Albireo: An Interactive Tool for Visually Summarizing Computational Notebook Structure

John Wenskovitch*, Jian Zhao†, Scott Carter†, Matthew Cooper†, and Chris North*
*Virginia Tech †FX Palo Alto Laboratory

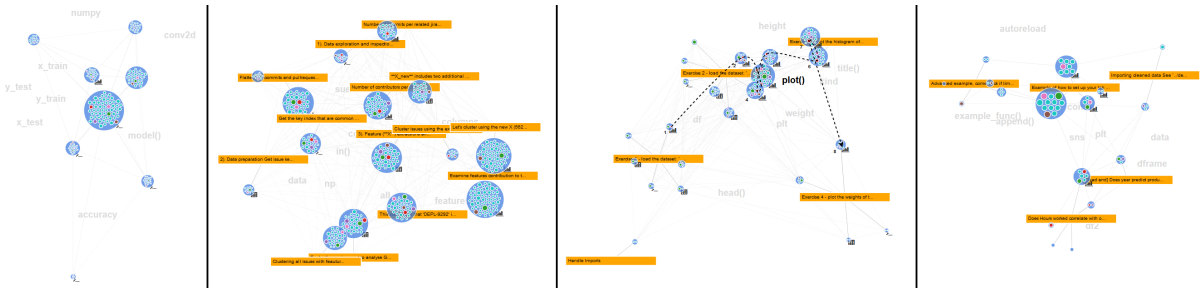

Figure 1: Albiero generates visual summaries of computational notebooks, such as the four shown above. Markdown (orange rectangles) and code cells (blue circles) are encoded into an interactive force-directed graph, which is further enhanced by additional features such as background annotations to label the space and paths tracing the use of variables and functions.

Abstract

Computational notebooks have become a major medium for data exploration and insight communication in data science. Although expressive, dynamic, and flexible, in practice they are loose collections of scripts, charts, and tables that rarely tell a story or clearly represent the analysis process. This leads to a number of usability issues, particularly in the comprehension and exploration of notebooks. In this work, we design, implement, and evaluate Albireo, a visualization approach to summarize the structure of notebooks, with the goal of supporting more effective exploration and communication by displaying the dependencies and relationships between the cells of a notebook using a dynamic graph structure. We evaluate the system via a case study and expert interviews, with our results indicating that such a visualization is useful for an analyst's self-reflection during exploratory programming, and also effective for communication of narratives and collaboration between analysts.

Keywords: Software visualization, computational notebooks, provenance analysis, insight communication.

1 Introduction

Computational notebooks (e.g., Jupyter Notebook [27]) have become a major medium for data exploration and insight communication in data science. These notebooks combine code, documentation, and output in a variety of forms (e.g., charts, tables, and images) within a single document. Notebooks are expressive, dynamic, and flexible, providing rich support for exploring and sharing ideas. Notebooks have also been extended to support automated creation via templates and scheduling, the injection of different parameter sets, and container execution [36]. The content of a notebook is broken down into *cells*, including *markdown cells* for documentation and *code cells* for scripts. Unlike traditional code execution, these cells can be executed in any order.

However, these computational notebooks, originally intended to be digital narratives [25], are often loose collections of scripts, charts, and tables that rarely tell a story or clearly represent the analysis process. Recent studies [23, 29] found that analysts use these notebooks in a variety of ways, including sharing results, quick experiments, prototyping code for inclusion in later pipelines, and data analysis.

Both the popularity of computational notebooks, as well as the variety of ways that they are used by data scientists, present a number of challenges. Individual analysts may create multiple versions of similar but not identical cells or entire notebooks to test small changes in how data are collected, cleaned, and processed. They then struggle to keep track of which version of the code produced specific results, presenting a navigational challenge [17, 35]. Additionally, the very act of using these notebooks for undirected, exploratory data analysis [16] conflicts with the linear structure of the notebook. These greatly increase the difficulty of following and explaining the ideas in the notebook, especially in collaborative work settings.

Further, temporary experiments are treated as "throw-away" code that is poorly annotated or documented [19], yet this may still end up in the final notebook. Therefore, analysts who are collaborating with others on a single notebook may struggle to make sense of the high-level structure of an existing notebook [34], potentially leading to slower development and/or the re-creation of already-implemented analysis. For example, one analyst may be unfamiliar with the conventions of the other, requiring additional assistance to understand the purpose and structure of a notebook that has been shared with them.

The above challenges motivate the need for a tool to facilitate the exploration and communication of analyses presented in computational notebooks. Visualizations have been proven effective in helping analysts to record computational flows [3, 9] and reasoning processes [10, 15], i.e., supporting the *provenance* of data analysis [28]. In this work, we describe the design, implementation, and evaluation of a visualization assistant, Albireo, to complement standard computational notebook systems. Albireo supports the exploration of notebooks at multiple levels, communicating both a high-level overview of a notebook as well as permitting deeper exploration of structures and cell relationships through interaction. In particular, we note the following contributions:

1. An analysis of difficulties with existing notebook software via expert interviews, which led to a set of tasks that motivated the features of Albireo.

*{jw87, north}@vt.edu.
†{zhao, carter, cooper}@fxpal.com; J. Zhao is the correspondence author.

2. The design and implementation of Albireo, an interactive visualization assistant that complements traditional computational notebook software.
3. A demonstration of the usefulness of this system via a two-part case study, followed by an evaluation via a set of interviews with experienced notebook users.

We found that our expert evaluators responded generally positively towards Albireo, and they were able to draw insights from explorations of their notebooks than they did not notice in the linear notebook view.

2 Background

2.1 Computational Notebooks

Computational notebooks are programming environments that support interactive, iterative development of software. While Jupyter Notebooks [27] are anecdotally the most common, other tools such as Databricks [8], Apache Zeppelin [2], and Sage Notebooks [30] all share a common set of data-centric development features. Of particular note to the visualization that we present is the structural breakdown into blocks referred to as *cells*.

These cells appear in two primary forms. Markdown cells contain formatted text that is often used to provide context to the accompanying code, supplementing traditional code comments with discussion of the functionality, structure, and/or results of nearby code. In contrast, code cells are executable components, performing computations as any other executable file would, though limited by the bounds of the cell itself. After successfully executing, these code cells also contain formatted output of various types: text, table, graph, image, video, and more.

Though a notebook appears to be intended to run linearly through the cells, this sequencing is not enforced. An analyst may create a notebook that is entirely code cells (or markdown cells) executed in arbitrary order, potentially with some cells run multiple times, edited between executions, and with potentially multiple nonlinear execution paths within the notebook [29]. Indeed, analysts use these notebooks for a variety of purposes, including experimenting with alternate techniques, sharing results, temporary code, and exploratory data analysis [23]. This range of uses for computational notebooks naturally leads to communication challenges when sharing notebooks with other analysts.

