visualization (right) shows the steps that lead to the current selection and prediction. Insights (orange) are used to group and aggregate steps that lead to them.

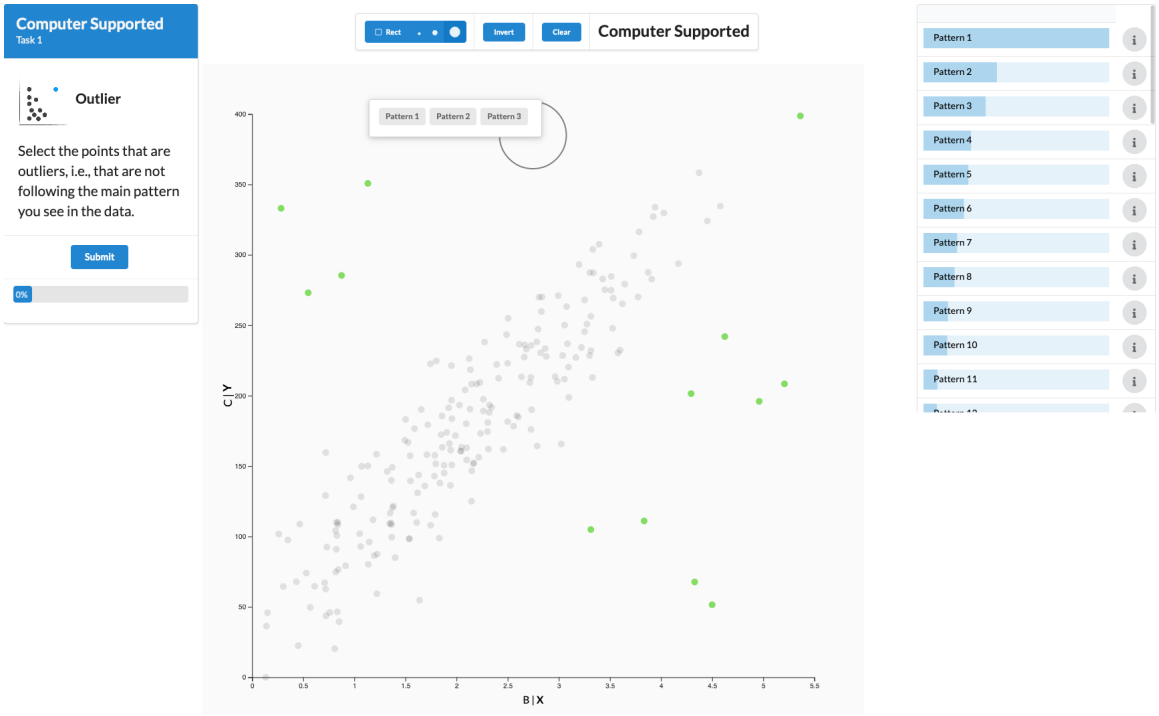

Figure 4.7: The study interface for the computer-supported condition for an outlier task. The user-driven condition was identical, except for the absence of the ranking on the right and the prediction pop-up.

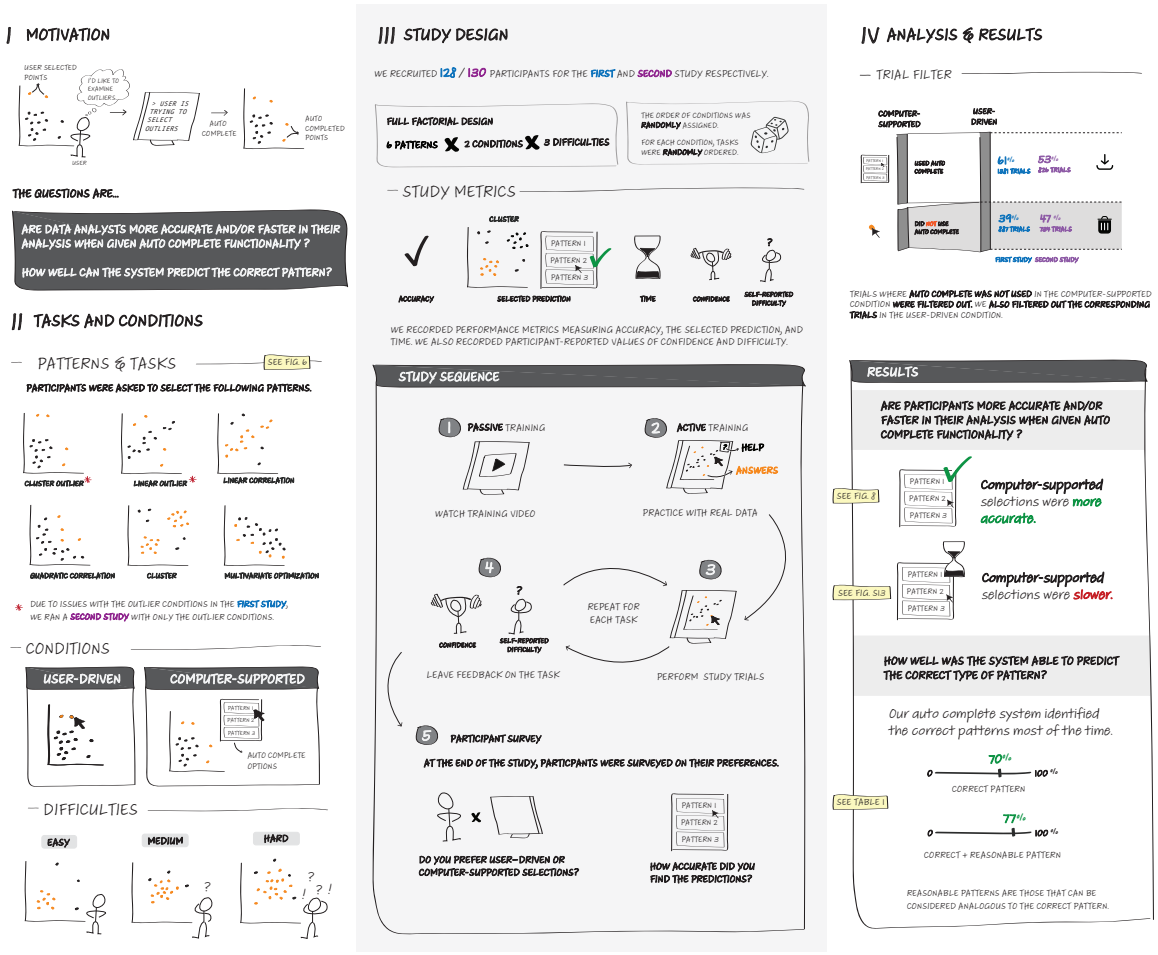

Figure 4.8: Data comic [116] showing the motivation, tasks, conditions, study design, analysis, and results of our study.

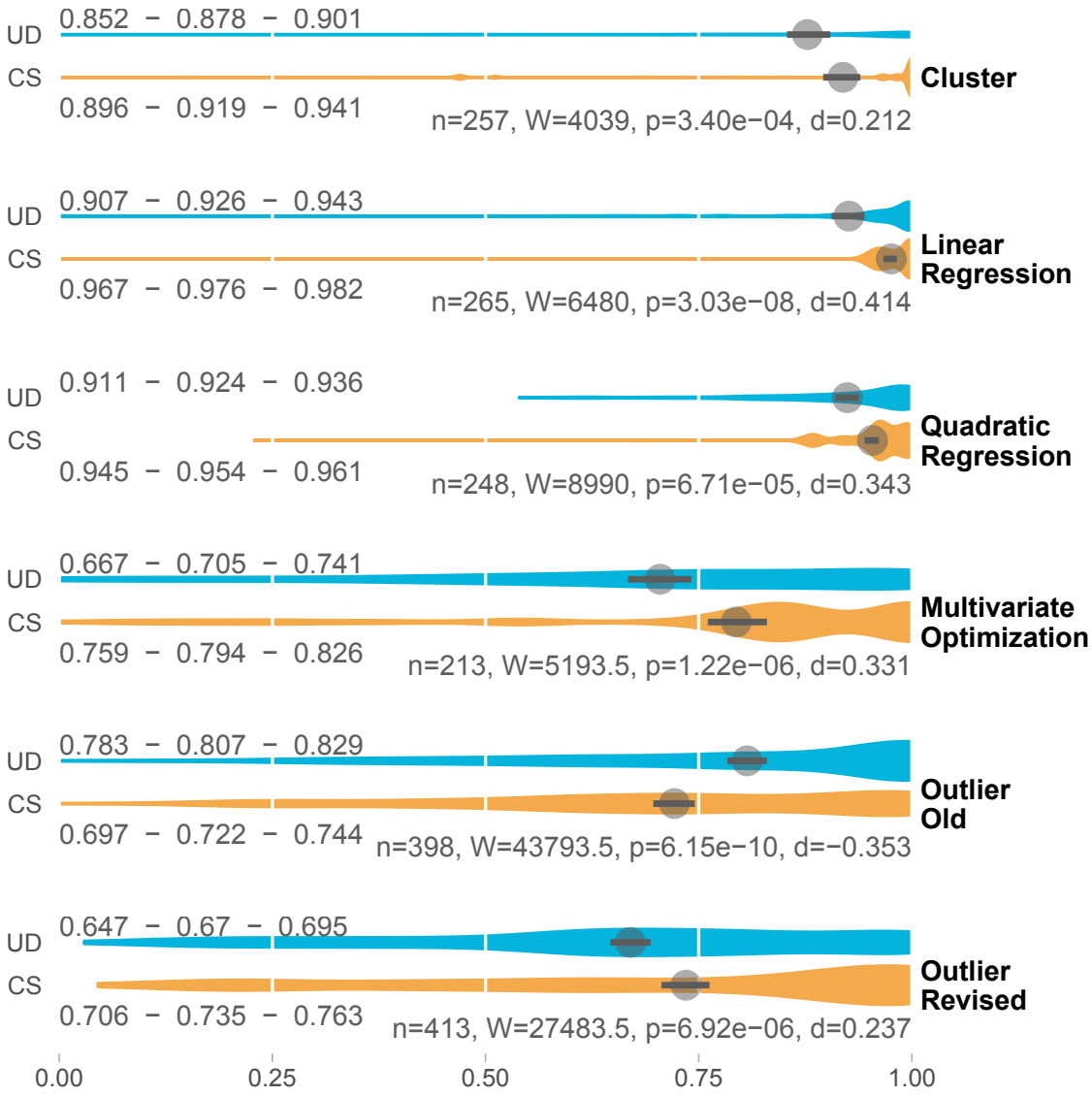

Figure 4.9: Task-specific accuracy shown as medians and 95% confidence intervals on a scale of 0–1. Blue (UD) encodes the user-driven condition, orange (CS) the computer-supported condition. Violin plots visualize the underlying distribution. The numbers on the left show the median and the extent of the 95% confidence interval. We also give the number of trials per condition for each task (n), Cohen's d for effect sizes (d), and p-values. All differences are significant. Note that the number of trials varies due to our exclusion criteria and that the outlier tasks have higher numbers of trials as they group multiple outlier configurations (outliers on top of clusters, regressions, etc.) Also note that "Outlier Old" shows the result of our original study, whereas "Outlier Revised" shows the result of a separate study with an improved outlier detection algorithm.

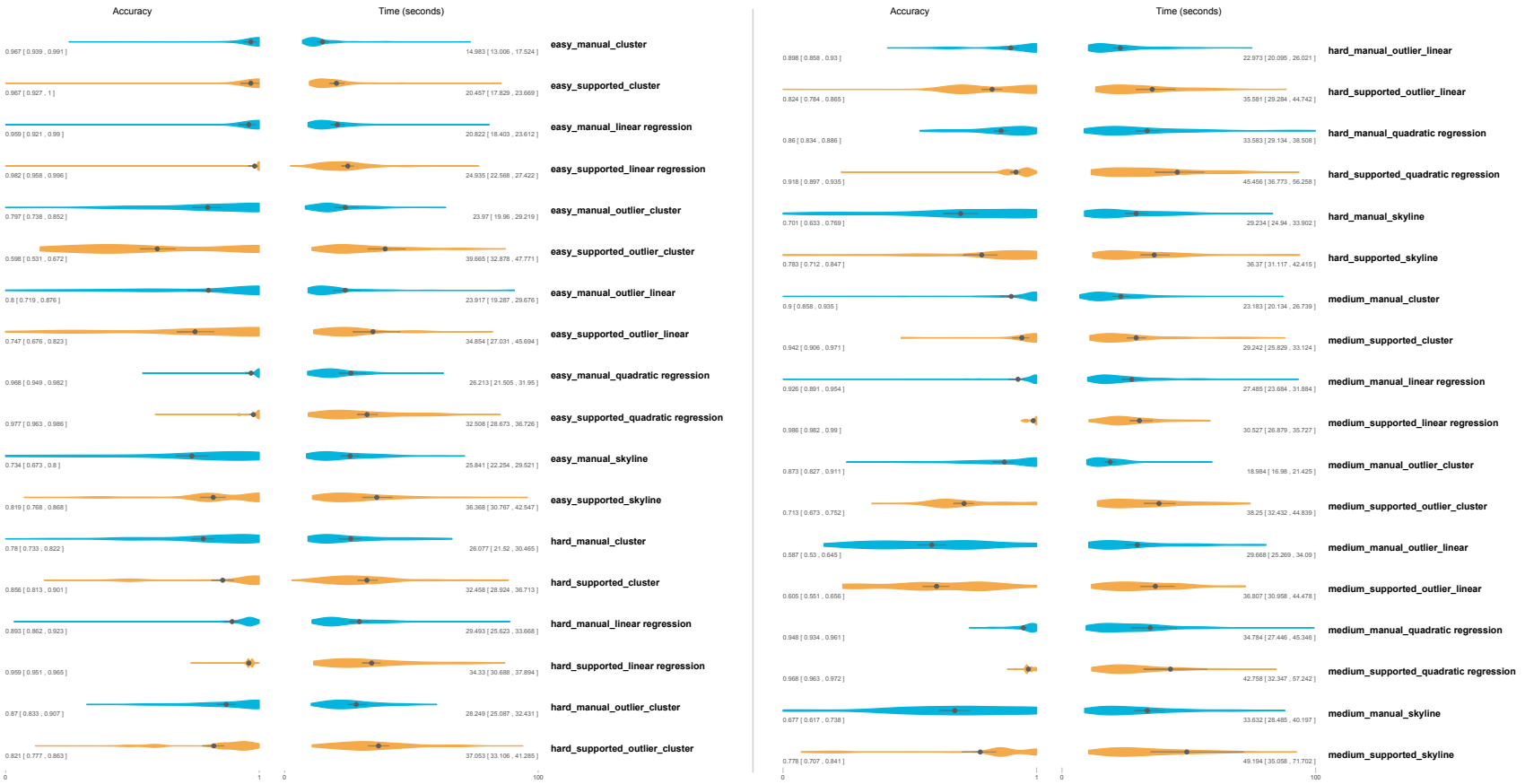

Figure 4.10: Accuracy and time in seconds for all tasks during the initial study. Blue (UD) encodes the user-driven condition, orange (CS) the computer-supported condition. Violin plots visualize the underlying distribution.

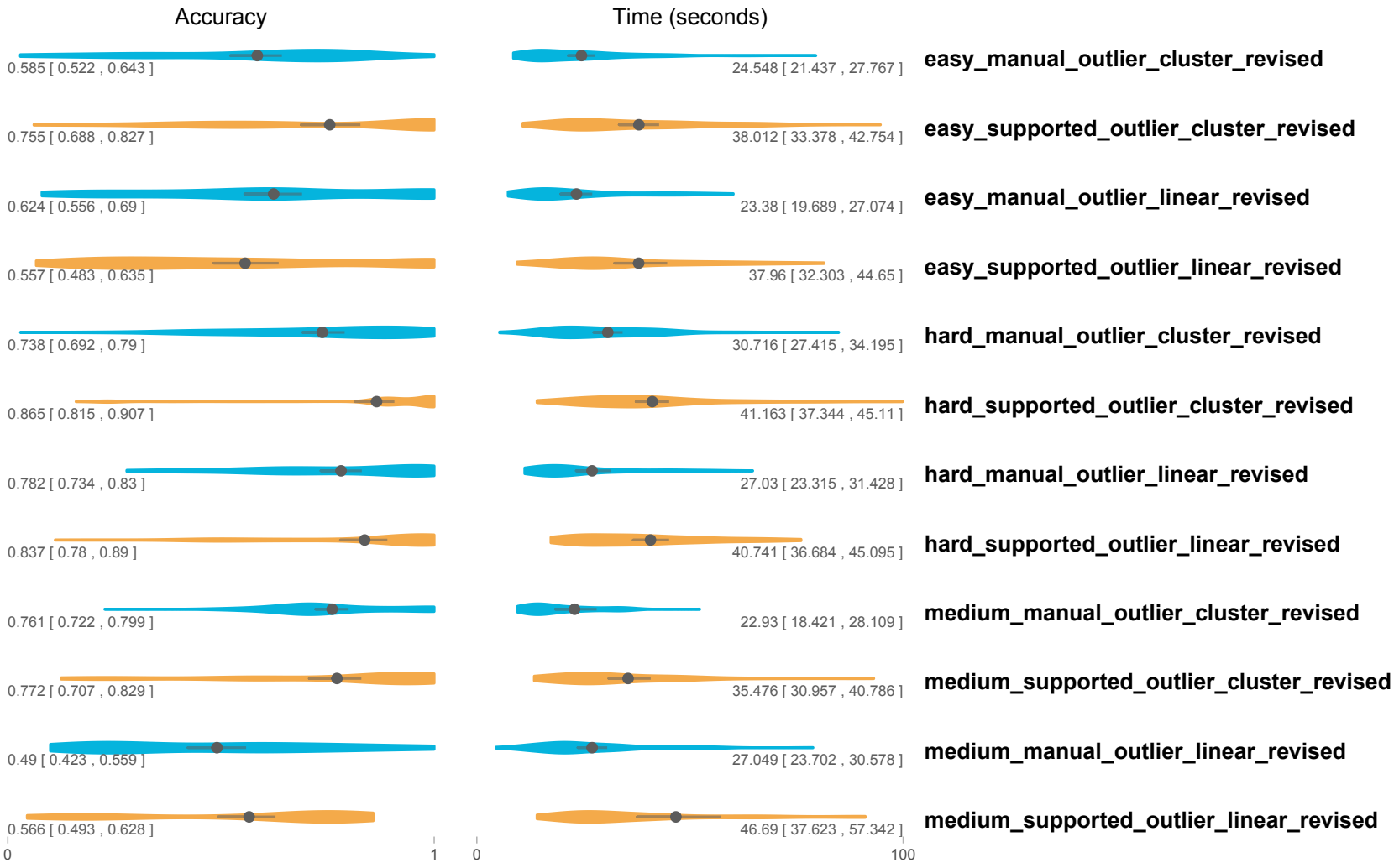

Figure 4.11: Accuracy and time in seconds for all tasks during the revised study. Blue (UD) encodes the user-driven condition, orange (CS) the computer-supported condition. Violin plots visualize the underlying distribution.