2.2 Software Visualization

Software visualization is a maturing field, with a number of techniques proposed to provide visual overviews of functional and structural components of existing code. Unified Modeling Language (UML) is the standard approach, consisting of numerous views and diagrams to describe software components, the interactions between them, execution sequences, data propagation, and many more [4]. UML renders the structure of code at natural divisions such as classes or modules, similar to but structurally different from the cells of a computational notebook. Beyond the UML standard, Caserta and Zendra provide a survey of existing 2D and 3D visualization techniques for static software [7].

As with our visualization approach, many published methods supplement the code to provide greater developer comprehension. For example, Hoffswell *et al.* provide in situ visualizations within the code itself to summarize variable values and distributions [21]. Similar to our work, Seider *et al.* use a force-directed scheme to visualize modules and dependencies in code [31]. Likewise, dependency graph techniques have been adopted for relationships between variables and functions [14]. Other tools provide a visualization aid to assist a developer in understanding code structure [33, 37]. Still others focus on collaboration between analysts, such as the collaborative code review tool CFar [20].

While these existing techniques are sufficient for traditional software, computational notebooks include additional challenges and behaviors that are not supported by existing tools (among others, these include the previously mentioned overall cellular structure and variable execution order). Indeed, the difficulties inherent in comprehending notebooks were recognized and have been addressed for standard paper notebooks as well [35]. With respect to computational notebooks, some work has been done to address these challenges, such as cell versioning in Variolite to replace the tendency of analysts to create a sequence of cells with slight parameter changes to test analysis alternatives [22], and gathering relevant segments of code for a particular outcome [18]. However, there is no other broader software visualization work specific to computational notebook challenges.

2.3 Provenance Visualization

As our goals are to support effective exploration and communication of the analysis processes contained in computational notebooks, the design of Albireo is influenced by the body of research on provenance, which is the trace of the changes and advances during the analysis. Ragan *et al.* provide an organizational framework that outlines five types and six purposes of provenance information in visualization design [28]. In particular, this work is related to *data* and *insight* types and the purposes of *recall*, *presentation*, and *collaborative communication*.

Numerous visual analytics tools have been proposed to visualize these types of provenance and facilitate the above purposes. For example, VisTrails is one of the earliest visualizations that formally focuses on maintaining a pipeline of the analytical activities generated from exploring raw data in scientific workflows [3]. The HARVEST system derives hierarchical semantic actions of users in analyzing business and finance data to support insight provenance [15]. In synchronous collaboration, Mahyar and Tory explore how provenance information is used in team work and develop CLIP to reveal relationships and awareness of users' findings [24]. Recently, in asynchronous collaboration, KTGraph employs a graph visualization and a timeline widget to facilitate "hand-offs" between users [39]. However, few studies have explored how to support provenance in exploratory programming, especially with the new computational notebook environment that is rising in data science. In this paper, we develop Albireo towards offering an effective means, a visualization assistant, for analysts to explore and communicate the complicated materials in notebooks.

To better show provenance by revealing clusters, trends, and relationships for a set of unstructured objects, similarity measures and projection techniques are widely used in visualization. As the cells in a notebook are loosely connected, we employ a "proximity $\approx$ similarity" metaphor to facilitate the exploration of notebook content. We adopt the standard seen in similar tools, in which the spatial positioning of an observation relative to others in the population communicates a similarity or dissimilarity measure. This is seen in tools designed for both quantitative data [6, 11, 32, 38] and text data [5, 13]. Indeed, previous work has shown that human analysts prefer to make use of space in order to organize and synthesize data [1, 12], thereby supporting their personal sensemaking process [26]. We adopt this similarity projection approach to reveal the relationships between elements of a computational notebook in an interactive and flexible manner.

3 Task Elicitation via Expert Interviews

To determine the task requirements for this project, we employed an iterative, user-centered design process. We began by performing semi-structured interviews with three self-reported frequent notebook users (these participants are referred to here as P1–P3). These three subjects each reported more than 10 years of experience with computational data processing, as well as several years of experience with Jupyter notebooks specifically. Their workflows included both single-user and collaborative notebook

development, and their reasons for using notebooks spanned the use cases identified by previous studies [23, 29], including prototyping code, sharing both code snippets and results, and exploratory data analysis. As the development process continued, we consulted these experts several more times, during which we discussed our design, presented the current prototype state, and collected their feedback. From these interviews, we uncovered the following four interrelated tasks that could be supported by a visualization assistant:

T1: Summarize Notebook Content. We identified a common need for a method to obtain an overview or summary of a notebook. The study performed by Kery *et al.* identified the same issue, with their participant IP01 noting "*I can't get an overview of what's going on in my notebook... it's just a lot of stuff*" [23]. This overview addresses one comment frequently noted in these interviews: while markdown cells are a suitable method for explaining the contents of a notebook, they take time to create and are often too much effort for a notebook that may only see temporary use.

P1 noted that he would create only code cells during development, saving markdown cells for a presentation version of the notebook only created after all of the code was functional. P2 stated a preference for using standard Python comments within code cells rather than taking the time to create markdown cells, even though he works collaboratively on notebooks with other data analysts. It seems that unless the notebook is adapted to share results in a professional venue, markdown documentation is not prioritized by data analysts, despite being a useful summarization and documentation feature.

T2: Provide Communication Support. Data analysts often work collaboratively on notebooks, both in parallel and in series. While there are certainly strengths to notebooks for collaboration (P2: "*sharing results is easier because they are embedded within the notebook*"), notebooks also present a deeper communication challenge than the overview level addressed by T1. A data analyst seeing a collaborator's notebook for the first time must work to understand the approach taken by the initial analyst, as well as the portions of analysis that have already been completed. This provenance information is not captured by current computational notebook software.

Each of our interview participants reported using notebooks in collaborations, and each expressed frustration with the process regardless of whether the group developed a solution simultaneously (the most frequent method of P1) or developed part of a solution and then handed the notebook off to a collaborator (P2 and P3). P3 noted that switching between datasets is often an issue when collaborators are trying to develop a single notebook to handle multiple data challenges. Similar to his statement in the previous task, P1 recognizes the importance of markdown cells for communication to describe which cells provide what functionality, but still does not create these cells until he is ready to share the notebook.

T3: Support Effective Navigation. The design of notebooks to serve as computational narratives [25] directly leads to the fact that notebooks can become quite long. The linear structure enforces a one-dimensional ordering of cells, despite the fact that two neighboring cells may be functionally independent. Tracing a debugging issue through the notebook requires frequent scrolling up and down to understand both the contents of the overall notebook and the relationships between the cells. Even with such actions, an issue with a single variable will likely not be obvious.

Rather than dealing with this issue directly, data analysts come up with their own workarounds. P2 reports that in order to test a variety of learning models, he will create a set of notebooks that differ in only one or two cells. This results in his management of a complex local folder structure to manage his projects, but he finds that navigating this folder structure less of a challenge than navigating a single massive notebook. P1 is more comfortable with copying an existing cell to try a new version of existing code, but

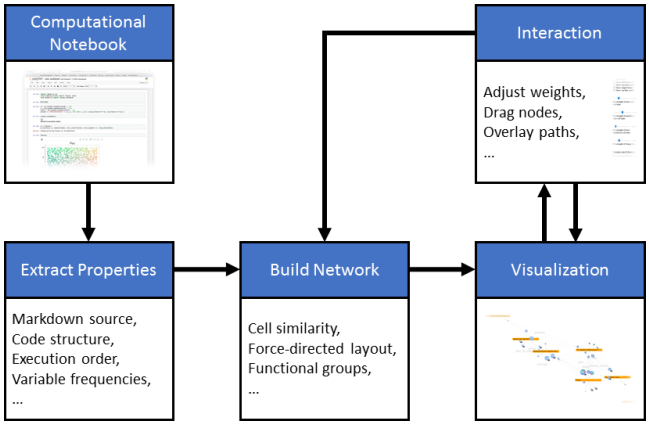

Figure 2: High-level architecture of the Albireo system.

notes that this results in difficulty identifying the important structures in the notebook and the ideal execution path through a notebook. P3 started with R notebooks and had previous Python experience, but struggled to transition to Python notebooks due in part to the lengthy and unwieldy format.

T4: Facilitate Nonlinear Development. Building on the linear structure issue from T3, our interviews showed that exploratory data analysis is a nonlinear process that is being forced into the linear structure of notebooks. Indeed, P2 noted that he regularly skips cells during development, often executes cells out of the linear notebook order, and reorganizes cells frequently during development. Development often involves testing various learning models or analysis methods in separate cells that have no dependency to each other, and yet the linear notebook structure implies such a dependency. The ability to execute cells out of the notebook sequence can also cause unexpected behavior to occur during development and testing due to the persistence of variable values, yet as P1 pointed out, no cell or variable dependency tool is provided beyond the implied linear structure, despite the fact that tracing the use of a variable is a standard debugging operation. He reports being frustrated that no support exists for "*integrating good debugging beyond print statements.*" P3 has become frustrated enough with hidden variable states not conforming to his mental model of the code structure; he now only uses notebook technology for temporary or prototype code that is later incorporated into standalone systems.

4 Albireo System Design

4.1 Architecture and System Overview

We address the aforementioned tasks with a visualization solution, with the design heavily influenced by the comments of our interviewees. In particular, P2 noted that it would be useful to have a cell dependency graph, saying that "*understanding the execution order of cells out-of-sequence is often a critical issue*" during development and debugging. P3 suggested a need to visualize larger functional structures within the notebook as well, a sentiment echoed by P2 during a later interview when suggesting that the cell dependency graph could more broadly communicate functionality dependencies. There was hesitation to visualize structures smaller than the cell granularity level, as both P1 and P3 observed that the division of code cells often follows a natural semantic segmentation of the problem they are trying to solve, and that the relationship and dependence of variables is often more important to understand across multiple cells rather than within a single cell. We also acquired some more general visual and structural feedback from analysts who were not necessarily notebook users during an internal poster session near the end of the development process.

Figure 3: Five different types of output badges: **(A)** text output, **(B)** chart, **(C)** table, **(D)** no output, **(E)** file output. The size difference of the nodes corresponds to the frequency with which the cell components are used in the overall notebook.

Given these issues, we developed a visualization assistant to complement standard notebook software. This visualization would better show relationships and similarities between cells, and would enable an analyst to glimpse the structure of an entire notebook in a single view. Both P1 and P3 stressed this difficulty, especially when working to restructure an existing notebook for presentation. Figure 2 shows an architectural overview of the Albireo system. Taking a computational notebook as input, we extract each markdown and code cell. We extract a collection of properties, including data stored in the notebook (e.g., execution order) and properties computed by processing the notebook (e.g., frequency of each variable). Each cell is then embedded as a node in a force-directed graph visualization, with these properties influencing the structure of the graph. The graph and its properties are discussed in the next subsection.

4.2 Visualization and Interaction Features

Here, we discuss both the visualization and interaction design and features of Albireo at three different levels of the visualization. We begin with a description of the types of nodes and their properties, which are mapped to the notebook at the cell level. This is followed by a discussion of some of the variable-level features that are extracted from the individual cells. Finally, we bring these nodes and features together into the full dynamic graph structure, shown with the accompanying interface in Fig. 5.

4.2.1 Cell-Level Properties

As with notebooks, we create two types of nodes in Albireo: markdown nodes and code nodes, to visually summarize the contents in notebooks at the cell level (T1). Because of the substantial difference in the meaning and usage of markdown and code cells in notebooks, we render a difference in the visual design of the nodes: markdown cells are rendered as orange rectangular nodes, while code cells are designed as blue circular nodes.

The markdown nodes display the first few words of text from that cell in the notebook, allowing an analyst to glean the purpose of nearby code cells. Each of the markdown cells are represented as identically-sized nodes, with the intention of these nodes serving as labels or control points for sets of subsequent code cells that comprise higher-level functions (e.g., data loading, feature construction, and pre-processing).

For code nodes, their area is roughly mapped to the "complexity" of the code in the cell. This complexity measure and size is the result of a circle-packing technique described in Sec. 4.2.2. Additionally, an output badge in the lower-right region of code nodes indicates the type of output produced by that cell, as shown in Fig. 3. The goal of these badges is to provide a quick visual reference to these code cells, allowing an analyst to identify regions of the visualization that correspond to file output, charts, tables, and text in the notebook.

Mousing over the markdown node provides the full source text for the associated cell. Likewise, hovering over the background of a code node displays the source code for the matching cell. With the above visual representation and interaction of notebook cells, an analyst exploring a shared notebook for the first time can browse these markdown and code nodes to determine high-level information about the notebook (T1).

4.2.2 Variable-Level Properties

Summarization at the cell level alone is not enough for analysts to gain an informative big picture of a complicated notebook (T1). Thus, in each code cell, we extract all of the variable names and function calls, and also tokenize each of the string literals (this collection of extracted features is referred to hereafter as "components" of the cell). As our expert interviewees only expressed interest in viewing information more fine-grained than the cell level when it conveyed information about the relationship between cells, we filter this collection to only include the components that are used in more than one code cell. The collection is also filtered for stop words. We further compute the frequency with which each of the components appear in the entire notebook, and map that frequency to the area of a collection of component sub-nodes.

Each of these sub-nodes is then placed within its related code nodes via a circle-packing technique. The five most frequently-used components are also encoded with a unique color, further drawing the attention of an analyst to these components (see Fig. 4 for an example). We limit this color encoding to the top five components as a balance between indicating many of the most common components while also not introducing too much color variety. Users are also provided with an option to remove these component sub-nodes from the display, in order to reduce visual clutter.