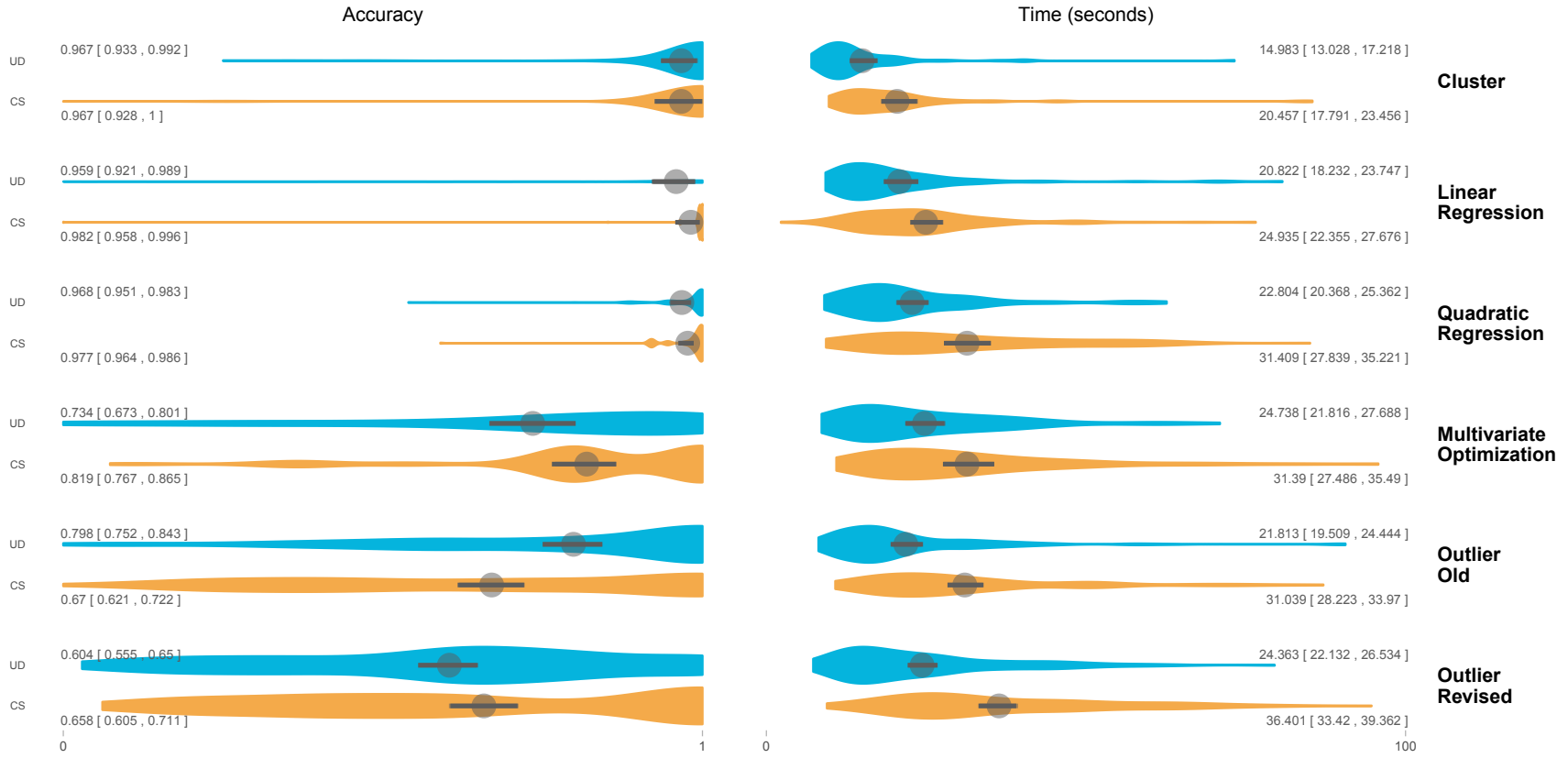

Figure 4.12: Accuracy and time in seconds for easy tasks. Blue (UD) encodes the user-driven condition, orange (CS) the computer-supported condition. Violin plots visualize the underlying distribution.

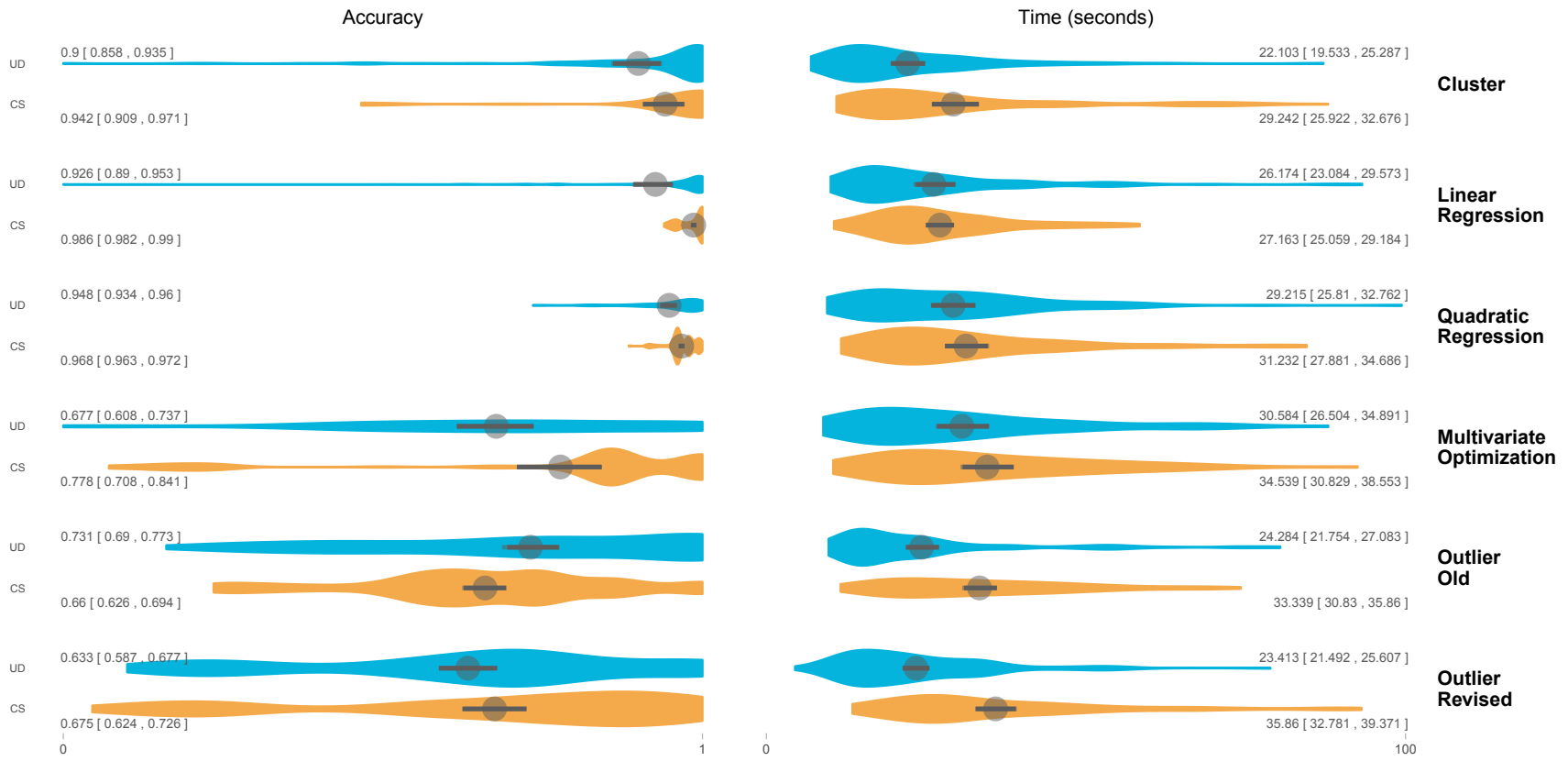

Figure 4.13: Accuracy and time in seconds for medium tasks. Blue (UD) encodes the user-driven condition, orange (CS) the computer-supported condition. Violin plots visualize the underlying distribution.

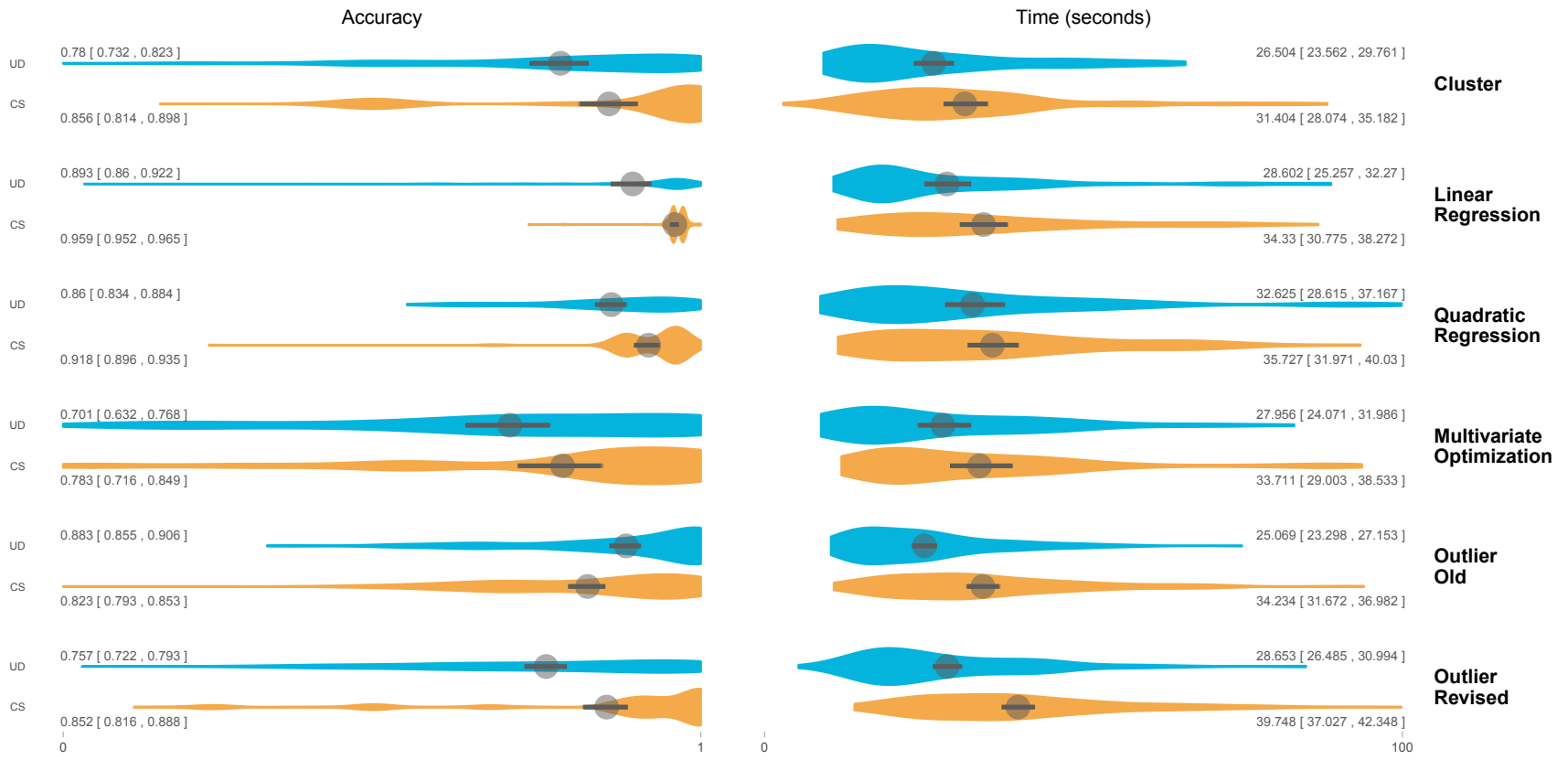

Figure 4.14: Accuracy and time in seconds for hard tasks. Blue (UD) encodes the user-driven condition, orange (CS) the computer-supported condition. Violin plots visualize the underlying distribution.

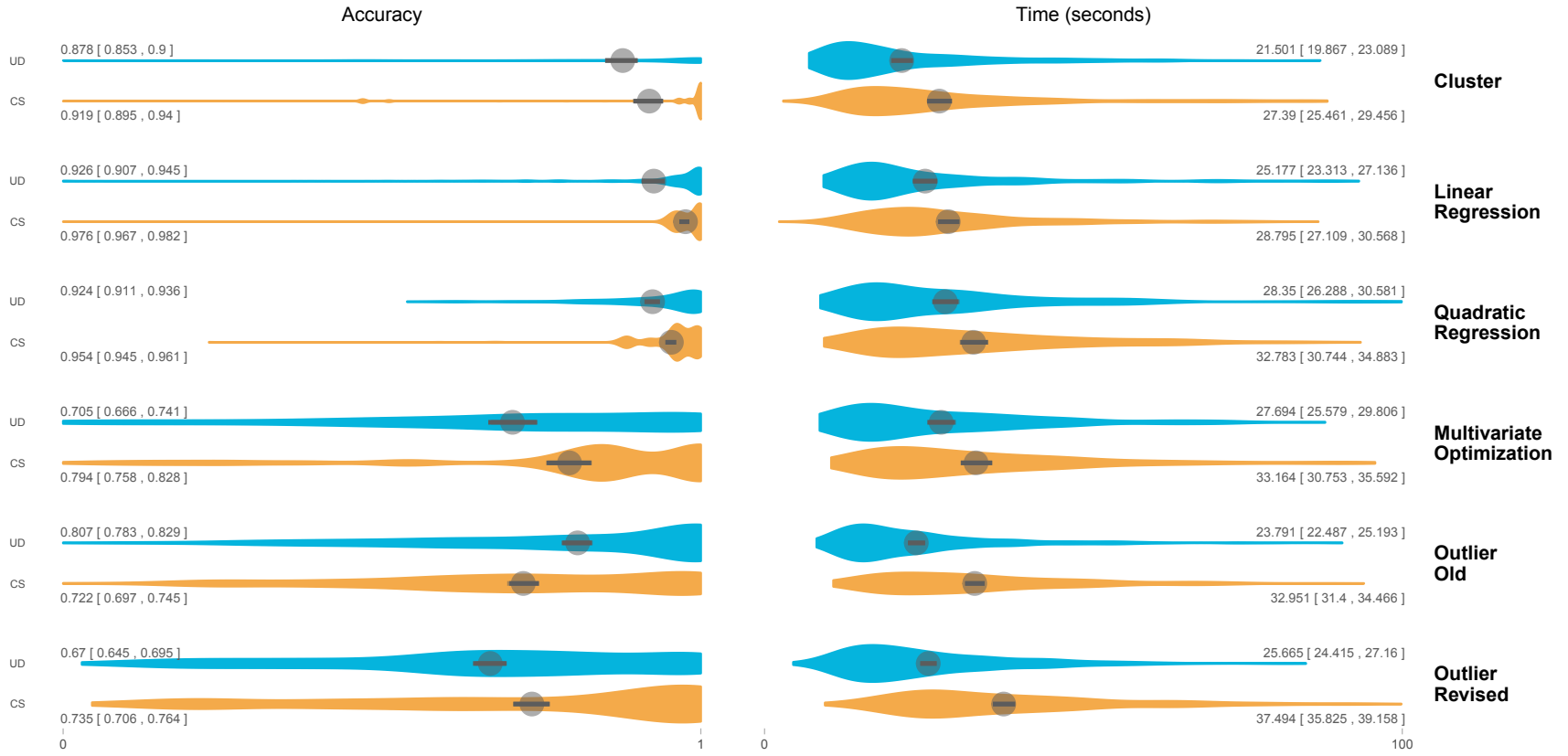

Figure 4.15: Accuracy and time to completion for all tasks. Blue (UD) encodes the user-driven condition, orange (CS) the computer-supported condition. Violin plots visualize the underlying distribution.

Table 4.1: To analyze the matches between patterns, participants were instructed to select (columns) and patterns they chose from our set of predictions. The values in the cells show the number of times participants selected a certain predicted pattern for a pattern they were asked to select. For example, the first column, "Cluster," contains responses for all trials that instructed participants to select a cluster. The first row, also "Cluster," shows all trials where participants chose a cluster pattern from the predictions. Consequently, the first cell shows that when asked to select a "Cluster," in 256 trials, participants have also selected a predicted pattern of type "Cluster," Cells containing precise matches are highlighted in green. Cells highlighted in yellow show matches with analogous patterns. For example, the "outside linear regression" pattern is a reasonable substitute for the "outlier" pattern. The numbers in parentheses show how frequently a response resulted in a correct solution (assuming correct to be an accuracy > 0.75). The bottom two lines give percentages for the correctly identified patterns and correct+analogous patterns combined. Again, the number of responses with an accuracy > 0.75 is in parentheses. Overall, correct patterns were used frequently for clusters, multivariate optimization, and outliers. Results for linear and quadratic regression show a frequent mix-up with the other type of regression pattern, and for quadratic regression also with clusters. Interestingly, these mismatched patterns do not negatively influence accuracy.

		Pattern Asked For in Tasks						
		Cluster (N=257)	Linear Regression (N=265)	Quadratic Regression (N=248)	Multivariate Optimization (N=213)	Outlier Old (N=398)	Outlier Revised (N=520)	Average
Pattern Predicted	**Cluster**	**256 (223)**	6 (6)	82 (78)	13 (2)	43 (25)	44 (5)	—
	Linear Regression — Within	0 (0)	**136 (136)**	54 (54)	1 (0)	4 (1)	1 (0)	—
	Quadratic Regression — Within	0 (0)	82 (82)	**84 (84)**	0 (0)	2 (1)	0 (0)	—
	Multivariate Optimization	0 (0)	0 (0)	0 (0)	**175 (163)**	21 (10)	22 (22)	—
	Linear Regression — Outside	0 (0)	1 (0)	0 (0)	0 (0)	**0 (0)**	**60 (44)**	—
	Quadratic Regression — Outside	0 (0)	0 (0)	0 (0)	2 (0)	**0 (0)**	**13 (0)**	—
	Non-Outlier	1 (0)	**40 (39)**	**28 (28)**	0 (0)	1 (1)	0 (0)	—
	Outlier	0 (0)	0 (0)	0 (0)	22 (2)	**327 (174)**	**380 (178)**	—
	Correct Pattern	**99% (87%)**	**51% (51%)**	**34% (34%)**	**82% (76%)**	**82% (44%)**	**73% (34%)**	**70% (54%)**
	Correct+Reasonable Pattern	**99% (87%)**	**66% (66%)**	**45% (45%)**	**82% (76%)**	**82% (44%)**	**87% (48%)**	**77% (61%)**

CHAPTER 5

REUSING INTERACTIVE ANALYSIS WORKFLOWS

In this chapter, we introduce techniques to achieve goals G3 and G4. To achieve G3 — curating and reusing interactions — we discuss techniques to leverage the semantically meaningful provenance based on techniques from Chapter 4 to automatically reapply the interactions to an updated dataset. We also show techniques to curate the interaction provenance into reusable workflows. For goal G4, using the interactions in a different analysis environment, we contribute techniques and a Python library to load the workflows captured in our prototype visualization tool in a Jupyter notebook and apply the workflow to a pandas dataframe. Figure 5.1 shows the contribution of this chapter toward this dissertation. The chapter is based on our previously published work [3]. K. Gadhave, Z. Cutler, and A. Lex. "Reusing interactive analysis workflows," *Comput. Graph. Forum*, vol. 41, no. 3, Jun. 2022, pp. 133–144.

5.1 Motivation and Overview

Data visualization enables analysts to leverage the powerful human visual system to identify patterns and draw conclusions. When data visualizations are made interactive with selections and data transformations such as filters, labels, and aggregation, they can be used for various analysis, cleanup, and processing tasks. However, one significant drawback of interactive visual data analysis is that analysis processes remain ad hoc. The analysis must be redone when a dataset is updated or changed. Updating datasets, however, is very common. For example, businesses regularly add new sales data, and scientists expand or correct their datasets as errors are discovered or new samples come in. Economists get updated data about various countries' indicators yearly. Data visualization tools typically cannot reapply actions like filters to new dataset versions. This lack of reusability in an interactive visual analysis contrasts sharply with computational analysis

workflows: A function that filters a dataset based on parameters can be reapplied to an updated dataset. This application has the usual drawbacks of computational approaches: Analysts should know how to program, are hard to write, and cannot leverage the benefits of graphical perception.

This chapter proposes methods to capture and reuse workflows in an interactive visualization system. Our workflows are based on interactions made in interactive data visualizations, such as choosing data dimensions and selecting, filtering, labeling, categorizing, or aggregating items. We introduce methods to capture these workflows semantically, making them robust to changes in the datasets, as shown in Figure 5.2.

When reapplying a workflow, human review and potential updates are required to ensure that the actions in a workflow still achieve the analysis goal. To address this, we introduce a review interface that visualizes changes in the dataset and consequences of the actions in a workflow, enabling corrections if necessary.

Finally, in addition to making workflows available for reuse within our interactive prototype, we expose the workflows so they can be used directly in code. This approach allows bridging between interactive visualization systems and scripted data analysis processes. For example, an analyst could do some preprocessing in a Jupyter notebook, launch an interactive visualization system from the notebook to execute a series of complex selections and data transformations that are more easily achieved in a visualization system, and then return to the notebook to apply, e.g., an algorithm to the transformed dataset. Because we now have reusable visualization workflows, various parts of such an analysis can be reapplied to an updated dataset.

We demonstrate these capabilities in a prototype visualization system that captures interaction provenance, from which analysts can extract workflows. These workflows can be reapplied to updating or changing datasets with some examples. We also introduce a Python library to bridge between the visualization system and Python code and provide examples for these workflows.

In summary, our contribution is a method to capture workflows in interactive visualization systems that can be reapplied to a new dataset. To ensure the accuracy of the reapply process, we introduce visualizations of changes and review and update capabilities for these workflows. Finally, we introduce methods to use these workflows as

part of computational workflows. We believe that our methods will make it possible to use interactive visualizations even for analyzing datasets that are being updated. We also push the limits of possible integration between computational and interactive workflows, enabling analysts to leverage the best tool for each part of a job.

5.2 Capturing and Reusing Workflows

We propose an approach by which analysts curate their workflow from the provenance of their analysis sessions rather than explicitly modeling the analysis workflow either in a graphical workflow editor or in code. This way, analysts can freely work and explore until they have achieved a result they think is worth saving in a workflow. Only on reaching an interesting state can they extract the relevant steps to preserve it as a workflow, thereby minimizing the required overhead to the analyst and encouraging open exploration. In the rest of this section, we describe how we can capture provenance in a reusable manner and use it to curate reusable workflows.

5.2.1 Capturing Interaction Provenance

Usually, capturing analytic provenance involves tracking the low-level mouse, pointer, or keyboard events by the analyst. Such events are well-suited for implementing features like undo/redo or logging the analysts' activity. However, such low-level events lack the information necessary to recreate the interaction. We propose capturing the interactions as an abstraction rather than as events. The abstraction captures all the information that is necessary to recreate the interaction. In our specification, we capture the interactions that are relevant for data manipulation, namely *view specification*, *selections*, and various types of *data transformations* [26].

View specification interactions are concerned with choosing the subset of dimensions. An example is to show two dataset dimensions in a scatterplot: Here, the view specification entails both the choice of dimensions and the choice of visualization technique.

Selections are a basic but essential interaction available in visualizations. Selections cannot only be used to highlight items of interest but also form the basis for further data transformation on selected subsets of the data. For our specification, we break down types of selections by the level of semantics they capture:

The simplest form of selection is *ID-based selection*, which directly stores the IDs of the selected items. ID-based selection has the lowest level of semantics and is the least useful when reusing a selection. When items are added to a new dataset version, they are not considered, even if they fall into a selected pattern.

Next, we specify the *range selection* that stores ranges over dimensions, capturing a set of rules for a selection similar to, e.g., an SQL query predicate. Range selections are usually specified using rectangular brushes [30], [33] in scatterplots or a series of brushes along an axis in parallel coordinates. They are reusable for updates in the data as long as the updates happen within the extent of the range selection. For example, if an updated dataset version has three new points within a rectangular brush area, these points will be selected automatically.

The next level of selection is *semantic selection* as introduced in our previous work [2]. This approach captures higher level semantics behind the apparent selection when visualizing data. A pattern-based selection recognizes, for example, that an analyst selected all outliers or a cluster centered at a specific location. By selecting based on higher level patterns in the data, such selections are robust to changes in the data. For example, when outliers are selected in a dataset, similar outliers can be selected in an updated dataset, even if the outliers appear in new locations. We could also use other approaches, such as query relaxation proposed by Heer et al. [34]. Semantic selection aims to capture complex selection patterns rather than a list of IDs or simple rules. Such selections are the most reusable, as they are robust to fairly complex updates to the dataset.

Our specification supports four **data transformation** actions: *Filter* actions track whether certain items are filtered in or filtered out of the dataset. Filters help focus on a select group of items by filtering them in or removing irrelevant data items by filtering them out. *Labeling* sets of items in a dataset help annotate or tag items with metadata or observations. *Categorize* actions are used to classify items or assign them to categories from a set of dynamically defined categories. Usually, each point is assigned to a unique category. Categorization helps divide the dataset into subsets that can be compared or used separately in subsequent analysis steps. Deriving new data items by *aggregating* groups of existing ones is an important data transformation, as is evident from the popularity of pivot tables in Excel. An aggregate item can replace the items it was derived from,

simplifying the data. Aggregation also allows analysts to compare groups with shared characteristics effectively. Aggregation is done by grouping multiple data items into a new item. Attributes are aggregated based on a mathematical function (often called an "apply function in programming libraries). These functions usually summarize multiple values into a single representative value like the *mean*, *median*, *sum*, *min*, or *max* of the item's attribute, but custom functions are also common. For example, it might be helpful to aggregate all European countries into a single "Europe" item and compare it to an "Asia" item in a dataset on metrics about countries. For this aggregation, we have to define "the population" of Europe as the sum of the population for all the countries in it. However, a "life expectancy" column would need a different, more sophisticated function for meaningful aggregation.

We have chosen these interactions as they allow us to demonstrate our approach. Yet, other operations, such as deriving attributes or manipulation, such as moving around items, could be supported by our methods. Together, these view specification, selection, and data transformation actions make up a robust set of tools that benefit enormously from being available in an interactive visualization interface and are commonly used in data science tasks.

5.2.2 Capturing Analysis Process

We propose capturing the analysis session in a provenance graph where each node represents an interaction. A provenance graph records the interactions in the sequence they were performed, allowing us to infer the dependency of the interactions on each other. Provenance graphs are typically directed acyclic graphs. Downstream interactions can depend only on upstream interactions. Data transformations are derived from selections to categorize a group of points; for example, they are first selected and then assigned to a category. The robustness of the transformation depends on the type of selections used, as described previously. For example, if an analyst selects 15 items individually and assigns them to a category, and the category is associated with just those items. However, if the items were selected using a range selection, the category is associated with the range rather than individual items. When the dataset is updated, items appearing in the region of the range selection are categorized as well. Using semantic selections further improves

the robustness of subsequent actions. An analyst can use the pattern-based selection to refine the initial selection of 15 items as a cluster. The category is then associated with the cluster rather than the original selection. Updating the dataset by adding or removing items automatically updates the categorization as well. If the groups of items were to move, the semantic selection would still accurately track the items, in contrast to a range selection, which would lose items that move outside the range.

Analysis sessions are rarely linear and usually involve trial and error. An analyst can make multiple attempts by going back a few steps in the analysis provenance by undoing actions and starting a different analysis flow. Alternatively, an analyst might clear all their current interactions and return to an initial state before trying something else. A graph representation allows us to capture the former type of iterations as parallel branches in the provenance, whereas the latter are in a sequence.

An analyst can also add metadata to each node in the provenance graph. The metadata can be in the form of annotations, where the analyst tries to externalize their knowledge, such as assumptions about the data, known errors in the data, etc., which are typically impossible to capture automatically. The analyst can also bookmark specific nodes as points of interest in large analysis provenances, especially ones with lengthy analysis sequences and multiple branching analyses. Using annotations and bookmarks simplifies extracting workflow after an analysis goal is achieved.

Capturing the abstract interactions enables editing of the analysis session to curate workflows and reuse the sessions on the same dataset or, more interestingly, on an updated dataset. We capture the analysis process as an abstraction of the interactions, which makes the captured analysis agnostic to the environment.

5.2.3 Curating Workflows

As we discussed in the previous section, the captured analysis provenance can be cluttered by the iterative nature of the analysis process. Hence, extracting and cleaning a particular analysis branch or a subsequence of the analysis as a reusable workflow is desirable.

We previously introduced different levels of selections that can be captured. These different types of selections can — and commonly are — combined. An analyst can start

with a rough selection of points either by specifying the points directly or specifying a range. They can modify this selection and add or remove points to it. They can then decide to use semantic selection to refine their existing selection. A provenance graph will be populated with multiple nodes as the analyst iterates over different points before settling on a pattern. While curating the workflow, an analyst can easily remove extra selection attempts, keeping only the most informative selection to drive downstream interactions (see 5.2), creating a succinct reusable workflow.

It is also possible to automatically prune workflows to remove actions that do not impact the eventual result. For example, when a "clear selection" action is detected as part of a filter sequence, all previous selections and the clear selection operation could be automatically removed.

5.2.4 Reusing Workflows

Capturing the analysis sessions and workflows in a reusable manner makes it easy for an analyst to rerun the analysis on the same dataset (for reproducibility) and, more importantly, apply the analysis to an updated version of the dataset (for reusability).

Tabular datasets can change in limited ways: Attributes associated with rows can change, and rows can be added or removed. Also, dimensions or rows (items) can be added, removed, or reordered [123]. For our purposes, we limit ourselves to changes to existing attributes, adding or removing items, and updating attribute values, as order is relevant only for certain representations, and adding or removing dimensions is beyond our work's scope.

We have different approaches to reapply selections, which are straightforward for IDs and ranges. The method for reapplying semantically captured selections varies by the type of selection. If we capture pattern-based intent [2], we can rerun the appropriate algorithm and captured parameters to recreate the selection. Of these three types of selections, the ID-based selections are the least useful for reusing a workflow on an updated dataset. Range-based selection performs better but fails to capture dataset updates outside the range. Semantic selections have the potential to be robust to updates in the dataset.

However, even with semantic selections, automatically reapplying the analysis session or a workflow to an updated dataset might not always make sense. The dataset changes

might be so drastic that a previous analysis session is no longer appropriate. Hence, it is essential to allow analysts to review the captured sessions for different versions of the dataset and mark whether different interactions are valid for those versions.

5.2.5 Bridging Between Environments

The environment-agnostic nature of our approach allows us to integrate workflows with computational analysis systems and re-execute a workflow on an updated dataset just like a function. The abstract representation of the analysis can be expressed in any commonly used data interchange formats like JSON, YAML, etc. We can build visual analysis tools that can capture the analysis sessions in one of the formats and companion libraries to use the captured analysis sessions in other environments like notebooks.

5.3 Reusing Workflows Prototype

To demonstrate the feasibility of our approach for capturing and reapplying workflows, we developed prototype tooling to demonstrate all aspects of the approach. In an interactive visualization tool, we demonstrate how to capture the analysis process and reapply to updated datasets in the same tool. The prototype is available at https://reapply-workflows.github.io/reapply-workflows/. We also implemented a Python library that can load captured workflows and be used in computational environments like Jupyter notebooks.

5.3.1 Interactive Visualization Tool

The interactive visualization tool allows analysts to create projects and upload different dataset versions. Analysts may select dimensions of the tabular dataset to visualize in one or multiple **scatterplot(s)** (Figure 5.3(a)). We provide rectangular and free-form "paint-brushes" of three sizes for selection. Analysts may also add a **parallel coordinate plot** to visualize multiple dimensions at once, which supports brushing on the axis.

We use the Trrack library [1], which we previously developed and stored the actions in a directed acyclic provenance graph to capture the analyst interactions. Each node in the provenance graph is the abstract representation of a corresponding interaction the analyst makes. We visualize the provenance graph in a tree-like layout (5.3(c)), where each action is described and can be annotated by the user. Analysts can go back in the provenance

graph to a previous step and start a new branch, which supports the iterative nature of the visual analysis process.

We use techniques from our previous work [2] to capture semantically rich pattern-based intents. Our prototype monitors user selections in any plot and compares them to a large set of patterns computed for a given dataset using various algorithms and parameterizations. The different patterns are ranked based on the Jaccard Similarity between the selection and the prediction, as shown in Figure 5.3(b). In Figure 5.3, we see a rectangular selection partially covering a cluster, and the system ranks a clustering pattern as a good match. When an analyst hovers over the cluster prediction, the extent of the cluster is shown as a polygon, and the items that are not part of the selection are highlighted. When an analyst chooses to confirm this prediction as the intended pattern, our system stores the details of this pattern.

We predict five patterns: Clusters, inliers and outliers, correlations, multivariate optimization, and ranges. For each pattern, we store the information necessary to recreate the pattern in an updated dataset. For example, for clustering, we store which algorithm was used (e.g., KMenas or DBScan) with which parameters and attributes about the selected cluster, such as its centroid. Figure 5.3 shows an example where an analyst first added a plot of a dataset that exhibits clusters. The analyst then continued with a crude rectangular selection. The system recommends a cluster as a match in the prediction interface. Hovering over that cluster prediction reveals the cluster's properties. The analyst decides this is match is good for the intended selection and confirms the prediction. Finally, the analyst filters out the selected items. Each of these steps is then reflected in the provenance graph. The analyst could now go on and continue with subsequent analysis steps and only create a robust workflow based on these steps at a later time.

5.3.1.1 Comparing and Updating Datasets

Our systems explicitly visualize differences between different dataset versions and how a current analysis would be applied to different versions. Figure 5.4 compares the dataset shown in Figure 5.3 and an updated dataset and a subsequent successful application of a cluster-based selection. We see in Figure 5.4(a) how the dataset changed compared to the one in Figure 5.3. Figure 5.4(b) shows how the selected cluster changed between the

datasets; the hulls of both clusters are shown. Figure 5.4(c) shows the cluster selection in the updated dataset. If the initial selection had been captured using range selection, i.e., not using a semantic pattern, the items that shift outside the range would not have been captured.

Figure 5.5 shows another version of the dataset, where the selected and subsequently filtered cluster broke apart into two clusters. Depending on the analyst's intent, several options are plausible: Remove both clusters, remove only the top or bottom cluster, or remove none of the clusters. An automatic determination is impossible here, as the right action depends on the analyst's higher level intent. Hence, the analyst has to review this situation and decide how to proceed, as illustrated in Figure 5.5. By default, the system assumes that one large cluster is the best match (Figure 5.5(a)), as it best corresponds to the previously selected cluster overall. Assuming that the analyst intends to select only the top, smaller cluster, they can reject the "Clusters Selection" node in the provenance graph and replace it with a better matching cluster, as shown in Figure 5.5(b).

5.3.1.2 Curating Workflows

Finally, the analysts can curate a reusable workflow based on the provenance data (see Figure 5.6). After switching to the appropriate analysis branch, the analyst can open the workflow editor and create a new workflow from the current branch. Doing so loads the current branch as an editable workflow. The analyst can remove individual nodes, name the workflow, and sync it to the workflow database. While theoretically possible, our current implementation does not support automatic pruning. The workflows stored in the workflow database are available for reuse in the tool and the Python library (see Section 5.3.2).

5.3.2 Bridging to Computational Notebooks

At the heart of our suite of tools is the **Reapply Workflows** library that performs all the predictions and the matching of actions between updated datasets. The library is used in our prototype tool but can be used by third parties, e.g., to load workflows in notebooks. The piece that connects the visualization tool and the computational environment is a **workflow database**. As discussed, an analyst can use the visualization tool to perform a visual analysis and capture workflows. Whenever a workflow is created or modified, it

is also stored in the workflow database. The library can then load the workflows from the database to a computational environment, such as a Jupyter notebook.

Figure 5.7 shows the process of loading and using a workflow in a notebook. The library interfaces with the workflow database to provide convenient access, printing descriptions, and an inspection of the steps in a workflow. Ultimately, workflows can be applied to a pandas dataframe. The output of applying the workflow depends on the actions in the workflow. If the workflow results in a selection, the output dataframe has an extra boolean column *isSelected* that denotes the selected items. More complex workflows with data transformation modify the output dataframe, e.g., filters return a subset of the original dataset, labeling and categorization operations add an extra column with the relevant label or category assignment, and aggregation workflows add a new row to the dataset with the aggregated values.

5.3.3 Implementation

The prototype is a web based application developed with React and TypeScript. The Flask server's backend leverages the Reapply workflow library for computations and workflow related features. The library is developed in Python and uses scikit-learn to run the prediction algorithms. The library is available on the TestPyPi package index by the name *reapply-workflows*. We show our computational demos in Google Colab notebooks, which are equivalent to Jupyter notebooks but can be collaboratively edited and hosted by Google.

We store datasets in an SQL database. Different dataset versions are tracked with a separate *record* table. The changes between the datasets are computed on the fly; hence, no additional storage is needed to track the diffs. We pre-compute patterns the prediction system uses for pairs of dimensions to help speed up the initial predictions. We switch to on-the-fly computations when any data transformation changes the dataset. The analysis sessions and the curated workflows are stored in the Google Firebase Realtime database as JSON.

5.4 Comparison with Alternative Approaches

This section compares our approach with Tableau Prep, B2 [19] and VisFlow [44]. All these systems have in common that they capture workflows, yet they all use different approaches.

Tableau Prep and VisFlow use **explicit modeling** of workflows. Tableau Prep provides a graphical workflow editor where analysts can drag and drop nodes as workflow steps. The visualizations in Tableau Prep are limited to distributions, although data can be imported into Tableau subsequently, which limits the possibility of open exploration before workflow generation. VisFlow is a graphical workflow editor similar to Tableau Prep but supports adding visualizations as a part of workflow nodes. VisFlow and Tableau Prep do not support exporting the generated workflows outside their environments. Tableau Prep and VisFlow workflows do not support semantic interactions and rely on rules for selecting the data. Further, both tools include the dataset in their workflow, making reapplying workflows difficult. A downside of such explicit workflow modeling systems is that they are more akin to graphical programming than to open exploration and refinement of datasets, which comes with the usual burdens of programming: High complexity and a steep learning curve.

B2 [19] is a Jupyter extension that aims to bridge the gap between interactive visualizations and computational environments. The strength of the B2 approach is that it integrates the interactive visualizations directly in the notebook and provides tight coupling between code blocks and the visualizations. B2 currently supports selections; any further data transforms require coding. Further, the selections are limited to brushing and do not capture the semantics behind the selection. B2 supports limited provenance tracking for interactive selections by generating code snippets to reflect the visual selection. The code snippets are time-stamped to keep track of the order. Older snippets are automatically commented out, which adds clutter to the code blocks. The lack of explicit tracking makes maintaining parallel data analysis approaches difficult. Combining our approach of semantically capturing selections and provenance tracking with B2's tight integration of code and interactive visualization would be possible.

5.5 Validation

We use three strategies to validate our approach: Usage scenarios, demonstrating the usefulness of our techniques in a realistic scenario (see the following section); synthetic datasets to demonstrate the robustness of reusing analysis workflows; and interviews with professional data analysts (Section 5.5.2).

5.5.1 Usage Scenario: Outlier Countries for COVID-19

We analyze outlier countries concerning COVID-19 metrics. The dataset [124] includes various COVID-19-related metrics for multiple countries worldwide. COVID-19 data attributes change frequently and are a good way to demonstrate our approach since selections and conclusions must be robust to updates in data. Let us look at a scenario for which we want to investigate countries with an aberrant trend in the number of new cases and deaths related to COVID-19. We start with data for January 2021 and load a scatterplot for **new monthly cases versus new monthly deaths**. We immediately see that many countries are far from the cluster of countries close to the origin. We then select a few of these countries using a paintbrush selection. The system computes predictions and suggests an outlier-based selection (see Figure 5.8(a)). We use this suggestion to refine our selection. We switch to different months of the dataset to see if the selection is applied correctly. We are happy with the selection, so we filter out everything but these items to focus on these outliers.

We then categorize the outliers. We select all the countries with high monthly cases and high monthly deaths with a rectangular brush and categorize them as countries with "High Deaths–High Cases." We then select countries with low monthly cases but high monthly dates and categorize them as countries with "High Deaths–Low Cases" and proceed to "Low Death–High Cases" (Figure 5.8(b)). We switch to different datasets and verify that the categorization is applied correctly. When we are satisfied with the result, we approve the interactions in the provenance history. Figure 5.8(c) shows an extreme example, June 2021, where cases and deaths in most countries are much lower, clustering close to the chart's origin — applying the categories (Figure 5.8(d)) results in several outliers being unassigned, hinting at the fact that even moderate COVID activity is an outlier in that version of the dataset.

After curating the provenance history in a **Categorize Outliers** workflow, we store it in the workflow database. We then move to a Jupyter notebook to load this workflow and analyze these newly categorized countries. We can create a histogram of the categorized countries stacked by region to get an idea about how different regions were affected by COVID-19; where we see that high deaths have shifted to South American countries in June, which were barely affected in January and that South American countries are predominantly in the High-Deaths–Low-Cases category.

5.5.2 Feedback Session with Data Practitioners

We evaluated our method through interviews with four data practitioners from different domains — nursing (P1), public health (P2), surgery (P3), and chemical engineering (P4) — who regularly do data analysis. We first interviewed them about their current data analysis process. We then introduced the participants to the principles of our technique and gave them a live demo of the prototype tool and the Colab notebook, after which the participants were asked to give feedback on the techniques and speculate how they could be applied in their work. We have analyzed the transcript of the interviews and grouped the responses into themes, which we describe below. The interview questions and transcripts of the interviews are available as supplementary material at https://osf.io/djb8p/.

5.5.2.1 Provenance Tracking

Our participants used different tools to analyze their data, but all participants reported that they frequently explore alternative analysis approaches. Participants who use scripts report that they use comments and code blocks to keep track of the different analyses they do. One participant who used Tableau mentioned the limited utility of the undo/redo stack in keeping track of diversions in the analysis process. The participants particularly liked the provenance-tracking approach we demonstrated in our prototype tool. P1 said that "I do like the, the branching piece from here because it's visually a lot easier than to click the back button, or forward button, because it also is telling me a little label of what changed with that." and P3 said that "I definitely liked the way it branches. I think that's a super cool aspect of it. And then being able to kind of settle on one sort of branch analysis I'd be able to explore like, that is really powerful."

5.5.2.2 Capturing Semantics

Our participants expressed varying sentiments about the semantic selection approach [2] we leverage in this work. P3 said that "Yeah, I think that's super smart. . . . they're saying, here's what I think is a cluster, and then the program is saying, 'okay, looks like this is what you're trying to define. Is that correct? That would be really helpful." Although all participants acknowledged the usefulness of semantic selections, two participants, who have a strong statistics background, mentioned their hesitation in relying on the predictions without detailed information about the algorithms and the parameters used in the prediction — information our system captures but does not currently provide easy access to. P2 said that ". . . I wanted to understand what was happening in the background." P4, whose data analysis relies on the segmentation of microscopy images, wanted the ability to add domain-specific models to the prediction system.

5.5.2.3 Visual Data Wrangling

Participants expressed interest in using visualization to directly interact with the data for selection, creating groups, and labeling. P1, for example, said that ". . . you do regrouping in Python, but then you always forget your variable name. And then you're always looking through your data frame for what is it type of thing, where Tableau, it's easy to do the groupings, and it just kind of makes a new variable right underneath it."