In addition to variable-level visualization, Albireo allows an analyst to navigate a notebook based on selected variables of interest. When a user clicks on one of these component sub-nodes, an animated marching ants-style path is drawn that connects each of the cells in which that component appears. The path is drawn in the order that the cells appear in the notebook, and a numerical label is added near each of the nodes indicating this order (Fig. 6). This could be useful if, for example, an analyst exploring a collaborator's notebook wishes to manipulate the `iris` dataset, and hence is interested in seeing where the initial analyst has already used that dataset. By highlighting a path for that variable through the notebook, the current usage of that variable can be quickly identified (T3). If an analyst is searching for a specific component to inspect, an alphabetized list is provided in the dropdown seen at the bottom-right of Fig. 5.

Displaying these paths also permits a data scientist to visually debug some issues with their code (T4). For example, variables such as `i` and `x` are frequently used as counters. If a data scientist forgets to reset that counter between cells, unexpected behavior can result. This unexpected behavior is compounded in computational notebooks because the data scientist can choose to execute cells out of order. Using the variable path allows for a quick summary of the code cells that should be inspected for a missing counter reset.

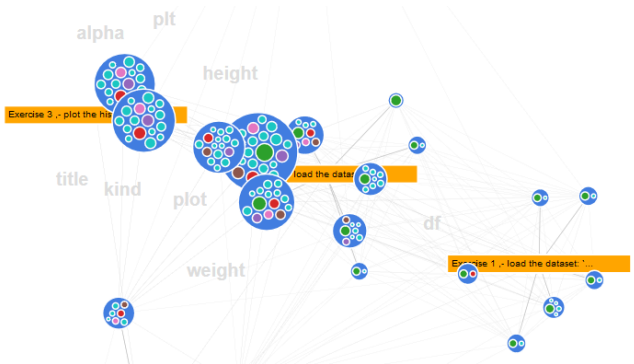

Figure 4: Cell nodes include the collection of common variables, functions, and string literals used in the corresponding code cell. The frequency of each component within the full notebook is indicated using both area and color.

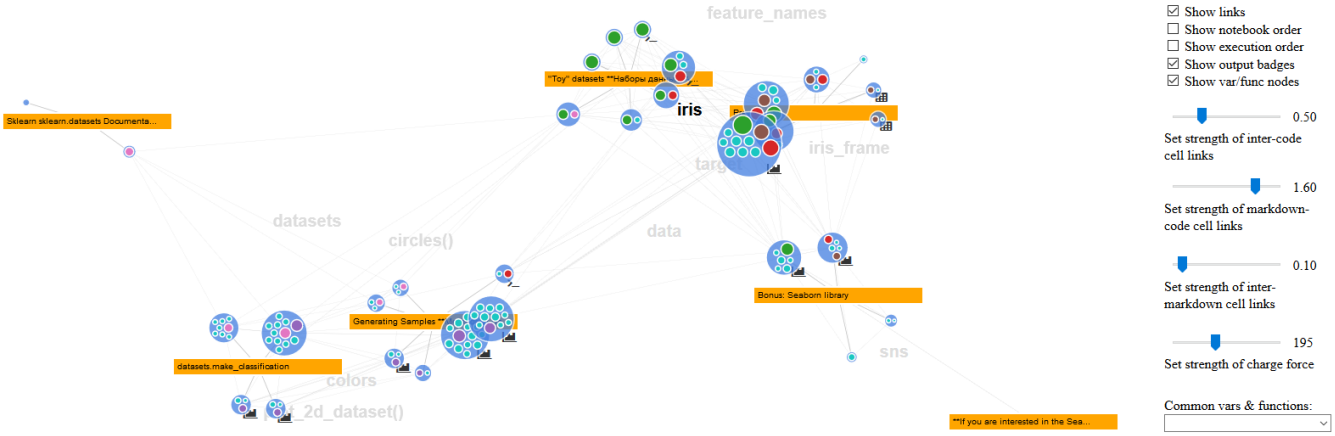

Figure 5: The Albireo system interface, including a large graph exploration space to the left and a panel of controls to the right. The markdown (orange rectangles) and code cells (blue circles) are positioned in a force-directed graph, with resting edge lengths representing the similarity of two cells. Individual variables, function calls, and string components are visualized as sub-nodes within the code cells, and are also used as background annotations to tag the space.

Finally, we select the top ten most frequent components and position them as annotations in the background of the exploration space, providing a contextual summarization of notebook cells (T1). Like the markdown and code nodes, these are also positioned by force-directed means, but without visible links connecting the annotations to the rest of the graph. This provides contextual information to the analyst, allowing them to quickly see which portion of the graph corresponds to what functionality of the code. The rationale behind selecting the top ten was again a balance between providing extra information to a data scientist while also minimizing visual clutter. A mouseover interaction on these annotations renders the text in black rather than in the default gray, which was selected to indicate their secondary importance.

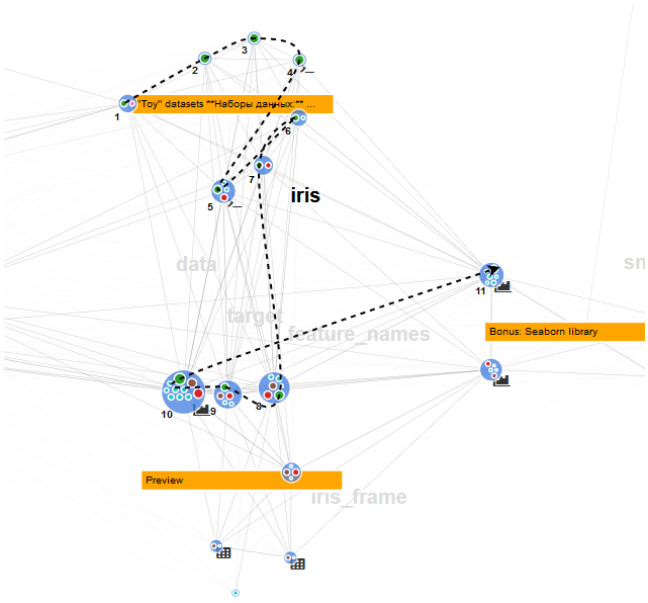

Figure 6: An example path connecting all cells that use the variable `iris`. The `iris` annotation in the background has also been highlighted via a mouseover interaction.