5.5.2.4 Workflows

All participants agreed that reusable workflows would be useful in their analysis. P4, for example, said "I definitely think it will be applicable because most of the time, we actually don't inherently change the method itself. . . . So I definitely can see this to be helpful." When asked whether they would like to create workflows or curate workflows after an analysis explicitly, the participants noted that their data analysis sessions often start with an open-ended exploration of data to detect interesting patterns. P3 described their analysis process as "definitely more exploratory." Participants liked the ability to curate their workflows from an existing analysis session: "I like this, because it's much more natural" (P1).

5.5.2.5 Bridging Between Tools

Participants were excited about the potential of using the workflows as a bridge between their different tools. P1, who switches between different tools frequently, mentioned the need to repeat certain steps as they switch: "...replicating essentially a lot of the filters and the sorting ...". The current approach of this participant usually involves modifying the data in one tool and then loading the modified data in a different tool. On demonstrating the use of workflows in the Python notebook, P1 said, "great to be able to click on the Select 53 points, and then see the all code, you know, to that would do the 53 points."

5.5.2.6 Collaboration with Domain Experts

P2 and P3 worked with clinicians and were interested in the applicability of our technique as a means of collaboration. They work in R and SAS heavily, whereas their clinical collaborators have no familiarity with scripting tools. P3 said that "I work with a lot of clinicians, a lot of doctors, and they are interested in research, but they don't have much research background, right. And this would be something that I think would be really, really beneficial for them, because they are going to be, they're going to want to do a lot more kind of looking at the data and sort of touching the data, and it's going to be really important that they have that log where they can come back, give me data to just sort of reflect on." and "If they were able to show me their workflow, and I was able to go through and see, how it progressed, I think that would be really helpful."

Overall, our participants expressed positive sentiments regarding our techniques to capture reusable workflows and thought they would be helpful in their current analysis environment. P1 said that "I think it's great. I would love to see how you would implement this in Tableau". They were most excited about the provenance tracking and the ability to bridge between interactive visual analysis tools and computational environments.

5.6 Discussion

In this section we will discuss some implications of our techniques and directions for future research.

5.6.1 Generalization to Other Visualization Techniques

Our technique is based on capturing interaction provenance as an abstraction that contains the information required to recreate the interaction (as opposed to a stream of mouse/keyboard events). We demonstrate the abstraction for common interactions such as selections, filtering, labeling, categorizing, and aggregation. Our methods are transferable between visualization techniques if they support equivalent interactions and datatypes. For example, to add a parallel coordinates plot, we did not have to modify our library developed initially with scatterplots. The interactions we describe are meant to be examples. Other types (e.g., selecting a neighborhood in a network or sorting a table) can be included if captured in the provenance graph, and the library is extended to handle the type of operations and data. Our current implementation and our choice of algorithms are specific to tabular data. However, our general approach applies to network, image, or volumetric data, provided we can identify suitable methods for robust selections.

5.6.2 Certainty of Fit for Reuse

When we apply a selection to a new dataset, we assume an analyst will review the updated selection. Although a review is certainly necessary if the data changes significantly, minor changes might not require a manual review. We could conceivably compute metrics about how "sure" we are about a specific operation as it is applied to a new dataset. If, for example, all points are in the same selection and have moved little, we might not need a review. If, in contrast, the dataset has changed significantly and the selection is affected, we could print a warning, emphasizing the need for a review.

5.6.3 Interaction Directly in Notebooks

Visualization libraries such as Altair [16] and B2 [19] have made interactive selections in visualizations within a Jupyter notebook possible. Papers like B2 [19] explore this approach thoroughly. Conceptually, our technique can support actions in embedded visualizations; hence, we plan on extending our library so that selections made within a notebook can also be autocompleted and extracted into a workflow. Although we expect that other aspects, such as compound actions and workflow review, are infeasible to integrate natively within a notebook, robust, pattern-based selections would enhance an analyst's ability to leverage the interactive capabilities of such simple visualizations. Directly integrating the

interactive visualizations in the notebooks would reduce the friction of switching between multiple environments while limiting the complexity of suitable visualization approaches.

5.6.4 Alternate Ranking Strategies

Our prototype uses the Jaccard similarity to rank different predictions. The Jaccard similarity is sensitive to the data size and can be distorted if a large variation exists between the sets being compared. Our technique can be extended to support alternative rankings, and we can potentially add a custom ranking approach tailored to a dataset type. For example, we can modify the Jaccard metric by adding a regularizing parameter based on the size of the dataset to reduce the penalty for uneven set sizes.

5.7 Conclusion

We have introduced a method to capture interactive actions taken in visualization semantically meaningfully and reuse sequences of actions (workflows) on updated datasets. In this way, we make actions taken in a visualization just as robust to changes as if they were implemented in a function in code. We introduce methods that match up selections between updated datasets that go beyond just reapplying a simple rule, instead leveraging various pattern-detection algorithms and knowledge about the properties of a prior selection. We introduce a mechanism to review changes and update workflows if necessary. Finally, we have demonstrated that this approach also allows us to bridge between an interactive visualization system and a computational workflow.

Whereas robust workflows could be implemented in code or using graphical workflow modeling tools, our approach is easier to execute and allows for an unencumbered exploration process. Our prototype and examples show that our approach works for a range of patterns and datasets that change in significant ways. We believe our approach could also be transferred to many other types of data and visualizations.

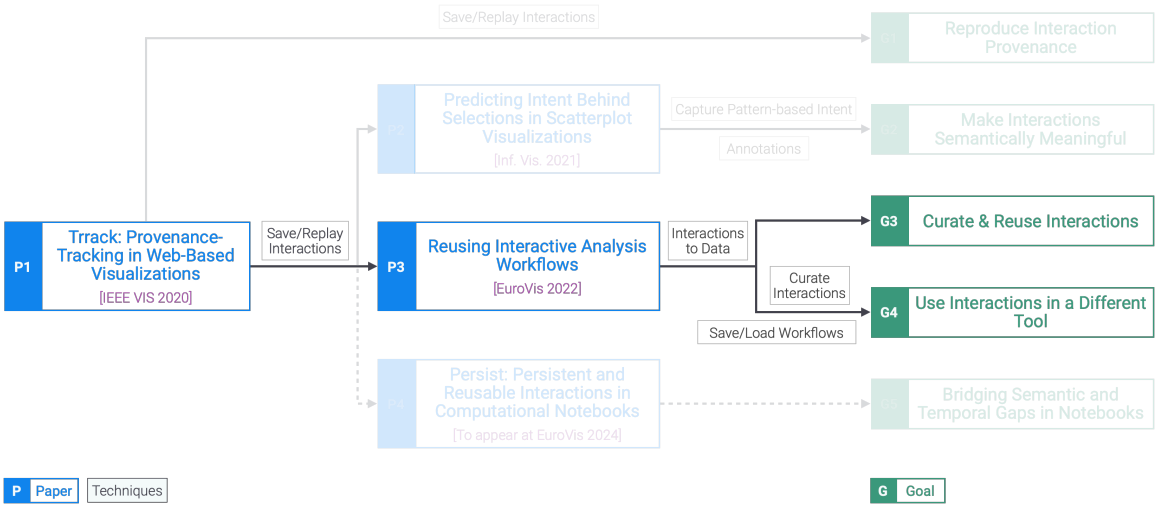

Figure 5.1: Techniques for reusing the interactions. We can leverage the pattern-based intent captured in the interaction provenance to map interactions to data operations and algorithms to automatically apply the pattern on an updated dataset. The pattern-based interactions can also replace previous interactions like manual selections, supporting the curation of the provenance. Curated interactions are saved as environment-agnostic workflows and can be loaded by a different analysis environment.

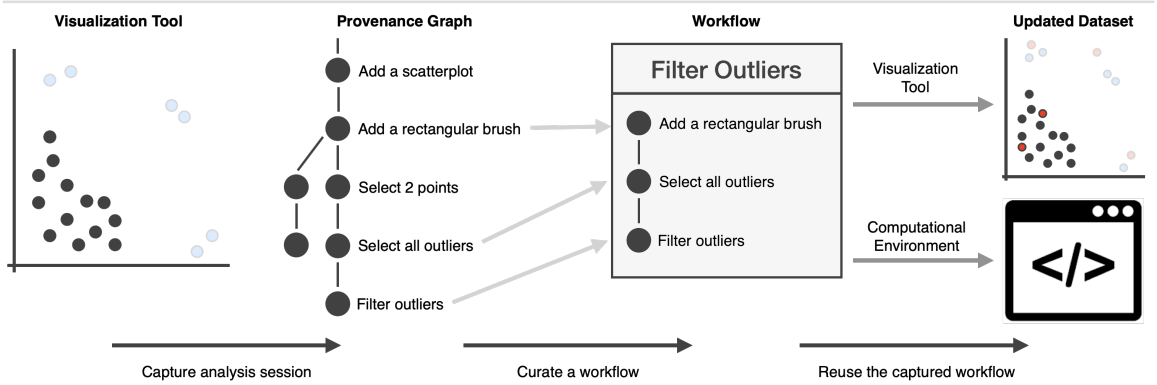

Figure 5.2: The process of reusing workflows created in interactive visualizations. Interactions, such as brushes, filters, or selections based on higher level patterns (outliers in this example), are applied to a dataset. A series of actions can be extracted into a workflow. This workflow can then be applied to an updated dataset either in an interactive visualization system or in a computational environment.

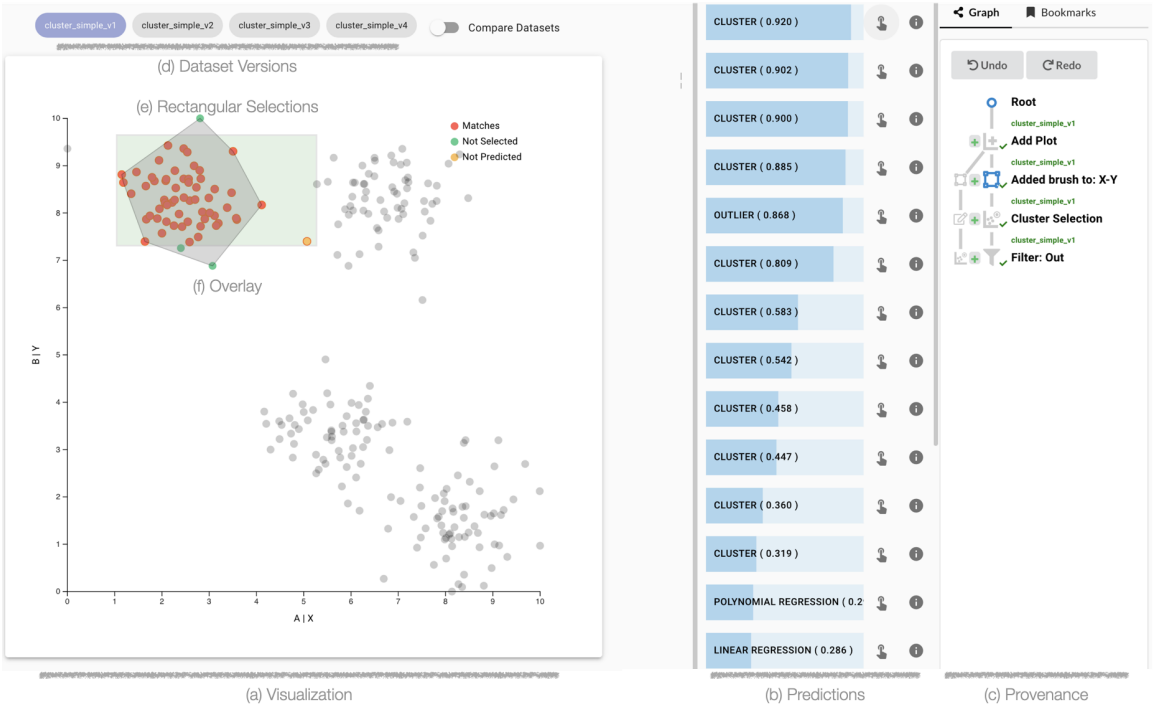

Figure 5.3: A dataset exhibiting clusters is shown in a scatterplot (a). (e) A rectangular brush selection is used to compute pattern predictions to capture the selection's semantics. (b) A ranked list of these predictions is shown to the right of the scatterplot. The analyst selects the top prediction, which is a cluster, and the system shows an overlay (f) to visualize the cluster's boundary. (c) The provenance graph on the right shows the captured interactions.

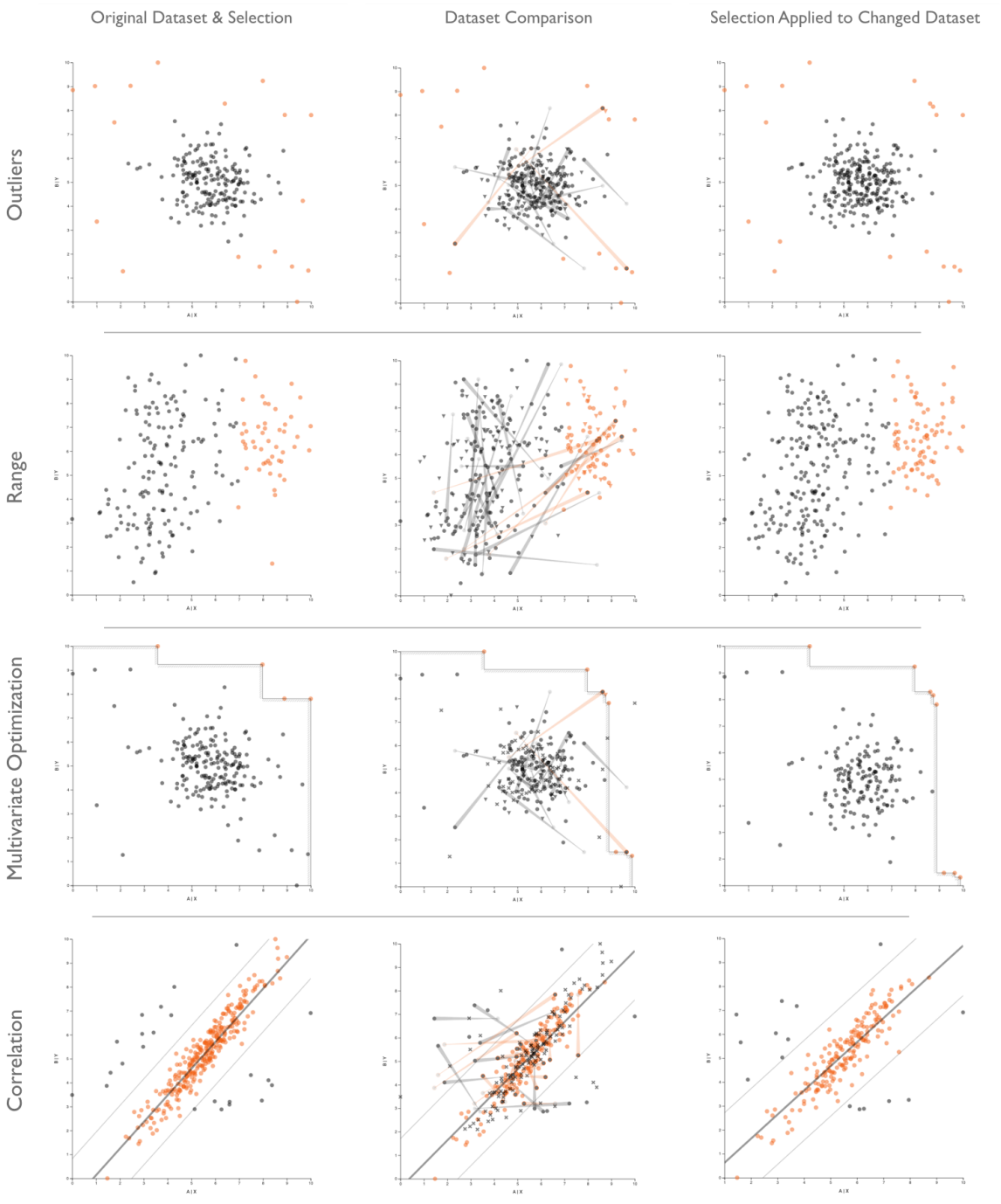

Figure 5.4: Comparing two datasets and reapplying a workflow. (a) The comparison mode explicitly shows the differences between the two selected dataset versions. The scatterplot encodes newly added items in the updated dataset as blue triangles ▼, removed items are shown as red crosses ✖, and items that have shifted positions show a comet-like trail from their original to their new position. (b) The selection made on the original dataset moves down to the new cluster and handles new and removed items correctly. (c) The updated selection.

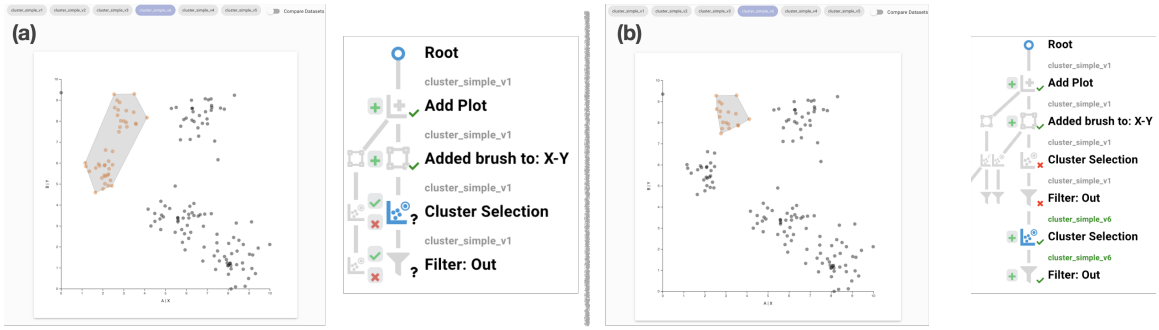

Figure 5.5: Reviewing and updating workflows applied to an updated dataset. Continuing the analysis from Figure 5.3, the analyst loaded a new dataset version where a part of a cluster broke off and moved down. (a) By default, the system considers these two clusters to be one larger cluster as the previously selected larger cluster biases the outcome toward a single cluster. The review interface indicates that certain actions have not been reviewed for this version of the dataset by showing a question mark (?) next to the node. To select just the upper cluster, the analyst first confirms the "Add Plot" and "Added Brush" actions, which are then shown as approved with a checkmark (✔). (b) The analyst then rejects the "Cluster Selection" action and picks a new cluster prediction that captures their intent.

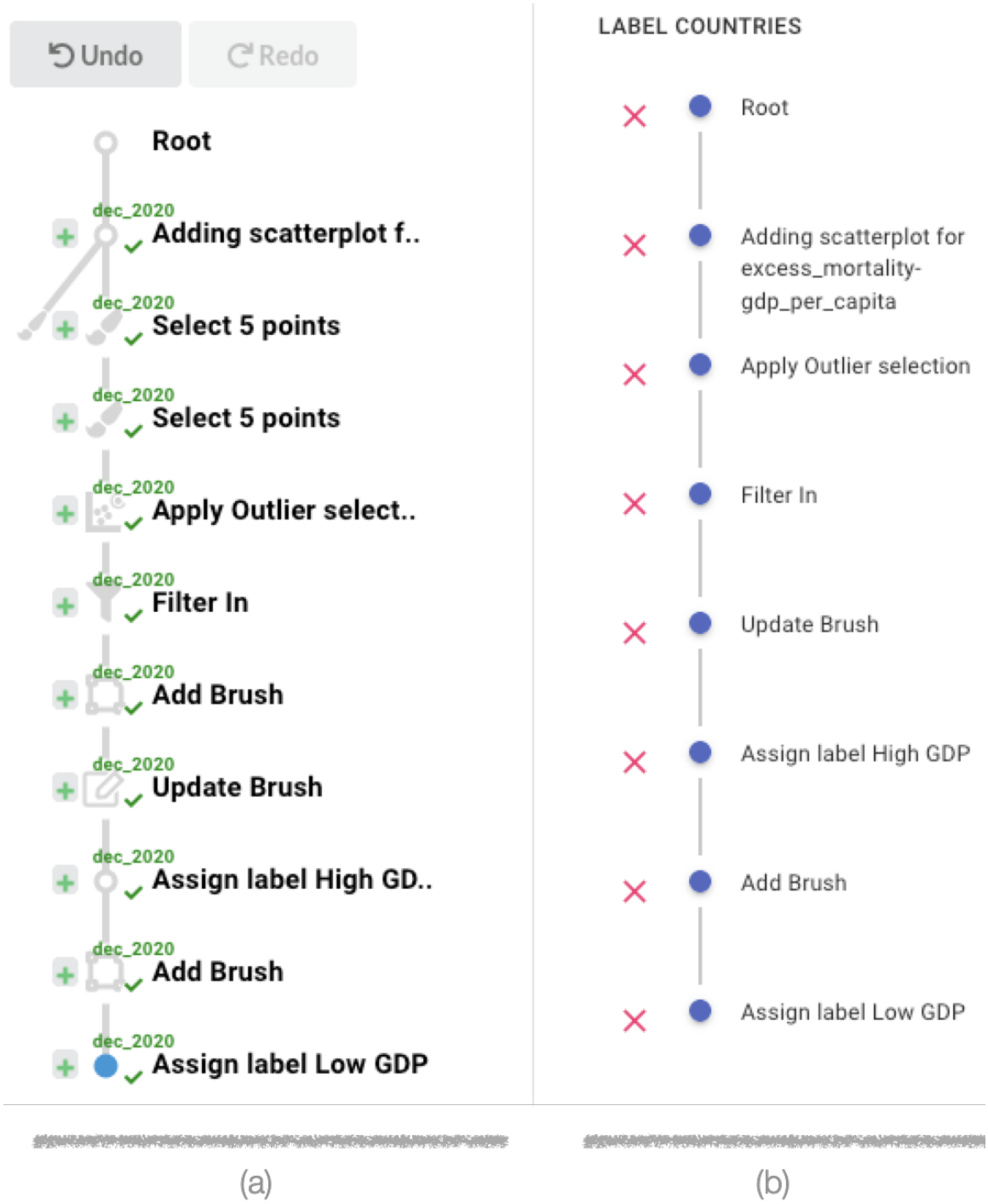

Figure 5.6: The workflow editor. An analyst can create a new workflow from the interaction history captured in (a), the provenance graph, and curate it in (b), the workflow editor interface, by removing unnecessary actions.

Figure 5.7: Executing a computational workflow defined in the visualization tool in a computational environment. (a) We first load the dataset. (b) We load the workflow library and the workflow we are interested in. We then apply the workflow to the dataset. The tool plots a preview of the actions. Note that new `isSelected` and `isFiltered` Boolean columns are introduced when a brush and Filter are added in the preview. (c) After the Filter is applied, the number of rows is reduced from 150 to 108. A visualization of the result shows the cluster was removed. Visit the notebook.

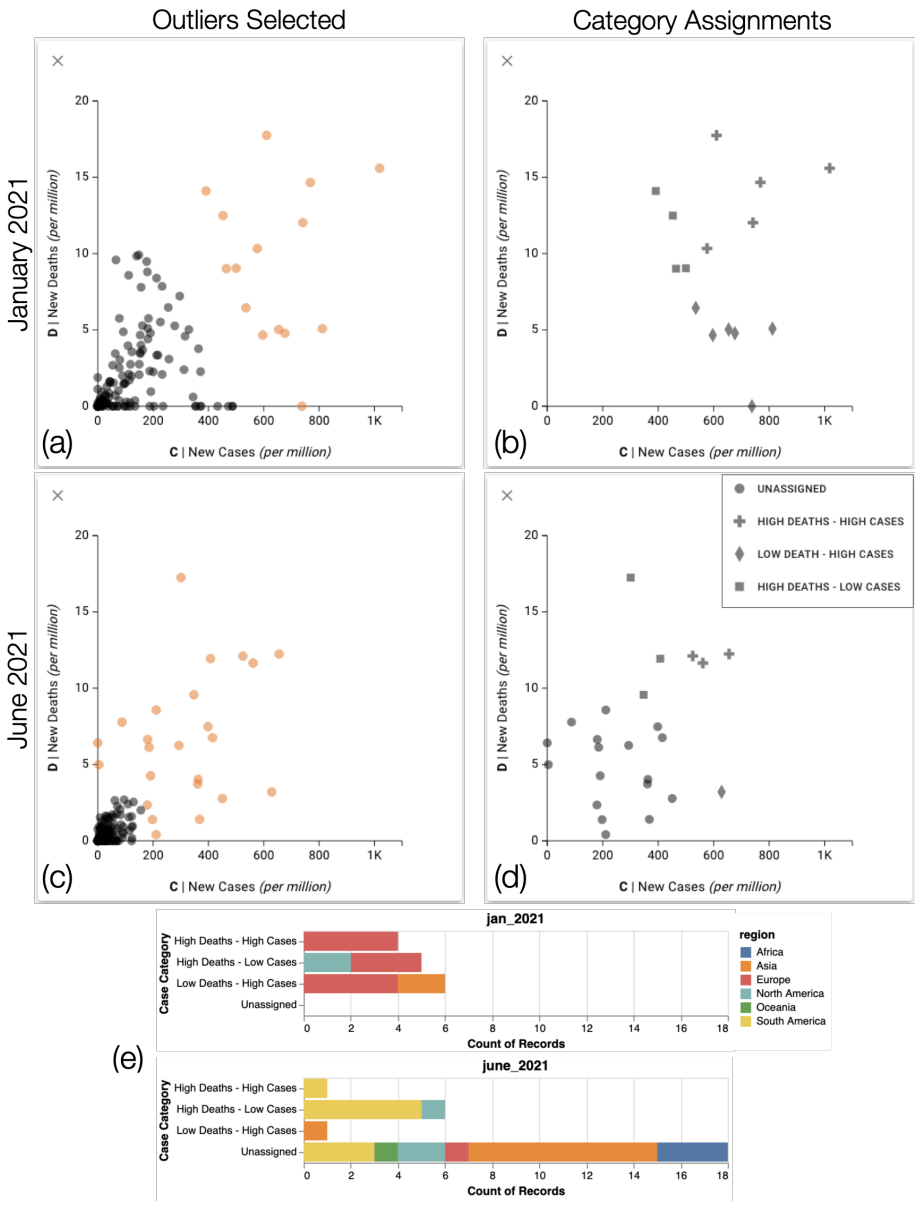

Figure 5.8: Categorization of countries in a COVID-19 dataset. (a) A scatterplot for January 2021 shows the number of new cases versus the number of deaths. We selected outliers, capturing countries with many cases, deaths, or both. (b) We filter out all countries except the selected outliers and categorize the countries. (c) We switch to an updated dataset for June 2021 and see that the number of cases and deaths has decreased worldwide, but the pattern-based selection has correctly selected the outliers. (d) The system automatically applies the range-based categorization to the countries that fall within the previous ranges. (e) A subsequent analysis in a notebook reveals that the worst of the pandemic has shifted to South America. Visit the interactive figure and the notebook.

CHAPTER 6

PERSIST: PERSISTENT AND REUSABLE INTERACTIONS IN COMPUTATIONAL NOTEBOOKS

In this chapter, we introduce techniques to achieve goal G5 — bridging the gap between code and interactions in the computational notebook. Computational notebooks like Jupyter exhibit a temporal and semantic gap. The temporal gap arises from the transient interactions in the notebook outputs and the persistence of changes in code cells. A lack of bidirectional communication between code and interactions causes the semantic gap. Code cells can generate interactive visualizations, but the results of the interactive visualizations are not accessible in the code. Figure 6.1 shows the contribution of this chapter towards the dissertation. The chapter is based on our work [21].

6.1 Motivation and Overview

Computational notebooks allow for narrative data analysis combining code, data visualizations, text, figures, etc., in the spirit of literate programming proposed by Knuth [9]. 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 [90]. 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 of usability to complexity, programming lies on the complex side, yet it can be applied in generic contexts. On the other side of the spectrum are specialized interactive visualization tools. Interactive analysis tools can make advanced operations simple but lack generality: They are good at specific tasks but lack other desirable characteristics, such as broad applicability, reusability of analysis processes, reproducibility, etc.

It is unlikely that there are data analysis solutions that are simple yet equally expressive

as programming languages, yet certain data operations are much easier to achieve in interactive and visual interfaces than they are in code. We postulate that hybrid, well-integrated solutions can be a significant improvement over the current, mostly isolated state of programming versus 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 integrate 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 like Vega-Altair [16], Plotly [125], bokeh [17], and IPython widgets [18] 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 [20].

Recently, systems like B2 [19] and Mage [103] introduced solutions to address the barrier between interactions and code by generating code for interactive selections. As our primary contribution, we introduce a new principle for integrating interactions with code based on provenance data. As illustrated in Figure 5.2, Persist tracks actions taken in interactive visualizations that are specified in code and apply 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 in keeping the visualization and the code in sync. Third, excessively 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, and changing values) that can be triggered either with direct manipulation, a toolbar, or a combination thereof. We also provide a custom interactive data table that enables a range of manipulations that can be tedious to achieve in code but are natural in an interactive system. Persist is an open-source JupyterLab extension. The source code is available at https://github.com/visdesignlab/persist.

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

6.2 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 [126]. 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) versus `Vertical_inches` (the thickness of the avalanche) columns of the dataset. This visualization helps in identifying the general data distribution and potential outliers. The analyst notices that some records show elevations below 2,000 and above 15,000 ft, which are outside Utah's elevation range (2,300–13,500 ft) and concludes

that these must be erroneous entries (see Figure 6.2(a)). Using a brush, the analyst selects these points and uses the **Remove Selection** button in the Persist toolbar to remove the selected points (Figure 6.2(b)). Persist tracks all the interactions in a provenance graph (Figure 6.2(c)) 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 6.2(d)). The bar chart reveals a seasonal pattern, with avalanches peaking in February. The analyst creates a new category called *Avalanche Season* with three options, *Start*, *Middle*, and *End* using the **Add Category** button on the Persist toolbar. They now brush ranges in the bar chart and assign them to one of the avalanche season phases using the **Assign Category** button from the toolbar. 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 Persist Table (see Figure 6.3). This reveals some data artifacts, such as extraneous semicolons in column headers. They correct these in the table by directly editing the affected column header.

6.3 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 5.2(c). Next, we lay out the implementation-independent design principles of Persist before discussing concrete design and implementation in the next section.

6.3.1 Injecting Operations

Persist listens to native operations of a component, such as selection, and injects operations into the component, as illustrated in Figure 5.2(d), 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.

6.3.2 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 5.2(e) and shown in Figure 6.2(c). The provenance graph documents all the steps taken and can be used to navigate back in history and to create branches.

6.3.3 Transforming Data

Based on the provenance information, Persist applies the operations to a dataframe in sequence, as illustrated in Figure 5.2(f). 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 5.2(g), with the changes being illustrated in Figure 5.2(h). 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 which both can then be used in the notebook.

6.3.4 Updating Interactive Components

Operations can change how data is best displayed in the interactive components, illustrated by the arrow connecting Figure 5.2(f) to the scatterplots. For example, a chart should update after a filter was applied (removing or graying out the data points) or after a category was 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 should be displayed in the chart when a data point is labeled. We do this by updating the chart specification to

encode these additional attributes, either as a visual channel or as a tooltip.

6.3.5 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, there are scenarios where 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.

6.4 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 different visualization options: 1) an interface to arbitrary Vega-Altair charts and 2) an interactive table that can be used to view and manipulate dataframes directly.

As part of the **Vega-Altair integration**, we inject the Persist toolbar, shown in Figure 6.3(a). *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 6.3, 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 6.3(e)) or with individual cells (Figure 6.3(f)). 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 6.3(c)). 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 by a **provenance visualization**, rendered as a tree (Figure 6.3(d)). 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.

6.5 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. Other notebook frontends like VS Code Jupyter and Google Colab do not support the Notebook frontend API directly. Persist uses the Trrack [1] provenance tracking library, which we developed previously. Persist can be installed with `pip install persist_ext`.

The Persist extension package contains two modules: The first is the JupyterLab

Code Cell extension which augments the CodeCell API with Persist-specific features like managers for provenance and generated datasets. The cell extension is developed with TypeScript and React. The second module is the PersistOutput widget developed using anywidget [127], which is an abstraction over the popular JupyterWidgets [18]. The PersistOutput widget contains the core Persist module, the interfaces for Vega-Altair charts, and the custom data table UI. The widget backend is developed in Python, and the front end uses TypeScript, React, and Mantine React Table.

6.6 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, accurate, and reproducible and, additionally 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 will be able to complete most tasks, but Persist would result in fewer incorrect solutions.
- **H3-Completeness:** That using Persist would result in fewer skipped tasks.
- **H4-Reproducibility:** That using Persist would result in more reproducible notebooks.
- **H5-Workload:** That participants 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 6.4. We recruited 11 participants who have experience in data analysis using Jupyter notebooks and Pandas (4 identified as women, 7 as men). We recruited from our local student pool; our participants included undergraduate (2) and graduate students (4 PhD and 5 Master's). 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).

6.6.1 Procedure

The study employed a **within subject design** using two datasets: Avalanche records from the Utah Avalanche Center [126], and Video Games sales data from the Corgis Dataset Project [128]. The **tasks** involved data cleaning and data manipulation, such as (1a) removing columns not required for analysis, (1b) fixing column names to remove stray characters, (1c) changing the data type of a column, (2a) removing outlier records, (2b) removing records within a range, (3a) deriving a categorical column based on a numerical column, and a final task (3b) where participants looked at a plot with the new derived column to answer a question. Participants were given a prepared Jupyter notebook for each condition that contained instructions about each task and included boilerplate code, such as imports and data loading. Also, all visualization code (for both conditions) was given so that participants only had to execute data manipulation steps. See the supplemental material for the notebooks used. Each participant completed these tasks under both the Persist and traditional Pandas conditions. Tasks were matched between the datasets but varied slightly to fit each dataset. The order and dataset assignment were randomized using a Latin square to counterbalance any effects of datasets and order of conditions.

The study began with an introduction that included disclosures that the screens and room audio were recorded and their notebook would be analyzed, after which we obtained consent. This was followed by a survey where participants reported their experiences with Python, Pandas, and data wrangling. Following this, participants performed the main data analysis tasks under each condition. For the Persist condition, the participants were first given a 15-minute tutorial about Persist, followed by a hands-on session in the tutorial notebook. For the Pandas condition, participants were permitted to use any resource to find help, including using their own laptops to use search engines, consult documentation, or employ LLMs like ChatGPT for help. Participants could skip any task if they felt they could not proceed; if a participant skipped, the experimenter loaded a prepared dataset so that they could proceed with subsequent tasks. After each condition, the participants completed the NASA TLX [129] questionnaire to assess their subjective workload. Upon completing the tasks, participants filled out a poststudy survey and completed a semi-structured debrief interview to discuss their experiences with Persist. Each session lasted

approximately 1 hour and 45 minutes. Participants were compensated with a $30 gift card.

6.6.1.1 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 [129]). We also record preferences and feedback in a survey and a semi-structured interview.

6.6.1.2 Pilots, Analysis, and Experiment Planning

We conducted four pilots to evaluate tasks, different conditions and datasets, experimental setup, and our procedure. Due to the limitations of null hypothesis significance testing, we base our analysis on best practices for fair statistical communication in HCI [117] 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.

6.6.2 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 it from condition-specific discussions below.

Figure 6.5(a) 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 [130]. Moreover, the data shown in Figure 6.5(b) 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 skipped. Conversely, in the Pandas condition, two were incorrect and one was partially correct, while 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**, lending support to H4-Reproducibility.

The results of the subjective workload assessment are shown in Figure 6.6. We omitted the *physical demand* metric from these results as it was not relevant to our study context. Figure 6.6 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, 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 6.7, lending support to H6-Helpfulness.

6.6.3 Qualitative Results

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

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

Our goal with Persist is to enable seamless switching between code and interactions, allowing the analyst to use the best tool for the job. Therefore we asked participants about their thoughts on switching between Persist and Pandas. P2 responded, "maybe there are some features which are not present in this and we might want to use the code. So it is helpful to have both 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."

6.6.4 Study Discussion

Our results unambiguously demonstrate that participants were, on average, significantly faster using Persist than using standard data frame operations, validating H1-Speed. We also have some evidence to support H2-Correctness, H3-Completeness, and H4-Reproducibility, although the overall low number of errors, skipped tasks, and not-reproducible notebooks indicate that a more powerful study is necessary to make definite statements on these hypotheses. While we haven't conducted statistical tests to evaluate H5-Workload, we think the evidence from the NASA-TLX survey and the interviews unambiguously supports that participants do have lower subjective workload using Persist than using data frame operations. Similarly, our survey data and the qualitative interviews validate H6-Helpfulness. Looking closer at the data for completion times, it is noticeable that the Pandas condition has a higher variance in completion times, while the Persist condition has minimal variation. This could indicate that Persist is easy to learn and can be consistently applied, while the ease of using Pandas operations depends on the analysts' experience. Almost all participants had to look up syntax for most Pandas operations. It is notable, that even the more proficient participants in our study expressed that they found Persist helpful. We conclude that a tool like Persist can significantly speed up the workflow of most somewhat proficient data scientists, while still being an appreciated tool

for experts, thereby contributing to making computational data analysis accessible to a wider audience.

Another interesting observation is comparing the completion times of the two filter tasks (2a and 2b). While the difference in the Persist condition is negligible, the difference in the Pandas condition is large: Cutting the average from 520 seconds to 224 seconds. We observed that this speed-up is because participants copied the code they had written for Task 2a when completing 2b. This observation makes us consider whether we should provide functionality to copy workflows created with Persist [3].

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

6.7 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 general and complex (such as programming languages). This relationship is illustrated in Figure 6.8. Most visualization systems fall somewhere in the middle between these tools: It takes effort to learn to use a general-purpose visualization tool, yet it can be used for many things. The complex–specific quadrant is undesirable, while the easy-to-use–general quadrant is likely impossible to populate. We believe that Persist fills a unique niche by seamlessly bridging between the usability–specificity and the complexity–generality quadrants, thereby allowing some *operations* that would usually be in the domain of programming languages to be executed with easy-to-use interactive systems, while not

reducing the overall generality of the data analysis approach.

However, we recognize that Persist has limitations. While data operations on pandas and similar tools are scalable to large datasets with millions of rows, Persist is limited w.r.t. scalability by what can be plotted in a reasonable way. While scalable visualization solutions like VegaFusion exist, 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 could develop more sophisticated approaches for managing large interaction provenance, such as query-based retrieval of states of interest [52], automatically chunking multiple interactions as a higher level interaction, or features like undo-as-delete [45] 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 doesn't 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. While poststudy interviews indicate that Persist is easy to pick up after 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.

6.8 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 the 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 doesn't simplify is chart creation. Combining Persist with the chart creation technology shown by others [19], [131] 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 data frame.

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

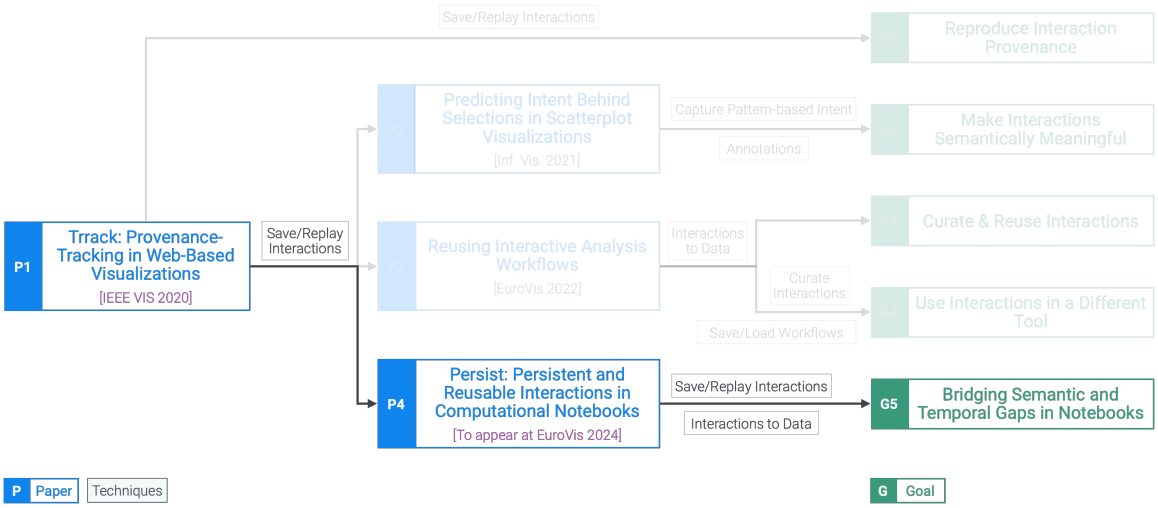

Figure 6.1: Interaction provenance for interactive visualizations in notebooks can be used to save and replay the interactions. Doing so enables the persistence of the interactions in the notebook, addressing the temporal gap arising from the transient nature of interactions. Mapping the interactions to data operations allows updating of the underlying dataframe, enabling the use of interaction results in code, addressing the semantic gap due to one-way communication between code and interactive visualizations.

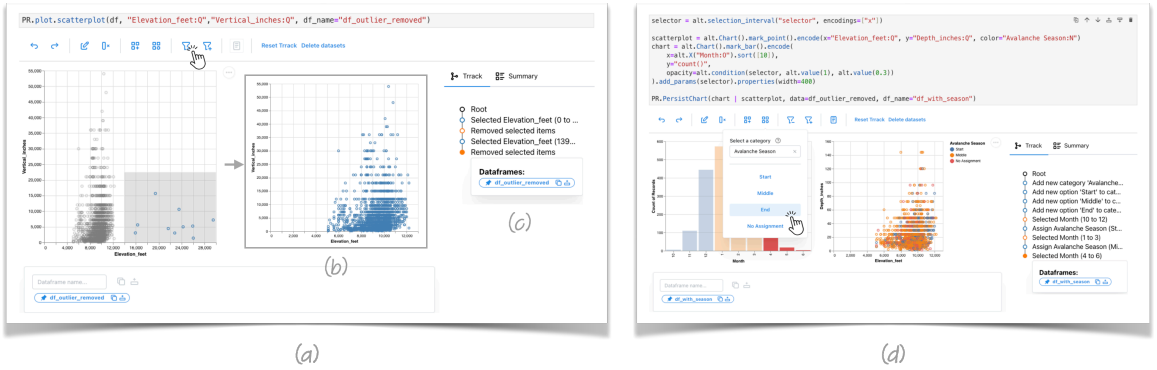

Figure 6.2: Examples of how Persist is used for data manipulation. (a) An analyst creates an interactive Vega-Altair scatterplot showing the elevation and depth of avalanches. They notice the outliers in the elevation and proceed to select the outliers and remove them (b). (c) The interactions are tracked in the provenance graph, and Persist creates a dataframe containing the updated data. (d) In a follow-up cell, they use the cleaned data to create a composite Vega-Altair chart with an interactive bar chart showing avalanche records aggregated by month next to the scatterplot of elevation versus 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.