4.2.3 Notebook-Level Properties

After creating each of the nodes and extracting their properties, the full set is placed into an interactive force-directed graph. The graph aims to provide a high-level summarization of the structure of a notebook (T1) and facilitate the communication between analysts with semantics (T2). Because cells in a computational notebook can be executed out of linear order, we determined that such an undirected graph is more appropriate than a more standard directed dependency graph. We note that this does lead to a visual tradeoff in which we lose some structure and ordering of the cells which can lead to difficulty in tracing some execution paths, though gaining the benefit of more clearly showing relationships between cells that may be spread throughout the notebook. Within this graph, node proximity indicates similarity (similar nodes are pulled close together, dissimilar nodes are widely separated). We compute the similarity between two code nodes as the number of components shared by the pair, with the pair of cells sharing the greatest number of these components mapped to the strongest link strength in the graph. However, other alternative approaches towards computing these similarities could include (but are not limited to) allowing users to specify strengths, identifying components that are unique to certain cell pairs, taking into account the distance between cells in the linear notebook structure, and using a deep learning approach to identify underlying cell commonalities. In this prototype, we opted for an efficient similarity computation in order to better support interactivity within the visualization.

We also draw links between the nodes in the graph by default, though these too can be removed by a user to reduce visual clutter. The width and opacity of the links are mapped to the measured similarity of the node pairs that they connect. The length of the link is also mapped to this measured similarity, but the non-optimal layout of the force-directed computation can make the rendered length imprecise, justifying the dual encoding.

We define three types of links in the graph: code–code links, code–markdown links, and markdown–markdown links. Each type of link has an associated weight, which allows a data scientist to update the graph layout to enhance certain structures. In the current implementation, fine-grained control of these weights is permitted by a set of sliders in the control panel, but these controls can be simplified into a list of useful views from which a data scientist can select. For example, in the top view of Fig. 7, the user has increased the weight applied to the code–code links and decreased the weight on code–markdown links. As a result, the global relationship

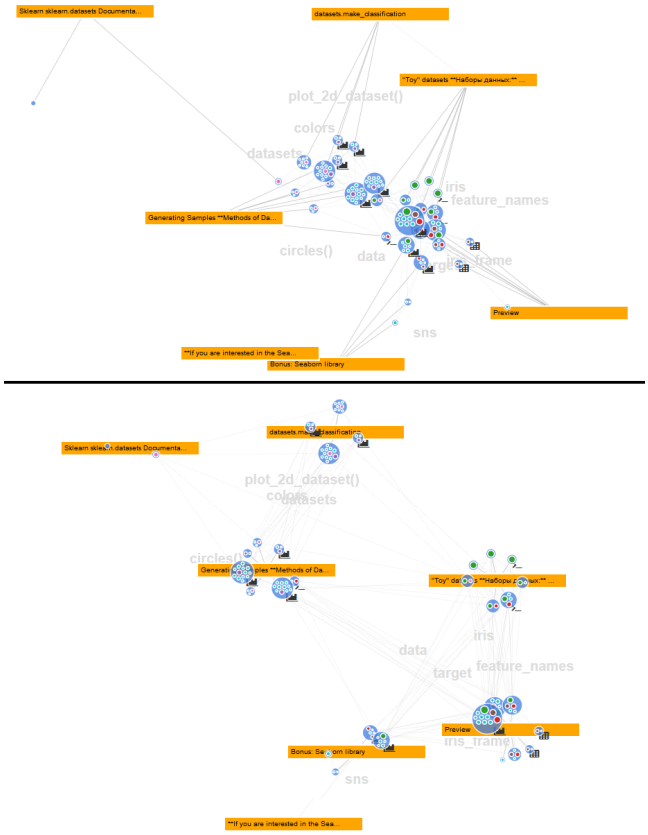

Figure 7: Restructuring the graph exploration space to highlight similarity between code cells by increasing the weight on code–code links and decreasing the weight on code–markdown links *(top)*, and to highlight high-level structures in the notebook by increasing the weight on code–markdown links and decreasing the weight on code–code links *(bottom)*.

between code cells is treated as the most important feature of the visualization. With such a view, a data scientist could identify if a cell might be better positioned linearly within the notebook, as such a node will be pulled away from its corresponding markdown node and towards the code nodes of another group.

In contrast, the bottom view of Fig. 7 is produced by the analyst increasing the weight applied to the code–markdown links and decreasing the weight applied to the code-code links. As a result, the higher level functional groups of cells within the notebook become more visible in the structure of the graph, while the relationship between individual cells in the notebook is lost. With this view, an analyst can quickly identify these high-level functional structures within a notebook. Increasing the strength of the markdown–markdown links will pull similar structures closer together, indicating functional relationships or interdependencies between these structural groups.

In addition to the optional variable paths mentioned previously, we provide two more animated paths through the graph: one that connects the cells in linear order through the notebook, and one that connects the cells in execution order. These paths allow a data scientist to map the structure of the graph to the structure of the notebook, and can further assist with visual debugging and non-linear development (T4). For example, combining a variable path with the execution order path narrows the search space for a missing counter reset by eliminating code cells that have not been executed yet.

5 Case Study

In this section, we briefly demonstrate the utility of Albireo via a use case with two parts, based on a real-world notebook retrieved from the repository collected by Rule *et al.* [29]. In the first part, a data scientist Alice has created a `scikit-learn` tutorial notebook for a collaborator Bob, who is inexperienced with both notebooks and machine learning. She is trying to track down a bug in her notebook with the assistance of Albireo. In the second part, she has now sent the working notebook to Bob, who seeks to understand its components and functionality.

5.1 Identifying a Bug

Alice has been given responsibility for training Bob on the use of `scikit-learn` for a significant company project. She has developed a tutorial notebook to walk Bob through the features of the library, including classification and regression, as well as plotting results with `Matplotlib`. She is also demonstrating the effects of these features on a variety of datasets. The notebook she has developed is nearly finished, but she realizes that she forgot to demonstrate one of the features of the regression package earlier in the notebook. She scrolls up to an area twenty cells previous to where she was recently working, adds a few lines of code to the cell, and executes it again. Unfortunately, the plot generated by that cell is completely different from what she had expected.

Puzzled, she opens Albireo, performs a search for the cell in question, and begins to inspect the glyph. As she does so, she notes that this cell uses the counter variables `x` and `c` to iterate over some data. Though these counter variables are reused in other cells, she knows that they are always initialized in their loops. But then she recognizes that a similar variable reuse issue might be the problem. She locates and clicks on the sub-node representing the `data` variable, drawing a path for this variable through the graph (shown in Fig. 8, with the node she is inspecting at the beginning of the path). She immediately sees that this variable has been reused in several different regions of the notebook. Hovering over another cell later in the path, she realizes that she had reprocessed and overwritten part of this dataset later in the notebook for a later demonstration. Though she returned back to a previous cell and executed it out of linear order, the updated version of the `data` variable still persisted in Python, and so she saw a plot of the new data rather than the old. To prevent this issue for also happening to Bob, she elected to change the variable name during the reprocessing step.