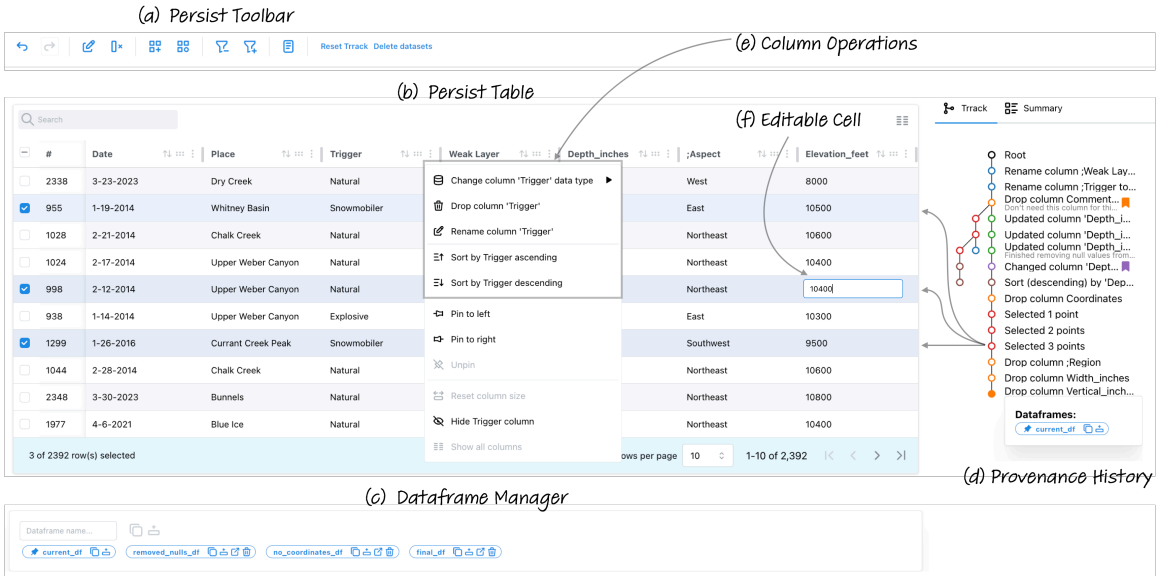

Figure 6.3: The Persist Table is an interactive data table that can be used for manipulating data frames. (a) The Persist toolbar is also injected into Vega-Altair charts. (b) The paginated table. Analysts can interact with the (e) headers, rows, or (f) individual cells. (c) The Dataframe Manager serves 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.

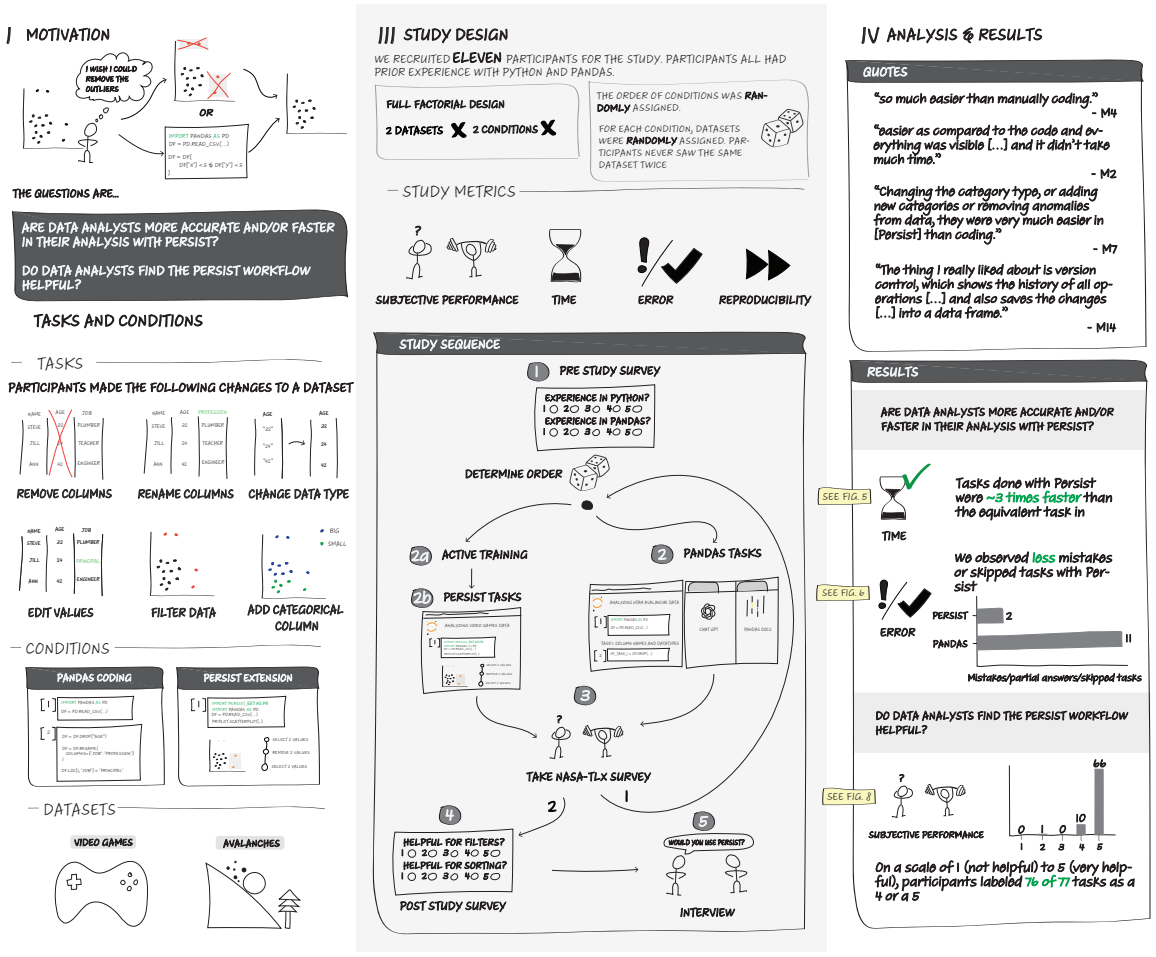

Figure 6.4: Illustration of study tasks and conditions, design, analysis, and results.

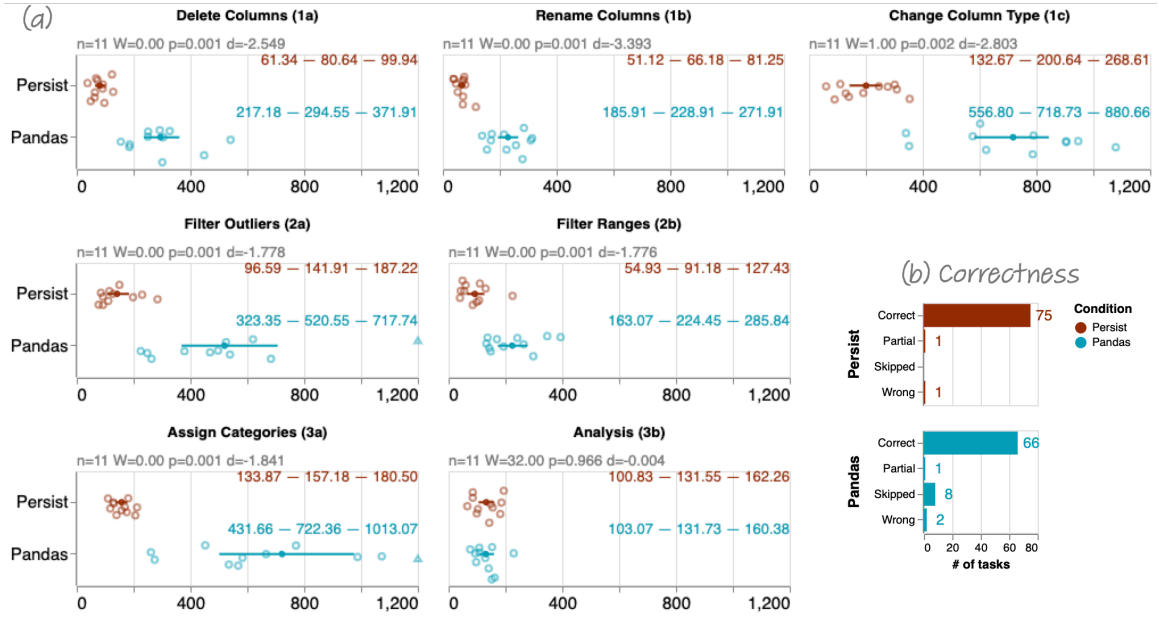

Figure 6.5: Time and correctness results. (a) Overview of task completion times for both conditions in seconds. Raw values are shown in jittered dot plots; the solid dot and lines show the mean and the 95% confidence intervals. The colored numbers show the lower and upper bound of the confidence interval and the mean respectively. The plot is clamped at 20 minutes (1200 seconds); data points exceeding 1200 seconds are shown as triangles. Statistical information is provided above the plots in gray. Persist was at least three times faster in all tasks where it was used (note that Task 3b–Analysis was a visual analysis task identical in both conditions). All the differences are statistically significant with "very large" to "huge" effect sizes [130]. (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.

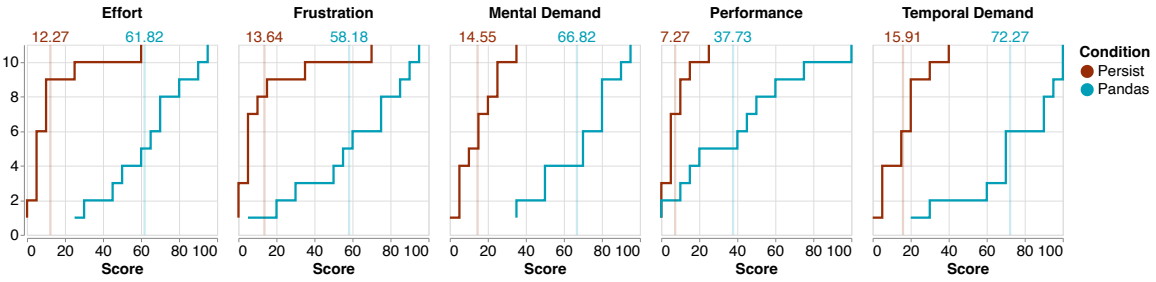

Figure 6.6: Subjective workload as measured by the NASA TLX shown as an empirical cumulative distribution function (eCDF), where the index of participants is on the y-axis, and the score is on the x-axis. Low values are "good" in all cases (low effort, low frustration, etc.) For performance, a low value 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.

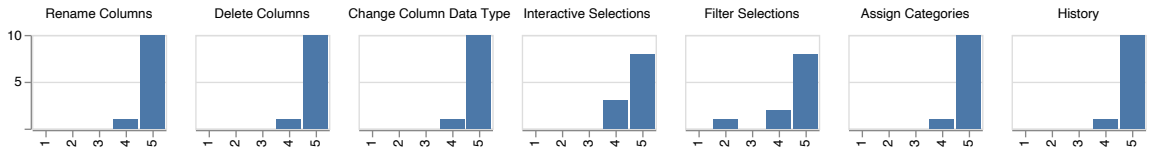

Figure 6.7: Histograms of ratings for the helpfulness of Persist for tasks on a 5-point Likert scale, where 1 corresponds to "not helpful," and 5 corresponds to "very helpful." Participants find Persist helpful or very helpful across tasks. For filters, one user expressed a preference for entering precise queries and rated Persist lower.

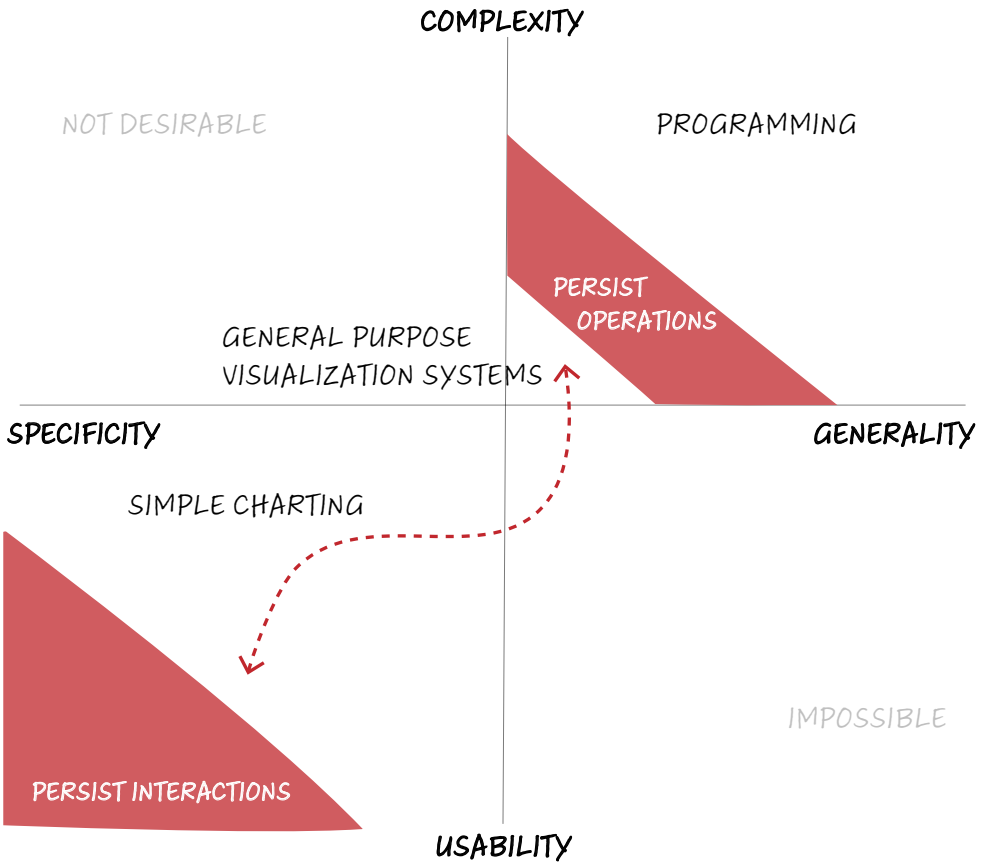

Figure 6.8: Conceptual trade-offs of data analysis systems.

CHAPTER 7

DISCUSSION

In this chapter, we will first reflect on the implications of our contributions to this dissertation and then look at the wider impact of our work on interactive visual analysis. Finally we will reflect on the challenges and limitations of our techniques and directions for future research in addressing the challenges.

7.1 Toward Literate Visual Analysis

For an analysis to be reproducible, the steps of the analysis and the data should be available for independent verification. Therefore, we must share the **process** and **narrative** to make visual analysis reproducible.

For visual analysis, the **process** is made up of the sequence of interactions — the interaction provenance — made by the analyst. Most visual analysis tools lack the capability to capture the interaction provenance, and when available, it is often limited for tasks such as a linear undo/redo stack or non-reproducible logs. To capture the interactions in visual analysis, in Chapter 3, we introduce the provenance tracking library Trrack [1]. Trrack can manage non linear provenance as branches in the provenance graph, therefore supporting non linear interaction provenance. Trrack also supports replaying the captured interactions.

Sharing the analysis narrative is a much larger challenge. One aspect of the narrative is the pattern-based intent, which depends on the data, but other high-level aspects of reasoning rooted in the domain knowledge play a more important role in the narrative. In Chapter 4, we introduce techniques to make the captured interaction provenance semantically meaningful. Our techniques semiautomatically capture the pattern-based intent of the analyst for selections and enable annotation of the captured provenance by the analyst. Capturing semantically meaningful provenance and the analysis narrative in the form of annotations can allow for independent reproduction and verification of the

visual analysis, thus building trust.

Our techniques for capturing semantically meaningful interactions and annotations open a path toward a literate visual analysis framework. Knuth proposed the paradigm of literate programming [9] as a way to write computer programs that are readable by humans and compilers. The crux of the literate programming approach is the co-location of executable code *(the process)* with the exposition of the program logic *(narrative)*. This idea of literate computing has also been extended to visualization and data analysis. The idea of literate visualization, as proposed by Wood et al. [132] aims to capture the design rationale behind the visualization design *(narrative)* along with the executable code for the visualization *(the process)*. Mathisen et al. [133] propose the paradigm of literate analytics where the goal is to capture the data analysis narrative, which includes insights about the method and the data *(narrative)* along with the analysis and the results *(the process)*.

A reproducible visual analysis session includes the provenance of the interactions *(the process)*, the semantics of the interactions, and the analyst's annotations *(the narrative)*. Following the literate paradigm, a reproducible visual analysis moves us closer to realizing a literate visual analysis framework. Such a framework would document the step-by-step process of visual analysis and include the analyst's narrative, including relevant domain knowledge.

Our approach to capturing the narrative of the visual analysis involves the analyst manually annotating the interaction provenance. However, this comes with its own challenges. The annotation process requires added effort on the part of the analyst, often lacks guidance, and is ad-hoc, making it difficult to analyze programmatically.

A framework for capturing the annotations in a structured manner can go a long way in addressing this challenge. In their work on literate visualization, Wood et al. [132] present the idea of "narrative schemas" for structuring the design exposition during visualization design. Narrative schemas specify different parts of the narrative document coupled with rules for using them. Schemas can be structured in the form of Socratic "dialogue" with questions to prompt the designer or with a more formal framework like "visualization algebra." The idea of schemas can be extended to prompt the annotations in the interaction provenance. Prior research on capturing structured annotations like the SVC (subject-verb-complement) structure proposed by Vanhulst et al. [134] can be

adapted for creating a schema. Schemas can also be created by extending the knowledge externalization frameworks like the implicit error framework by McCurdy et al. [135] or the Data Hunches framework by Lin et al. [136].

Making visual analysis reproducible is just the first step toward the paradigm of literate visual analysis. We believe that a combination of strategies like provenance tracking, inferring interaction intents, structured annotations, and storytelling [50] to combine data analysis and reporting is the way to achieve literate visual analysis.

7.2 Reusable Workflows as Templates

A reusable visual analysis can be applied to an updated dataset and yields similar results. We must meaningfully apply the captured interaction provenance to an updated dataset to reuse the analysis. Further, the interactive analysis process is often iterative, and the captured provenance can contain multiple branches. It is often desirable to only reuse parts of such interaction provenance.

In Chapter 5, we introduce techniques to reuse the captured interactions on an updated version of the dataset. Further, our techniques support curation to extract parts of the interaction provenance as workflows for later use. These curated workflows are analogous to functions in computational data analysis in that they can be defined once and reused multiple times.

However, an important distinction between these workflows and functions is that functions can be parameterized. Therefore, functions are highly adaptable to changes in the data. Workflows based on semantically meaningful interactions, while more robust to dataset updates than workflows based on mouse/keyboard interactions, might still not be reusable on datasets with large changes or from a different domain. Our workflow capture technique relies on tracking view specifications, downstream selections, and transforms. Applying the dataset to an unrelated dataset will almost always result in incorrect results.

We address this issue by introducing a review process by the analyst. However, after determining that the reapplication of the interaction was incorrect, the analyst must manually fix the workflow. It would be valuable to have the system recommend alternatives when the analyst marks the interaction as incorrect. Developing this approach would require the system to understand the changes to the dataset, which becomes a harder

challenge if the new data belongs to a different domain altogether.

We also see potential in using workflows as templates for recurring tasks, such as data cleanup on datasets generated by the same instrument, although for different experiments. Going back to parallels between the workflows and functions, parts of workflows can be parameterized where the analyst can specify any changes to these parts when using the workflow. Parts of the workflow, like the dimensions on which the interactions are done or the algorithm and hyperparameters used by a pattern-based interaction, are ripe for parameterization. For example, an analyst could reuse the templatized workflow on a new dataset by updating the selection operation to select the correct outliers in the new data but reuse the downstream data transforms like filters or aggregates.

Programming-by-example (PBE) systems like Wrex [100] and FlashExtract [137] have demonstrated the utility of generating code from interactions for data analysis and wrangling tasks. A common approach in PBE is to synthesize code from interactions and then edit the code to remove parts that are not required or for a new context. Our techniques above can be extended to curate the interaction provenance to keep only the required parts before generating the code. Templatized workflows could further augment the code synthesis by adapting the interactions to the new context and only then generating the code.