5.2 Understanding Notebook Functionality

Bob has now received the tutorial notebook from Alice, and wants to first grasp the big-picture idea of the structure and contents of the notebook. He also opens Albireo, and restructures the graph to best view the functional groups of cells by selecting options from the control panel to emphasize code–markdown links and de-emphasize all others (shown in the lower panel of Fig. 7). He immediately can see the groups of cells in the notebook that walk through the tutorial, including among others groups for classification, data loading, and

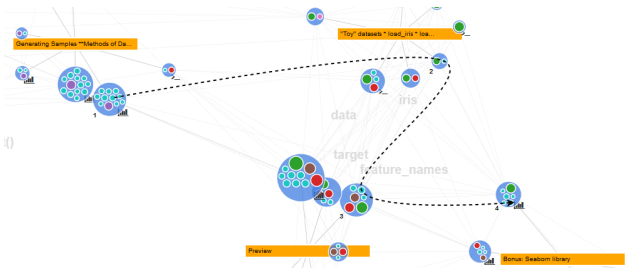

Figure 8: Showing the usage of the variable `data` within the notebook with a path overlaid on the cell nodes.

sampling. As he begins to work through the examples that Alice has provided in the notebook, he is puzzled by the operations necessary to plot some of the results. He has seen the `plot_2d_dataset()` function call in the background, but cannot find any documentation for that function online. He begins to inspect some of the cell nodes in the region of that background annotation. He quickly sees that Alice has helpfully created this function to abstract away the complexities of `MatPlotLib` plotting, so that Bob only needs to provide his data and color selection as arguments to get a basic plot.

Continuing through the notebook, Bob discovers that he also is not fully grasping the concept of data frames. He selects the `dataframe` variable from the dropdown list, but sees that it only appears in two cells. Still confused, Bob then notices the `iris_frame` variable shown in the background annotations in gray text, which is always placed next to a cluster of large code cells in different layouts. This indicates that it might be critical for analysis in this notebook (see Fig. 7). Further, scrolling down through the list further, he finds the `iris_frame` variable and selects it. He sees that Alice has used this variable to manipulate the Fisher's Iris dataset in a plotting example near the end of the notebook, and scrolls down to that region of the notebook to revisit the data frame concept in more detail. Eventually, Bob comes to understand that data frames are a central data structure in `Pandas` for storing and manipulating data.

6 Expert Interviews

To further support the first component of our case study, we conducted three interviews with frequent notebook users. These interviews lasted approximately one hour each. One of the three participants in this interview study also participated in the task elicitation interviews discussed in Sec. 3; the other two participants were interacting with the visualization for the first time during this study, though they had passing familiarity with it. Each of these participants regularly uses notebooks in their day-to-day work as machine learning and HCI researchers.

6.1 Procedure and Design

During these interviews, each study participant (denoted as *SP[x]* hereafter) was presented with side-by-side views of Jupyter and Albireo. Both systems were used simultaneously by the participant to explore a series of notebooks that the participant originally developed. In some cases, the participants had not reviewed the contents of these notebooks in months or longer. The notebooks that were explored by the participants also came in a variety of forms, ranging from roughly 10 to 100 cells and both with and without markdown cells. The interviews were semi-structured, with general topics of inquiry centered around the four tasks from Sec. 3.

6.2 Results

Here we report both our observations as well as quotations from the participants with respect to the tasks from Sec. 3.

6.2.1 Summarize Notebook Content

All three participants found different methods for summarizing the contents of a notebook that they explored using Albireo. SP1 used the annotations in the background to quickly summarize his notebook (the leftmost notebook in Fig. 1) as "*creating a convolutional model that is tested for accuracy,*" using the annotations `conv2d`, `model`, and `accuracy`. Evaluating a notebook that had no markdown cells for structure and guidance, SP2 turned off many of the visual options, and then was quickly able to explain the purposes of several groups of cells as being responsible for data loading, processing, and displaying (Fig. 9). This was aided in part by locating the annotations `hrv` and `rr_interval`, noting that these variables measured the same property. He also noted some of the structural properties of his notebook, seeing quickly that the larger nodes are the implemented functions and the smaller nodes

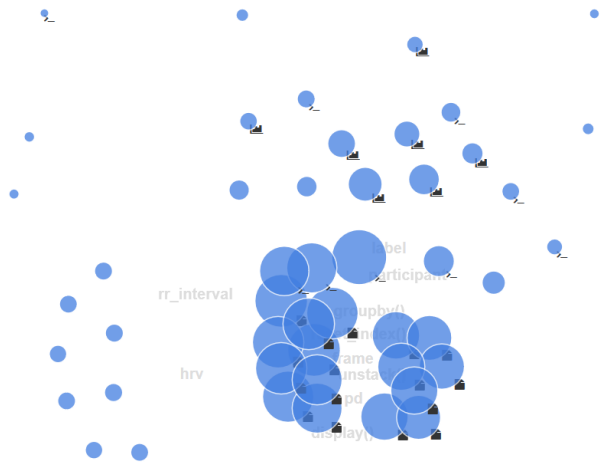

Figure 9: A notebook explored by SP2 in our expert interview study, highlighting the functional structures that this participant identified as the clusters within the graph structure.

are calling those functions. SP3 noted that she was able to quickly get an idea of the size of the notebook, noting that "*most of the cells involve a lot of computations with a lot of variables, though there are some smaller cells...*" before proceeding to identify those smaller cells as responsible for loading data. These three very different summarizations demonstrate the flexibility of Albireo as a tool to see high-level notebook structures.

6.2.2 Provide Communication Support

To investigate the qualities of Albireo with respect to collaboration and communication, we prompted our participants to demonstrate how they would explain their notebook to a colleague. SP1 and SP3 both highlighted the structure, behavior, and key results within their notebooks. SP3 specifically noted that she could imagine "*explaining to a collaborator that these four cells (circles with mouse) are most relevant to this dataset,*" and could also "*show collaborators where the results that are important are located, picking out cells to highlight early when presenting a notebook.*"

SP1 similarly noted that having the visualization would enable an analyst to quickly point to and reference individual cells, and could imagine telling a collaborator to "*hover over the big node*" and "*pick the* `model` *variable from the dropdown list and look for what it interacts with in the graph node.*" SP2 stated that the background annotations could be used to quickly comprehend an overview of the notebook that he was seeing for the first time, and that these would effectively serve as a visualization substitute for documentation in a notebook that is not well documented.

6.2.3 Support Effective Navigation

We observed that while the participants were interacting with the notebook, they were frequently performing a substantial amount of scrolling, often pausing in their verbal explanation while searching relevant information in the notebook. This presents an obvious navigation challenge with notebooks, as we discussed earlier. Though this pause-to-search behavior was also occasionally visible while seeking some information in Albireo, it was not nearly as prevalent. Further, each of the participants noted the convenience of quickly being able to quickly trace the usage of a function or variable through the notebook, identifying its usage and purpose. SP1 noted that it was "*fascinating to trace how the model was being transformed through the sequence of code cells.*" SP2 noted the convenience of following the variable-level paths from function calls to their definitions and back for debugging purposes.

6.2.4 Facilitate Nonlinear Development

Both SP2 and SP3 noted that visually identifying similarities between cells in the visualization could lead to assistance in development when testing multiple learning models in parallel. SP2 noted that he could easily "*reference a node in a functional group to locate and potentially copy for another test.*" SP3 quickly spotted two cells that were positioned close together in the visualization and identified them as cells that were both performing similar clustering analyses with slightly different models, though they were separated by two other cells in the linear notebook. Indeed, SP3 also noted that she has always "*had a linear mental model of the notebook*" but was pleasantly surprised by how easy it was to see relationships between cells that were not obvious in the notebook view.

We also noted that each of the participants learned new facts about the variables that they used in their notebooks. SP1 and SP2 both identified duplicate variables: two variables that stored the same data and performed the same purpose. Neither reported being aware of these duplicates when developing their respective notebooks. Similarly, SP3 found that she created a variable that was never used again. She loaded multiple datasets into their own `dataframe`s, but one of the `dataframe`s was never processed and used after this data loading stage. Again, she reports that she did not notice this oversight during development, but she detected it quickly with Albireo while she was tracing the usage of the other `dataframe`s through the notebook.

7 Discussion

As would be expected, both the Jupyter notebook view and Albireo have their own ingrained strengths and weaknesses. Participants were more comfortable with the Jupyter view and better able to perform tasks regarding cell dependencies and low-level functionality of code as a result, though at the expense of greater exploration time. Albireo allowed participants to better see deeper connections and functional structures, but presented both an unfamiliar interface and method for thinking about notebooks, leading to occasional frustration.

7.1 Limitations

We noted several limitations to our implementation approach for Albireo in the previous section. Most significantly, our focus on the relationships between cells is not ideal in notebooks that have significant internal functionality built into a single cell. Interview participant SP1 was unable to locate the function critical to the operation of one notebook in the dropdown list, as that function was only defined and used in a single cell. One solution to this issue could be a hierarchical display in which an analyst could select a cell to examine in more detail, thereby seeing relationships between code structures within the cell in a similar manner to what is currently displayed between cells.

Our use of frequency as a measure of importance was also criticized by several participants, who noted that the annotations and largest component circles often displayed common data reduction or processing functions, which would not be as interesting to display as variables or functions that are unique to the notebook. Constructing a balance of uniqueness and frequency in choosing the "important" components to display and highlight could resolve this issue. A similar issue was raised concerning the inclusion of strings in the components list with the variables and functions. While some strings like axis labels and column names are useful to include, others like input prompts are not. This could be resolved by examining the context of the string within the cell, making an importance decision based upon semantics. It could also be left as a choice to individual users, who could upload their own set of rules for components to display and components to suppress in the visualization.

In addition to the criticisms of the participants, we note that our expert interviews are somewhat biased, as the experts developed the visualized notebooks themselves. Some of the small clues suggested by the visualization could have been sufficient to trigger the memories of the experts to recall large purposes. This is certainly a helpful feature when a data scientist is revisiting one of their own notebooks for the first time in weeks or months, but these clues might be useless to someone else. We also note that the graph structure will occasionally present some occlusion effects, particularly when large nodes are positioned close to each other. While we offer controls to users that can overcome this by "expanding" the size of the overall graph, adding some automated overdraw prevention in the graph would reduce the interaction workload of those users.

7.2 Future Work

In addition to the extensions discussed in the previous subsection, several possibilities exist to further augment the functionality of Albireo. We previously noted that the final vision of this tool would ideally be implemented as a plugin to Jupyter or other computational notebook software. Such a plugin could then more efficiently implement brushing and linking features, connecting interactions with the nodes in the graph to the code cells in the notebook view and vice versa. This would also enable live updates to the graph as an analyst is developing a notebook, dynamically adding new cells and updating the execution path. Currently, analysts need to save their notebooks to trigger updates of the visualization, so that the live updates experience could be somewhat mimicked by using the auto-save feature of notebooks. As such, it would also provide a means to view the structural development history of the notebook. Each of our expert interview participants noted that such an implementation would be useful.

We also noted several usability issues during both studies that could be improved. Several participants noted that encoding variable access versus variable assignment information into the paths would enhance the visual debugging capabilities of the system. In the same vein, displaying the initialization values of variables in cells where they are declared or reset would be beneficial. A future controlled study is planned to better test the usability of Albireo against standard notebook software.

8 Conclusion

All new software presents usability issues, and computational notebooks are no exception. Indeed, previous studies [23, 29] have documented usability issues with computational notebooks that are not addressed by Albireo, including tracking cell versioning and notebook history. Still, computational notebooks appear to be a dominant exploration tool for data science in the near future.

In this work, we introduced Albireo, a visualization assistant designed to supplement computational notebooks. Through interviews with frequent notebook users, we identified a need for a better method to survey the contents of a notebook, develop nonlinearly, and effectively navigate through these nonlinear structures. To support these tasks, we developed Albireo as a force-directed graph visualization which, among other data, encodes relationships between notebook cells based on similar variables, function calls, and strings. We evaluated Albireo through a case study and expert interviews and presented a discussion of its strengths and weaknesses. We found that Albireo increased the exploration and insight abilities of our expert evaluators.

Acknowledgments

We wish to thank our interview participants, as well as the reviewers' comments to improve this work. This work was partially supported by FXPAL as an internship project.

References

[1] C. Andrews, A. Endert, and C. North. Space to think: Large high-resolution displays for sensemaking. In *Proceedings of the SIGCHI*

Conference on Human Factors in Computing Systems, CHI '10, pp. 55–64. ACM, New York, NY, USA, 2010. doi: 10.1145/1753326.1753336

[2] Apache Zeppelin. Zeppelin. https://zeppelin.apache.org/, 2018. Accessed: 2018-08-06.

[3] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Vistrails: enabling interactive multiple-view visualizations. In *IEEE Visualization*, pp. 135–142, Oct 2005. doi: 10.1109/VISUAL.2005.1532788

[4] G. Booch. *The unified modeling language user guide*. Pearson Education India, 2005.

[5] L. Bradel, C. North, L. House, and S. Leman. Multi-model semantic interaction for text analytics. In *2014 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 163–172, Oct 2014. doi: 10.1109/VAST.2014.7042492

[6] E. T. Brown, J. Liu, C. E. Brodley, and R. Chang. Dis-function: Learning distance functions interactively. In *2012 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 83–92, Oct 2012. doi: 10.1109/VAST.2012.6400486

[7] P. Caserta and O. Zendra. Visualization of the static aspects of software: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 17(7):913–933, July 2011. doi: 10.1109/TVCG.2010.110

[8] Databricks. Databricks – making big data simple. https://www.databricks.com/, 2018. Accessed: 2018-08-06.