7.3 Interactions as Shared Representations

As discussed in Chapter 6, data analysis systems can range from easy-to-use and task-specific systems to highly flexible and hard-to-learn systems. Tools like Tableau or PowerBI are easy to learn but are limited in the possible analysis by built-in interactions. In contrast, programming environments like Python or R provide a steep learning curve but support a wide variety of analysis tasks with a potential for extension using libraries. While it is likely impossible to have a single system that combines ease of use and high-flexibility in a single system, it is more practical to use a combination of user-friendly, task-specific systems for certain tasks and more versatile, code-based systems for complex, ad-hoc tasks. In practice, transitioning between various systems is challenging due to a lack of compatibility in porting the analysis between different tools.

In the realm of software development, Application Programming Interfaces (APIs)

serve as a solution to similar challenges. APIs act as contracts that define the rules and methods for different pieces of software to communicate and interact. Drawing inspiration from this, a well-structured representation of analytical steps could serve a similar purpose in exchanging data analysis steps. Such a representation would enable different analysis systems to understand and apply analysis steps from other systems, much like APIs allow for interaction between software. By establishing a standardized 'API' for data analysis, it would be possible to transfer data, context, methodology, and insights from one analytical tool to another, enhancing the overall reproducibility and reusability of the data analysis process. In Chapter 5, we discuss our contributions to facilitating such exchanges [3]. Our techniques capture interaction provenance and enable its reapplication within the same or different analysis environments using the interaction provenance as this 'API' or a shared representation between our visualization prototype and a Jupyter Notebook.

The shared representation sits between the interactions and data operations, allowing the translation of one to another. However, our implementation serves as a proof-of-concept to show the value of developing such a shared representation between different tools. Generalizing this approach presents huge challenges. Different tools support different interactions and operations, so creating a single API representing every combination is infeasible.

Future research could look into grammars and composability to address this challenge. Grammars have firmly found their place for creating arbitrary visualizations [138] by composing smaller units of a visualization. The most popular example of such a grammar is Vega/VegaLite [139]. Similarly, a grammar of basic interactions and operations could be developed with support for composing complex interactions and operations from the basic ones.

7.4 Supporting Iterative Analysis in Computational Notebooks

Computational notebooks are a hybrid approach to data analysis by combining the expressivity of programming languages with the interactivity of visual analysis. Despite their strengths, there are gaps between interactive outputs and the code cells in the computational notebooks. Two gaps identified by Wu et al. [19] are — 1) the semantic gap, where the code generates the interactive visualization, but the results of the interactions are not accessible in the code, and 2) the temporal gap, where the code and its outputs are persistent, but the interactions are lost on cell re-execution or notebook restart.

In Chapter 6, we present our contribution — Persist — to address these gaps by leveraging the interaction provenance as a shared abstraction between the code and interactions. Persist uses the interaction provenance for updating the visualization and the underlying dataset, allowing access to the updated dataset as a Pandas dataframe. Persist also directly saves the interaction provenance in the notebook metadata and automatically replays it when reloading the notebook or re-executing the cell.

We previously compared the Persist approach with the B2 approach proposed by Wu et al.[19] for addressing the semantic and temporal gaps. Wu et al. [19] discuss a third gap in computational notebook — the layout gap — that stems from the differences in the linear nature of a notebook and the non linear iterative nature of the interactive analysis. Iterative analysis with code often results in a messy notebook with out-of-order cells or fragile bits of copy-pasted code [140]. In the study by Chattopadhyay et al. [89], data scientists echo the sentiments about messy notebooks and further express concerns with the visualizations being limited by the boundaries of the cell output. The B2 systems address the layout gap by moving all the visualization outputs to a common dashboard, taking them outside the linear flow of the notebook. Interactions output their generated code independent of the original code cell that generated the visualization. The Persist approach of creating branches in the provenance to support alternate analysis paths supports basic iterations within the cell but does not contribute toward addressing the layout gap directly. Combining the B2 approach of collecting visualizations like a dashboard with Persist's powerful interaction provenance can potentially effectively support iterations in the analysis.

Our discussion of the Persist technique largely focuses on sequentially executed notebooks like Jupyter. Reactive notebooks, like Observable [13], automatically update in response to execution in any cell. Reactive models address issues like out-of-order execution states that notebooks inevitably reach during iterative analysis. Further research is needed to evaluate how the Persist approach of capturing per-cell provenance fares with the reactive model, where a cell change can propagate to multiple cells.

7.5 Role of Analysis Provenance

Analysis provenance plays a crucial role in ensuring transparency and reproducibility in research. In this section, we will discuss the role of provenance in data analysis, its implications, and the future questions it raises.

7.5.1 Improving Transparency and Collaboration

Provenance provides a detailed record of the data analysis process, which is invaluable for reproducibility. It allows other researchers to understand precisely how results were derived, making replicating studies and validating findings easier. This transparency is crucial in scientific research, where reproducibility is a cornerstone of credibility. In a team setting, provenance can help experienced researchers validate work done by the junior researchers. On the other hand, provenance can serve as an educational tool for new researchers. By examining the provenance of a data analysis, researchers can learn from the methodologies and thought processes of more experienced researchers. This aspect is particularly beneficial in collaborative projects where team members may need to understand and build upon each other's work.

7.5.2 Misusing Analysis Provenance

While provenance promotes transparency, it could also be misused. For example, P-hacking is the act of manipulating the data analysis until statistically significant results are obtained. This practice can lead to misleading conclusions and distort the scientific record. Provenance records can facilitate such manipulations by making it easier for researchers to track multiple analyses and only selectively report on favorable ones. Developing ethical guidelines and tools to detect and discourage such misuse is essential. However, doing so is hard and remains an open research topic.

7.5.3 Provenance as a Part of the Publication Process

Publication venues in fields like preclinical or applied research encourage the practice of **preregistration** because it is valuable for transparency, rigor, and reproducibility. Preregistration is the practice of regsitering the study design, hypothesis that are tested, and plans for the analysis before data collection or analysis. Preregistration helps distinguish between prediction and postdiction [141], and address data-driven manipulation of results. Similarly, it is common to share the data and analysis scripts/code with the publication to enable replication studies and validate the original published analysis.

Incorporating analysis provenance in the publication process can aid in validating published results. Journals could require the submission of analysis provenance alongside manuscripts. This requirement would allow reviewers to assess the methodology and data analysis process more thoroughly, leading to more rigorous and reliable scientific publications.

A key challenge in incorporating provenance in research and publication is the lack of standardization. Different fields, analysis tools, and data types require different provenance models. Developing universal standards or adaptable frameworks for provenance tracking could be a huge undertaking but would greatly benefit the scientific community. Meanwhile, there are also solutions with lower technical and organizatinal hurdles that could be implemented. For example, a provenance tracking library could generate a report for the analysis, showing intermediate states as screenshots and documenting the interactions, in a widely readable format like PDF. Measures such as digital signing could be adopted to ensure the validity of the generated reports.

CHAPTER 8

CONCLUSION

Interactive visual analysis leverages our perceptual capabilities and is intuitive compared to computational analysis, which is flexible but requires programming knowledge and has a steep learning curve. However, visual analysis falls short on the reproducibility and reusability front. Reproducibility plays a critical role in establishing trust, and reusability offers a way to improve efficiency and standardize analysis processes. Visual analysis steps cannot be easily captured, replayed, and reused. A hybrid analysis approach allows us to leverage the easy-to-use interactive analysis and flexible code-based analysis. However, the hybrid analysis approaches have their own challenges. Using multiple analysis tools is difficult, given the lack of compatibility between various tools for sharing the analysis steps. Computational notebooks do a good job of combining interactive and code-based analysis but have semantic and temporal gaps between the two paradigms. In the hybrid approach, the interactive analysis parts lag behind in reproducibility and reusability, thereby reducing reproducibility and reusability for the entire analysis.

In this thesis, we propose using interaction provenance — the sequence of interactions in an analysis — to improve the reproducibility and reusability of visual analysis. We introduce a software library and multiple techniques to capture, reproduce, and reuse the interaction steps. We demonstrate our techniques with prototype systems and evaluate the techniques with a combination of expert interviews, crowdsourced study, and in-lab study. Figure 8.1 summarizes out contributions towards the goals described in Chapter 1.

To address the challenge of effectively capturing and replaying the interaction provenance in interactive visualizations, we developed the Trrack library [1]. Trrack is a provenance tracking library for web based applications and is instrumental in capturing interaction provenance for interactive visualization prototypes we use for demonstrating our other techniques. The library can also replay the interactions, thereby improving the

reproducibility of the captured interactions.

The interactions captured by Trrack are based around keyboard/mouse events and lack information on the semantics of the interaction. Therefore, our next step was to develop techniques for capturing the analyst's rationale behind the interaction. We do so by automatically capturing the pattern-based intent of the analyst and allowing the analyst to annotate the interaction provenance in order to externalize their domain knowledge. Having access to interaction provenance and the analyst's rationale in the form of pattern-based intents and annotations further improves the reproducibility of the captured interactions.

As our next step, we devised techniques to utilize these semantically rich interactions to reapply interaction provenance on updated datasets. We also explored curating segments of interactions into reusable workflows, demonstrating their application across updated datasets and different analytical environments. These techniques improve the reusability of captured interactions and demonstrate the potential of interaction provenance to enable analysis spanning multiple tools and environments.

Our final contribution to this thesis is the Persist technique, which bridges the semantic and temporal gaps between code cells and interactive outputs in computational notebooks. We demonstrate Persist with a JupyterLab plugin that tracks the interaction provenance for interactive visualizations within notebook cell outputs. This provenance is then used to update data and visualizations, allowing access to the results of the interactions in downstream code cells. Persist stores the captured provenance in the notebook and replays them when the cell is re-executed, thereby enabling the persistence of the interactions alongside the code. Thus, Persist effectively bridges the interaction-code divide, improving computational notebooks' overall reproducibility and reusability.

In this dissertation, we explored various techniques for improving the reproducibility and reusability of visual analysis. Our techniques demonstrate the role interaction provenance can play in doing so. We also demonstrate how our techniques can bridge gaps between different analysis environments and tools. Our work leaves behind interesting research threads that can be pursued, such as annotating the interaction provenance to externalize analysts' knowledge, developing a standard for communicating analysis steps between multiple tools, and developing a literate visual analytics framework.

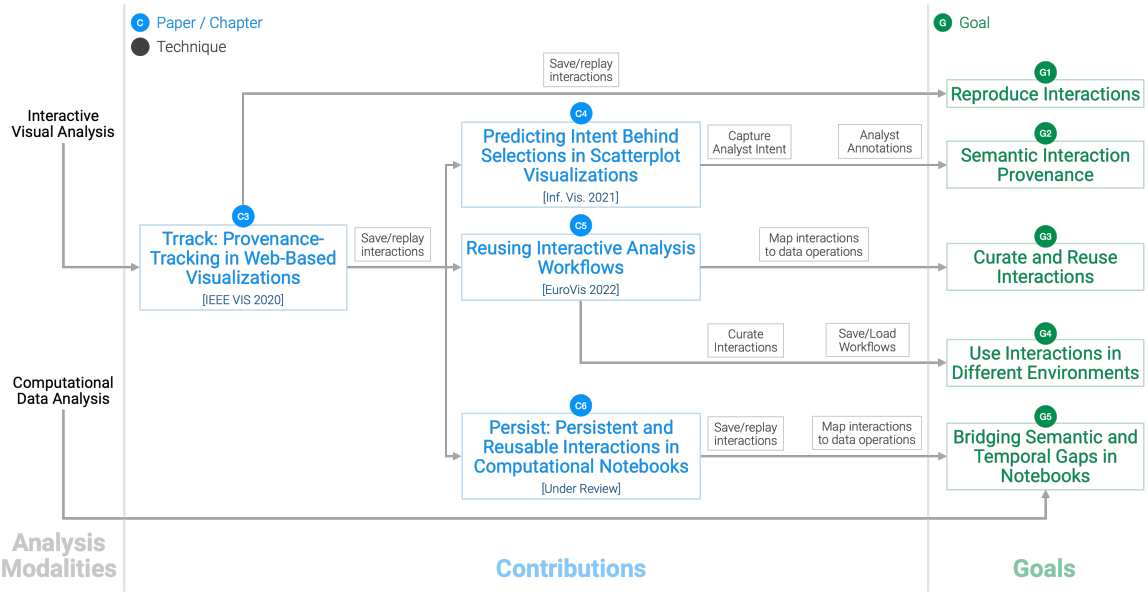

Figure 8.1: A summary of our contributions in the dissertation. In Chapter 3, we discussed our first contribution — Trrack — which is a provenance tracking library [1] that captures and replays the interactions (G1). In Chapter 4, we discussed techniques to make the interaction provenance semantically meaningful [2] by semiautomatically capturing the analysis intent and analyst annotations (G2). In Chapter 5, we discussed techniques for curating and reusing the interactions on an updated dataset by mapping the interactions to data operations and using the interactions in a different analysis environment using curated workflows [3] (G3, G4). In Chapter 6, discussed how we bridge the gaps between code and interactions in computational notebooks [21] by capturing the interaction provenance in the cell outputs (G5).

REFERENCES

[1] Z. Cutler, K. Gadhave, and A. Lex, "Trrack: A library for provenance-tracking in web-based visualizations," in *Proc. IEEE Vis. Conf., 2020*, Oct. 25–30, 2020, pp. 116–120.

[2] K. Gadhave *et al.*, "Predicting intent behind selections in scatterplot visualizations," *Inf. Vis.*, vol. 20, no. 4, pp. 207–228, Oct. 2021.

[3] K. Gadhave, Z. Cutler, and A. Lex, "Reusing interactive analysis workflows," *Comput. Graph. Forum*, vol. 41, no. 3, pp. 133–144, Jun. 2022.

[4] *Tableau*. (2015). Accessed: Feb. 2, 2024. [Online]. Available: http://www.tableau.com

[5] *Power BI*. (2015). Accessed: Feb. 2, 2024. [Online]. Available: https://app.powerbi.com

[6] R. Kosara, "Notebooks for data analysis and visualization: Moving beyond the data," *IEEE Comput. Graph. Appl.*, vol. 43, no. 1, pp. 91–96, Jan. 2023.

[7] M. Baker, "1,500 scientists lift the lid on reproducibility," *Nature*, vol. 533, no. 7604, pp. 452–454, May 2016.

[8] M. G. Pratt, S. Kaplan, and R. Whittington, "Editorial essay: The tumult over transparency: Decoupling transparency from replication in establishing trustworthy qualitative research," *Adm. Sci. Q.*, vol. 65, no. 1, pp. 1–19, Mar. 2020.

[9] D. E. Knuth, "Literate programming," *Comput. J.*, vol. 27, no. 2, pp. 97–111, Jan. 1984.

[10] NumFOCUS, Inc. (2013). *pandas - Python Data Analysis Library*, Accessed: Feb. 2, 2024. [Online]. Available: https://pandas.pydata.org/

[11] W. McKinney, "Data structures for statistical computing in Python," in *Proc. 9th Python Sci. Conf.*, Jun. 28–Jul. 3, 2010, pp. 56–61.

[12] B. E. Granger and F. Pérez, "Jupyter: Thinking and storytelling with code and data," *Comput. Sci. Eng.*, vol. 23, no. 2, pp. 7–14, Mar. 2021.

[13] M. Bostock. (2024). *Observable*, Accessed Jan. 25, 2024. [Online]. Available: https://observablehq.com

[14] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 data-driven documents," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 12, pp. 2301–2309, Oct. 2011.

[15] M. Bostock, "Introduction to Notebooks," *Observable*, Oct. 12, 2020. [Online]. Available: https://observablehq.com/@observablehq/introduction-to-notebooks

[16] J. VanderPlas *et al.*, "Altair: Interactive statistical visualizations for Python," *J. Open Source Softw.*, vol. 3, Art. no. 1057, Dec. 2018.

[17] Bokeh Development Team. (2018). *Bokeh*, Accessed: Feb. 2, 2024. [Online]. Available: https://docs.bokeh.org/en/1.0.1/docs/citation.html

[18] IPython Widget Team. (2015). *Jupyter Widgets*, Accessed: Feb. 2, 2024. [Online]. Available: https://ipywidgets.readthedocs.io/en/stable

[19] Y. Wu, J. M. Hellerstein, and A. Satyanarayan, "B2: Bridging code and interactive visualization in computational notebooks," in *Proc. 33rd Annu. ACM Symp. User Interface Softw. Technol.*, Oct. 20–23, 2020, pp. 152–165.

[20] A. Rule, A. Tabard, and J. D. Hollan, "Exploration and explanation in computational notebooks," in *Proc. 2018 CHI Conf. Hum. Factors Comput. Syst.*, Apr. 21–26, 2018, pp. 1–12.

[21] K. Gadhave, Z. T. Cutler, and A. Lex, "Persist: Persistent and reusable interactions in computational notebooks," 2023. *OSF preprint*. [Online]. Available: https://doi.org/10.31219/osf.io/9x8eq.

[22] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Proc. 1996 IEEE Symp. Vis. Lang.*, Sep. 3–6, 1996, pp. 336–343.

[23] L. Wilkinson, "The grammar of graphics," in *Handbook of Computational Statistics: Concepts and Methods*, J. E. Gentle, W. K. Härdle, and Y. Mori, Eds. Berlin, Germany: Springer, 2012, pp. 375–414.

[24] J. S. Yi, Y. ah Kang, J. Stasko, and J. Jacko, "Toward a deeper understanding of the role of interaction in information visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 6, pp. 1224–1231, Nov. 2007.

[25] D. Gotz and M. X. Zhou, "Characterizing users' visual analytic activity for insight provenance," in *Proc. 2008 IEEE Symp. Vis. Anal. Sci. Technol.*, Oct. 19–24, 2008, pp. 123–130.

[26] J. Heer and B. Shneiderman, "Interactive dynamics for visual analysis: A taxonomy of tools that support the fluent and flexible use of visualizations," *Queue*, vol. 10, no. 2, pp. 30–55, Feb. 2012.

[27] M. Brehmer and T. Munzner, "A multi-level typology of abstract visualization tasks," *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 12, pp. 2376–2385, Dec. 2013.

[28] A. Rind, W. Aigner, M. Wagner, S. Miksch, and T. Lammarsch, "Task Cube: A three-dimensional conceptual space of user tasks in visualization design and evaluation," *Inf. Vis.*, vol. 15, no. 4, pp. 288–300, Oct. 2016.

[29] C. Fan and H. Hauser, "Fast and accurate CNN-based brushing in scatterplots," *Comput. Graph. Forum*, vol. 37, no. 3, pp. 111–120, Jun. 2018.

[30] A. R. Martin and M. O. Ward, "High dimensional brushing for interactive exploration of multivariate data," in *IEEE Vis. Conf.*, Oct. 29–Nov. 3, 1995, pp. 271–271.

[31] M. Derthick, J. Kolojejchick, and S. F. Roth, "An interactive visual query environment for exploring data," in *Proc. 10th Annu. ACM Symp. User Interface Softw. Technol.*, Oct. 14–17, 1997, pp. 189–198.

[32] B. Shneiderman, "Dynamic queries for visual information seeking," *IEEE Softw.*, vol. 11, no. 6, pp. 70–77, Nov. 1994.

[33] R. A. Becker and W. S. Cleveland, "Brushing scatterplots," *Technometrics*, vol. 29, no. 2, pp. 127–142, May 1987.

[34] J. Heer, M. Agrawala, and W. Willett, "Generalized selection via interactive query relaxation," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, Apr. 5–10, 2008, pp. 959–968.

[35] S. L. Su, S. Paris, and F. Durand, "QuickSelect: History-based selection expansion," in *Proc. Graphics Interface 2009*, May 25–27, 2009, pp. 215–221.

[36] L. Xiao, J. Gerth, and P. Hanrahan, "Enhancing visual analysis of network traffic using a knowledge representation," in *Proc. 2006 IEEE Symp. Vis. Anal. Sci. Technol.*, Oct. 31–Nov. 2, 2006, pp. 107–114.

[37] E. D. Ragan, A. Endert, J. Sanyal, and J. Chen, "Characterizing provenance in visualization and data analysis: An organizational framework of provenance types and purposes," *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 1, pp. 31–40, Jan. 2016.

[38] J. Freire, D. Koop, E. Santos, and C. T. Silva, "Provenance for computational tasks: A survey," *Comput. Sci. Eng.*, vol. 10, no. 3, pp. 11–21, May 2008.

[39] C. North, R. Chang, A. Endert, W. Dou, R. May, B. Pike, and G. Fink, "Analytic provenance: Process+interaction+insight," in *CHI '11 Ext. Abstr. Hum. Factors Comput. Syst.*, May 7–12, 2011, pp. 33–36.

[40] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-Science: An overview of workflow system features and capabilities," *Future Gener. Comput. Syst.*, vol. 25, no. 5, pp. 528–540, May 2009.

[41] S. G. Parker and C. R. Johnson, "SCIRun: A scientific programming environment for computational steering," in *Proc. 1995 ACM/IEEE Conf. Supercomput.*, Dec. 8, 1995, p. 52.