[9] S. B. Davidson and J. Freire. Provenance and scientific workflows: Challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pp. 1345–1350. ACM, New York, NY, USA, 2008. doi: 10.1145/1376616.1376772

[10] W. Dou, D. H. Jeong, F. Stukes, W. Ribarsky, H. R. Lipford, and R. Chang. Recovering reasoning processes from user interactions. *IEEE Computer Graphics and Applications*, 29(3):52–61, May 2009. doi: 10.1109/MCG.2009.49

[11] M. Dowling, J. Wenskovitch, J. Fry, S. Leman, L. House, and C. North. Sirius: Dual, symmetric, interactive dimension reductions. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):172–182, Jan 2019. doi: 10.1109/TVCG.2018.2865047

[12] A. Endert, P. Fiaux, and C. North. Semantic interaction for sensemaking: Inferring analytical reasoning for model steering. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2879–2888, Dec 2012. doi: 10.1109/TVCG.2012.260

[13] A. Endert, P. Fiaux, and C. North. Semantic interaction for visual text analytics. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 473–482. ACM, 2012.

[14] B. F. Displaying dependence graphs: a hierarchical approach. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(3):151–185, 2004. doi: 10.1002/smr.291

[15] D. Gotz and M. X. Zhou. Characterizing users visual analytic activity for insight provenance. In *2008 IEEE Symposium on Visual Analytics Science and Technology*, pp. 123–130, Oct 2008. doi: 10.1109/VAST.2008.4677365

[16] P. J. Guo. *Software tools to facilitate research programming*. PhD thesis, Stanford University Stanford, CA, 2012.

[17] P. J. Guo and M. I. Seltzer. Burrito: Wrapping your lab notebook in computational infrastructure. *USENIX Workshop on the Theory and Practice of Provenance (TaPP '12)*, 2012.

[18] A. Head, F. Hohman, T. Barik, S. M. Drucker, and R. DeLine. Managing messes in computational notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pp. 270:1–270:12. ACM, New York, NY, USA, 2019. doi: 10.1145/3290605.3300500

[19] J. Heer, J. M. Hellerstein, A. Paepcke, and S. Kandel. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization & Computer Graphics*, 18:2917–2926, 12 2012. doi: 10.1109/TVCG.2012.219

[20] A. Z. Henley, K. Muçlu, M. Christakis, S. D. Fleming, and C. Bird. Cfar: A tool to increase communication, productivity, and review quality in collaborative code reviews. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pp. 157:1–157:13. ACM, New York, NY, USA, 2018. doi: 10.1145/3173574.3173731

[21] J. Hoffswell, A. Satyanarayan, and J. Heer. Augmenting code with in situ visualizations to aid program understanding. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pp. 532:1–532:12. ACM, New York, NY, USA, 2018. doi: 10.1145/3173574.3174106

[22] M. B. Kery, A. Horvath, and B. Myers. Variolite: Supporting exploratory programming by data scientists. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pp. 1265–1276. ACM, New York, NY, USA, 2017. doi: 10.1145/3025453.3025626

[23] M. B. Kery, M. Radensky, M. Arya, B. E. John, and B. A. Myers. The story in the notebook: Exploratory data science using a literate programming tool. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pp. 174:1–174:11. ACM, New York, NY, USA, 2018. doi: 10.1145/3173574.3173748

[24] N. Mahyar and M. Tory. Supporting communication and coordination in collaborative sensemaking. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1633–1642, Dec 2014. doi: 10.1109/TVCG.2014.2346573

[25] F. Perez and B. Granger. Project jupyter: Computational narratives as the engine of collaborative data science. https://blog.jupyter.org/project-jupyter-computational-narratives-as-the-engine-of-collaborative-data-science-2b5fb94c3c58, 2015. Accessed: 2018-07-17.

[26] P. Pirolli and S. Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. *Proceedings of International Conference on Intelligence Analysis*, 5:2–4, 2005.

[27] Project Jupyter. Project jupyter. https://www.jupyter.org/, 2018. Accessed: 2018-08-06.

[28] 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 Transactions on Visualization and Computer Graphics*, 22(1):31–40, Jan 2016. doi: 10.1109/TVCG.2015.2467551

[29] A. Rule, A. Tabard, and J. D. Hollan. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pp. 32:1–32:12. ACM, New York, NY, USA, 2018. doi: 10.1145/3173574.3173606

[30] SageMath. Sagemath – open-source mathematical software system. http://www.sagemath.org/, 2018. Accessed: 2018-08-06.

[31] D. Seider, A. Schreiber, T. Marquardt, and M. Brggemann. Visualizing modules and dependencies of osgi-based applications. In *2016 IEEE Working Conference on Software Visualization (VISSOFT)*, pp. 96–100, Oct 2016. doi: 10.1109/VISSOFT.2016.20

[32] J. Z. Self, M. Dowling, J. Wenskovitch, I. Crandell, M. Wang, L. House, S. Leman, and C. North. Observation-level and parametric interaction for high-dimensional data analysis. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 8(2):15:1–15:36, June 2018. doi: 10.1145/3158230

[33] A. M. Smith, W. Xu, Y. Sun, J. R. Faeder, and G. E. Marai. Rulebender: integrated modeling, simulation and visualization for rule-based intracellular biochemistry. *BMC Bioinformatics*, 13(8):S3, May 2012. doi: 10.1186/1471-2105-13-S8-S3

[34] S. Srinivasa Ragavan, S. K. Kuttal, C. Hill, A. Sarma, D. Piorkowski, and M. Burnett. Foraging among an overabundance of similar variants. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pp. 3509–3521. ACM, New York, NY, USA, 2016. doi: 10.1145/2858036.2858469

[35] A. Tabard, W. E. Mackay, and E. Eastmond. From individual to collaborative: The evolution of prism, a hybrid laboratory notebook. In *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work*, CSCW '08, pp. 569–578. ACM, New York, NY, USA, 2008. doi: 10.1145/1460563.1460653

[36] M. Ufford, M. Pacer, M. Seal, and K. Kelley. Beyond interactive: Notebook innovation at netflix. https://medium.com/netflix-techblog/notebook-innovation-591ee3221233, 2018. Accessed: 2019-08-29.

[37] J. Wenskovitch, L. A. Harris, J.-J. Tapia, J. R. Faeder, and G. E. Marai. Mosbie: a tool for comparison and analysis of rule-based biochemical models. *BMC Bioinformatics*, 15(1):316, Sep 2014. doi: 10.1186/1471-2105-15-316

[38] J. Wenskovitch and C. North. Observation-level interaction with clustering and dimension reduction algorithms. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics*, HILDA'17, pp. 14:1–14:6. ACM, New York, NY, USA, 2017. doi: 10.1145/3077257.3077259

[39] J. Zhao, M. Glueck, P. Isenberg, F. Chevalier, and A. Khan. Supporting handoff in asynchronous collaborative sensemaking using knowledge-transfer graphs. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):340–350, 2017. doi: 10.1109/TVCG.2017.2745279