[42] L. Bavoil *et al.*, "VisTrails: Enabling interactive multiple-view visualizations," in *Proc. IEEE Vis., 2005*, Oct. 23–28, 2005, pp. 135–142.

[43] C. Dunne, N. Henry Riche, B. Lee, R. Metoyer, and G. Robertson, "GraphTrail: Analyzing large multivariate, heterogeneous networks while supporting exploration history," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, May 5–10, 2012, pp. 1663–1672.

[44] B. Yu and C. T. Silva, "VisFlow - Web-based visualization framework for tabular data with a subset flow model," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 251–260, Jan. 2017.

[45] J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala, "Graphical histories for visualization: Supporting analysis, communication, and evaluation," *IEEE Trans. Vis. Comput. Graph.*, vol. 14, no. 6, pp. 1189–1196, Nov. 2008.

[46] N. Kadivar, V. Chen, D. Dunsmuir, E. Lee, C. Qian, J. Dill, C. Shaw, and R. Woodbury, "Capturing and supporting the analysis process," in *Proc. IEEE Symp. Vis. Anal. Sci. Technol.*, Oct. 12–13, 2009, pp. 131–138.

[47] M. Kreuseler, T. Nocke, and H. Schumann, "A history mechanism for visual data mining," in *IEEE Symp. Inf. Vis.*, Oct. 10–12, 2004, pp. 49–56.

[48] Y. B. Shrinivasan and J. J. van Wijk, "Supporting the analytical reasoning process in information visualization," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, Apr. 5–10, 2008, pp. 1237–1246.

[49] M. Streit, H.-J. Schulz, A. Lex, D. Schmalstieg, and H. Schumann, "Model-driven design for the visual analysis of heterogeneous data," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 6, pp. 998–1010, Jun. 2012.

[50] S. Gratzl, A. Lex, N. Gehlenborg, N. Cosgrove, and M. Streit, "From visual exploration to storytelling and back again," *Comput. Graph. Forum*, vol. 35, no. 3, pp. 491–500, Jun. 2016.

[51] T. Fujiwara, T. Crnovrsanin, and K.-L. Ma, "Concise provenance of interactive network analysis," *Vis. Inf.*, vol. 2, no. 4, pp. 213–224, Dec. 2018.

[52] H. Stitz, S. Gratzl, H. Piringer, T. Zichner, and M. Streit, "KnowledgePearls: Provenance-based visualization retrieval," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 120–130, Jan. 2019.

[53] A. Camisetty, C. Chandurkar, M. Sun, and D. Koop, "Enhancing web-based analytics applications through provenance," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 131–141, Jan. 2019.

[54] B. Karer, H. Hagen, and D. J. Lehmann, "Insight beyond numbers: The impact of qualitative factors on visual data analysis," *IEEE Trans. Vis. Comput. Graph.*, vol. 27, no. 2, pp. 1011–1021, Feb. 2021.

[55] B. A. Myers, "Creating user interfaces by demonstration," Ph.D. dissertation, Dept. Comput. Sci., Univ. Toronto, Toronto, Canada, 1987.

[56] P. H. Nguyen, K. Xu, A. Wheat, B. W. Wong, S. Attfield, and B. Fields, "SensePath: Understanding the sensemaking process through analytic provenance," *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 1, pp. 41–50, Jan. 2016.

[57] W. Dou, D. H. Jeong, F. Stukes, W. Ribarsky, H. R. Lipford, and R. Chang, "Recovering reasoning processes from user interactions," *IEEE Comput. Graph. Appl.*, vol. 29, no. 3, pp. 52–61, May 2009.

[58] E. T. Brown, A. Ottley, H. Zhao, Q. Lin, R. Souvenir, A. Endert, and R. Chang, "Finding Waldo: Learning about users from their interactions," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 12, pp. 1663–1672, Dec. 2014.

[59] D. Ceneda, T. Gschwandtner, and S. Miksch, "A review of guidance approaches in visual data analysis: A multifocal perspective," *Comput. Graph. Forum*, vol. 38, no. 3, pp. 861–879, Jun. 2019.

[60] A. Ottley, R. Garnett, and R. Wan, "Follow the clicks: Learning and anticipating mouse interactions during exploratory data analysis," *Comput. Graph. Forum*, vol. 38, no. 3, pp. 41–52, Jun. 2019.

[61] B. Steichen, G. Carenini, and C. Conati, "User-adaptive information visualization: Using eye gaze data to infer visualization tasks and user cognitive abilities," in *Proc. 2013 Int. Conf. Intell. User Interfaces*, Mar. 19–22, 2013, pp. 317–328.

[62] M. Gingerich and C. Conati, "Constructing models of user and task characteristics from eye gaze data for user-adaptive information highlighting," in *Proc. AAAI Conf. Artif. Intell.*, Jan. 25–30, 2015, pp. 1728–1734.

[63] S. Monadjemi, R. Garnett, and A. Ottley, "Competing models: Inferring exploration patterns and information relevance via Bayesian model selection," *IEEE Trans. Vis. Comput. Graph.*, vol. 27, no. 2, pp. 412–421, Feb. 2021.

[64] L. Battle, R. Chang, and M. Stonebraker, "Dynamic prefetching of data tiles for interactive visualization," in *Proc. Int. Conf. Manag. Data, 2016*, Jun. 26–Jul. 1, 2016, pp. 1363–1375.

[65] K. Dimitriadou, O. Papaemmanouil, and Y. Diao, "Explore-by-example: An automatic query steering framework for interactive data exploration," in *Proc. 2014 ACM SIGMOD Int. Conf. Manag. Data*, Jun. 22–27, 2014, pp. 517–528.

[66] J. Mackinlay, P. Hanrahan, and C. Stolte, "Show me: Automatic presentation for visual analysis," *IEEE Trans. Vis. Comput.*, vol. 13, no. 6, pp. 1137–1144, Nov. 2007.

[67] M. Tory and V. Setlur, "Do what I mean, not what I say! Design considerations for supporting intent and context in analytical conversation," in *Proc. IEEE Conf. Vis. Anal. Sci. Technol.*, Oct. 20–25, 2019, pp. 93–103.

[68] B. Saket, H. Kim, E. T. Brown, and A. Endert, "Visualization by demonstration: An interaction paradigm for visual data exploration," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 331–340, Jan. 2017.

[69] B. Saket, L. Jiang, C. Perin, and A. Endert, "Liger: Combining interaction paradigms for visual analysis," 2019, *arXiv:1907.08345*.

[70] Ç. Demiralp, P. J. Haas, S. Parthasarathy, and T. Pedapati, "Foresight: Recommending visual insights," *Proc. VLDB Endow.*, vol. 10, no. 12, pp. 1937–1940, Aug. 2017.

[71] D. P. Groth and K. Streefkerk, "Provenance and annotation for visual exploration systems," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 6, pp. 1500–1510, Nov. 2006.

[72] W. Wright, D. Schroh, P. Proulx, A. Skaburskis, and B. Cort, "The Sandbox for analysis: Concepts and methods," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, Apr. 22–27, 2006, pp. 801–810.

[73] M. Wohlfart and H. Hauser, "Story telling for presentation in volume visualization," in *Proc. 9th Joint Eurographics/IEEE VGTC Conf. Vis.*, May 23–25, 2007, pp. 91–98.

[74] R. Eccles, T. Kapler, R. Harper, and W. Wright, "Stories in GeoTime," *Inf. Vis.*, vol. 7, no. 1, pp. 3–17, Mar. 2008.

[75] B. C. Kwon, F. Stoffel, D. Jäckle, B. Lee, and D. Keim, "VisJockey : Enriching data stories through orchestrated interactive visualization," in *Comput. Journal. Symp.*, Oct. 24–25, 2014, pp. 1–5.

[76] D. Sharma. (2015). "Ask HN: Why did literate programming not catch on?" *Hacker News*. Accessed: Feb. 2, 2024. [Online]. Available: https://news.ycombinator.com/item?id=10069748

[77] J. Goecks, A. Nekrutenko, J. Taylor, and The Galaxy Team, "Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biol*, vol. 11, no. 8, Art. no. R86, Aug. 2010.

[78] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: An extensible system for design and execution of scientific workflows," in *Proc. 16th Int. Conf. Sci. Stat. Database Manag. 2004*, Jun. 23, 2004, pp. 423–424.

[79] M. R. Berthold *et al.*, "KNIME - The Konstanz information miner: Version 2.0 and beyond," *ACM SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 26–31, Nov. 2009.

[80] L. Zaman *et al.*, "GEM-NI: A system for creating and managing alternatives in generative design," in *Proc. 33rd Annu. ACM Conf. Hum. Factors Comput. Syst.*, Apr. 18–23, 2015, pp. 1201–1210.

[81] H. Lam, "A framework of interaction costs in information visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 14, no. 6, pp. 1149–1156, Nov. 2008.

[82] B. Shneiderman, "Direct manipulation: A step beyond programming languages," *ACM SIGSOC Bull.*, vol. 13, no. 2–3, p. 143, May 1981.

[83] K. Xu, A. Ottley, C. Walchshofer, M. Streit, R. Chang, and J. Wenskovitch, "Survey on the analysis of user interactions and visualization provenance," *Comput. Graph. Forum*, vol. 39, no. 3, pp. 757–783, Jun. 2020.

[84] S. van den Elzen and J. J. van Wijk, "Small multiples, large singles: A new approach for visual data exploration," *Comput. Graph. Forum*, vol. 32, no. 3pt2, pp. 191–200, Jun. 2013.

[85] Y. V. Chen, Z. C. Qian, R. Woodbury, J. Dill, and C. D. Shaw, "Employing a parametric model for analytic provenance," *ACM Trans. Interact. Intell. Syst.*, vol. 4, no. 1, pp. 6:1–6:32, Apr. 2014.

[86] K. Wongsuphasawat, Y. Liu, and J. Heer, "Goals, process, and challenges of exploratory data analysis: An interview study," 2019, *arXiv:1911.00568*.

[87] S. Alspaugh, N. Zokaei, A. Liu, C. Jin, and M. A. Hearst, "Futzing and moseying: Interviews with professional data analysts on exploration practices," *IEEE Trans. Visual. Comput. Graph.*, vol. 25, no. 1, pp. 22–31, Jan. 2019.

[88] A. Batch and N. Elmqvist, "The interactive visualization gap in initial exploratory data analysis," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 1, pp. 278–287, Jan. 2018.

[89] S. Chattopadhyay, I. Prasad, A. Z. Henley, A. Sarma, and T. Barik, "What's wrong with computational notebooks? Pain points, needs, and design opportunities," in *Proc. 2020 CHI Conf. Hum. Factors Comput. Syst.*, Apr. 25–30, 2020, pp. 1–12.

[90] T. Kluyver *et al.*, "Jupyter notebooks – A publishing format for reproducible computational workflows," in *Proc. 20th Int. Conf. Electron. Publ.*, F. Loizides and B. Scmidt, Eds. Bristol, UK: IOS Press, Jun. 7–9, 2016, pp. 87–90.

[91] RStudio. (2016). *R Markdown*, Accessed: Feb. 2, 2024. [Online]. Available: https://rmarkdown.rstudio.com

[92] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011.

[93] L. Buitinck *et al.*, "API design for machine learning software: experiences from the scikit-learn project," *ECML PKDD Workshop Lang. Data Min. Mach. Learn.*, pp. 108–122, Sep. 23–27, 2013.

[94] H. Wickham *et al.*, "Welcome to the tidyverse," *J. Open Source Softw.*, vol. 4, no. 43, Art. no. 1686, Nov. 2019.

[95] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*. New York, NY, USA: Springer, 2016. [Online]. Available: https://ggplot2.tidyverse.org

[96] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, "Wrangler: Interactive visual specification of data transformation scripts," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, May 7–12, 2011, pp. 3363–3372.

[97] P. J. Guo, S. Kandel, J. M. Hellerstein, and J. Heer, "Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts," in *Proc. Annu. ACM Symp. User Interface*, Oct. 16–19, 2011, pp. 65–74.

[98] Microsoft. (2023). *Data Wrangler Extension*, Accessed: Feb. 2, 2024. [Online]. Available: https://github.com/microsoft/vscode-data-wrangler

[99] J. Schmidt and T. Ortner, *Visualization in Notebook-Style Interfaces*. Eindhoven, The Netherlands: The Eurographics Association, 2020.

[100] I. Drosos, T. Barik, P. J. Guo, R. DeLine, and S. Gulwani, "Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists," in *Proc. 2020 CHI Conf. Hum. Factors Comput. Syst.*, Apr. 25–30, 2020, pp. 1–12.

[101] *Streamlit – A faster way to build and share data apps*. (2023). Accessed: Feb. 2, 2024. [Online]. Available: https://streamlit.io

[102] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 90–95, May 2007.

[103] M. B. Kery, D. Ren, F. Hohman, D. Moritz, K. Wongsuphasawat, and K. Patel, "mage: Fluid moves between code and graphical work in computational notebooks," in *Proc. 33rd Annu. ACM Symp. User Interface Softw. Technol.*, Oct. 20–23, 2020, pp. 140–151.

[104] A. Satyanarayan and J. Heer, "Lyra: An interactive visualization design environment," *Comput. Graph. Forum*, vol. 33, no. 3, pp. 351–360, Jun. 2014.

[105] Z. Liu *et al.*, "Data Illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring," in *Proc. 2018 CHI Conf. Hum. Factors Comput. Syst.*, Apr. 21–26, 2018, pp. 1–13.

[106] A. Lex, N. Gehlenborg, H. Strobelt, R. Vuillemot, and H. Pfister, "UpSet: Visualization of intersecting sets," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 12, pp. 1983–1992, Dec. 2014.

[107] C. Nobre, D. Wootton, L. Harrison, and A. Lex, "Evaluating multivariate network visualization techniques using a validated design and crowdsourcing approach," in *Proc. 2020 CHI Conf. Hum. Factors Comput. Syst.*, Apr. 25–30, 2020, pp. 1–12.

[108] Y. Ding *et al.*, "reVISit: Supporting scalable evaluation of interactive visualizations," in *Proc. 2023 IEEE Vis. Vis. Anal.*, Oct. 21–27, 2023, pp. 31–35.

[109] H. Stitz, T. Klaver, S. Verhoeven, M. van Meersbergen, P. Pawar, and M. Streit. (2019), *Provenance Core*, Accessed: Feb. 2, 2024. [Online]. Available: https://github.com/VisualStorytelling/provenance-core/

[110] D. Gotz and Z. Wen, "Behavior-driven visualization recommendation," in *Proc. 14th Int. Conf. Intell. User Interfaces*, Feb. 8–11, 2009, pp. 315–324.

[111] R. Amar, J. Eagan, and J. Stasko, "Low-level components of analytic activity in information visualization," in *Proc. IEEE Symp. Inf. Vis., 2005*, Oct. 23–25, 2005, pp. 111–117.

[112] A. Sarikaya and M. Gleicher, "Scatterplots: Tasks, data, and designs," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 1, pp. 402–412, Jan. 2018.

[113] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. 17th Int. Conf. Data Eng.*, Apr. 2–6, 2001, pp. 421–430.

[114] S. Gratzl, A. Lex, N. Gehlenborg, H. Pfister, and M. Streit, "LineUp: Visual analysis of multi-attribute rankings," *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 12, pp. 2277–2286, Dec. 2013.

[115] S. Carpendale, "Evaluating information visualizations," in *Information Visualization: Human-Centered Issues and Perspectives*, A. Kerren, J. T. Stasko, J.-D. Fekete, and C. North, Eds. Berlin, Germany: Springer, 2008, pp. 19–45.

[116] Z. Wang, J. Ritchie, J. Zhou, F. Chevalier, and B. Bach, "Data comics for reporting controlled user studies in human-computer interaction," *IEEE Trans. Vis. Comput. Graph.*, vol. 27, no. 2, pp. 967–977, Feb. 2021.

[117] P. Dragicevic, "Fair statistical communication in HCI," in *Modern Statistical Methods for HCI*, J. Robertson and M. Kaptein, Eds. Cham, Switzerland: Springer International Publishing, 2016, pp. 291–330.

[118] G. Cumming, *Understanding the New Statistics: Effect Sizes, Confidence Intervals, and Meta-Analysis*. New York, New York, USA: Routledge, 2011.

[119] C. Nobre, D. Wootton, Z. Cutler, L. Harrison, H. Pfister, and A. Lex, "reVISit: Looking under the hood of interactive visualization studies," in *Proc. 2021 CHI Conf. Hum. Factors Comput. Syst.*, May 8–13, 2021, pp. 1–13.

[120] R. Borgo *et al.*, "Crowdsourcing for information visualization: Promises and pitfalls," in *Evaluation in the Crowd: Crowdsourcing and Human-Centered Experiments*, D. Archambault, H. Purchase, and T. Hoßfeld, Eds. Cham, Switzerland: Springer International Publishing, 2017, pp. 96–138.

[121] K. Furmanova *et al.*, "Taggle: Combining overview and details in tabular data visualizations," *Inf. Vis.*, vol. 19, no. 2, pp. 114–136, Apr. 2020.

[122] Y. Liu, A. Kale, T. Althoff, and J. Heer, "Boba: Authoring and visualizing multiverse analyses," *IEEE Trans. Vis. Comput. Graph.*, vol. 27, no. 2, pp. 1753–1763, Feb. 2021.

[123] C. Niederer, H. Stitz, R. Hourieh, F. Grassinger, W. Aigner, and M. Streit, "TACO: Visualizing changes in tables over time," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 1, pp. 677–686, Jan. 2018.

[124] E. Mathieu *et al.*, "Coronavirus pandemic (COVID-19)," *Our World in Data*, 2020. [Online]. Available: https://ourworldindata.org/coronavirus.

[125] Plotly Technologies Inc. (2015). *Plotly*, Accessed: Feb. 2, 2024. [Online]. Available: https://plot.ly

[126] Utah Avalanche Center. (2023). "All observations," Accessed: Feb. 2, 2024. [Online]. Available: https://utahavalanchecenter.org/observations

[127] T. Manz. (2023). *Anywidget*, Accessed: Feb. 2, 2024. [Online]. Available: https://anywidget.dev.

[128] A. C. Bart *et al.* (2023). "CORGIS: The collection of really great, interesting, situated datasets," Accessed: Feb. 2, 2024. [Online]. Available: https://corgis-edu.github.io/corgis

[129] S. G. Hart and L. E. Staveland, "Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research," in *Advances in Psychology*, P. A. Hancock and N. Meshkati, Eds. Amsterdam, The Netherlands: North-Holland, 1988, vol. 52, pp. 139–183.

[130] S. S. Sawilowsky, "New effect size rules of thumb," *J. Mod. Appl. Stat. Methods*, vol. 8, pp. 597–599, Nov. 2009.

[131] W. Epperson, V. Gorantla, D. Moritz, and A. Perer, "Dead or alive: Continuous data profiling for interactive data science," *IEEE Trans. Vis. Comput. Graph.*, vol. 30, no. 1, pp. 197–207, Jan. 2024.

[132] J. Wood, A. Kachkaev, and J. Dykes, "Design exposition with literate visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 759–768, Jan. 2019.

[133] A. Mathisen, T. Horak, C. N. Klokmose, K. Grønbæk, and N. Elmqvist, "InsideInsights: Integrating data-driven reporting in collaborative visual analytics," *Comput. Graph. Forum*, vol. 38, no. 3, pp. 649–661, Jul. 2019.

[134] P. Vanhulst, F. Evequoz, R. Tuor, and D. Lalanne, "A descriptive attribute-based framework for annotations in data visualization," in *Comput. Vis. Imaging Comput. Graph. Theory Appl.*, D. Bechmann *et al.*, Eds. Cham, Switzerland: Springer International Publishing, 2019, pp. 143–166.

[135] N. McCurdy, J. Gerdes, and M. Meyer, "A framework for externalizing implicit error using visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 925–935, Jan. 2019.

[136] H. Lin, D. Akbaba, M. Meyer, and A. Lex, "Data hunches: Incorporating personal knowledge into visualizations," *IEEE Trans. Vis. Comput. Graph.*, vol. 29, no. 1, pp. 504–514, Jan. 2023.

[137] V. Le and S. Gulwani, "FlashExtract: A framework for data extraction by examples," *ACM SIGPLAN Not.*, vol. 49, no. 6, pp. 542–553, Jun. 2014.

[138] A. M. McNutt, "No grammar to rule them all: A survey of JSON-style DSLs for visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 29, no. 1, pp. 160–170, Jan. 2023.

[139] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-Lite: A grammar of interactive graphics," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 341–350, Jan. 2017.

[140] A. Head, F. Hohman, T. Barik, S. M. Drucker, and R. DeLine, "Managing messes in computational notebooks," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, May 4–9, 2019, pp. 1–12.

[141] B. A. Nosek, C. R. Ebersole, A. C. DeHaven, and D. T. Mellor, "The preregistration revolution," *Proc. Natl. Acad. Sci. USA*, vol. 115, no. 11, pp. 2600–2606, Mar. 2018.