Case Study Comparison of Computational Notebook Platforms for Interactive Visual Analytics

Han Liu and Chris North

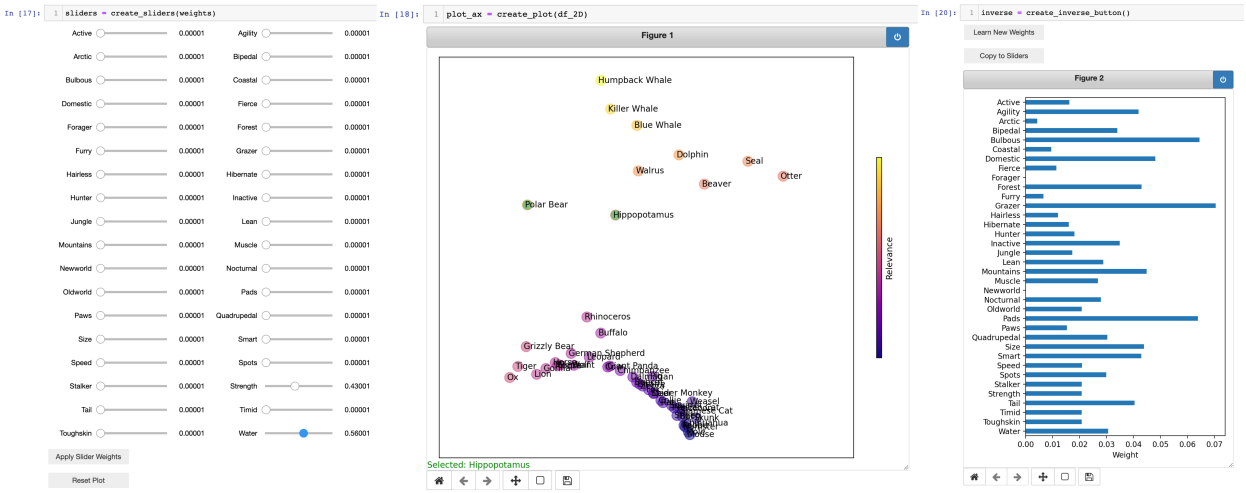

Fig. 1. An overview of the example Andromeda algorithm in Jupyter Notebook with three main parts: 1) sliders to adjust attribute weights, 2) dimension-reduction projection algorithm results, and 3) the resulting weights generated by the inverse projection algorithm.

Abstract—The way of using computational notebooks is quite different between data science and visual analytics. Data scientists focus on data exploration with the code, while visual analytics users are interested in engaging with interactive visual interfaces to facilitate analytical reasoning. Such a difference leads to design contradictions while merging visual analytics tools with data science tools in computational notebooks. In this work, we investigated the problem using an example called "Andromeda," which is an interactive dimension reduction algorithm, and implemented it using three different notebook platforms: 1) Python code in a Jupyter Notebook, 2) JavaScript code in an Observable Notebook, and 3) embedding both Python (data science use) and JavaScript (visual analytics use) in a Jupyter Notebook. Advantages and disadvantages are concluded for each platform by making comparisons based on various aspects, such as design logic, coding differences, performance, and usability. Laying the groundwork for data scientists, advice and recommendations are made on architecting similar notebooks and which platform to choose in various situations.

Index Terms—Visual Analytics, Data Science, Computational Notebooks, Dimension Reduction

1 Introduction

Traditionally visual analytics tools are developed with standalone applications, such as web-hosted applications. However, adapting full-stack systems can be potentially challenging for data scientists for four reasons: 1) lack of background knowledge of web-application development; 2) limited access to the code; 3) low reproducibility; and 4) low code reusability.

As computational notebooks, such as Jupyter and Colab [3], become more popular among data scientists, the issue of how to merge visual analytics tools with data science tools in notebooks becomes increasingly important. Schmidt and Ortner explored the differences between standalone tools and notebook-style workflows, and outlined the barriers to merge visualization techniques in both workflows [14]. Researchers have put effort into building visual analytics tools in notebooks to solve problems in diverse scenarios. However, best practices for implementing visual analytics methods in notebooks are not currently sufficient to benefit data scientists. We aim at exploring the challenges of building such visual analytics notebooks merged with data science packages. Challenges exist because the way of using notebooks is quite different between data science and visual analytics. Data scientists want to dig into code details to see and manipulate the process of how data changes [18], while visual analytics designers emphasize higher-level methods such as interacting with visual representation of the data [4]. While a variety of platforms are currently used, it is unclear what are the tradeoffs in choosing between these platforms.

To investigate the problem, we conducted a case study with an example called "Andromeda", which is an interactive dimension reduction algorithm, and implemented notebooks that merge data science with visual analytics using three common platforms: 1) implemented with only Python language in a Jupyter Notebook; 2) implemented with only JavaScript language in an Observable Notebook; and 3) embedded both Python (data science use) and JavaScript (visual analytics use) in a Jupyter Notebook. Furthermore, the three resulting notebooks are compared based on the design logic, coding differences, and usability.

2 Background and Related Work

A variety of research is influencing visual analytics to shift from full-stack to notebook implementations.

- *Han Liu is with the Sanghani Center at Virginia Tech. E-mail: han3@vt.edu.*
- *Chris North is with the Sanghani Center at Virginia Tech. E-mail: north@vt.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

2.1 Visual Analytics in Full-Stack Applications

Full stack web development was common for implementing visual analytics applications. Full-stack apps are composed of two major parts: 1) front end user interface, and 2) back end data processing. Full-stack development allows complex interactions by integrating different programming languages and frameworks. However, significant disadvantages of full-stack development [16] cause people to shift to use notebooks. First, the creator must be familiar with diverse languages and framework components. Second, the applications lack experimentation due to limited code access and modification. Third, complexity of the implementations make it difficult to distribute for reproducability and reuse. A method to adapt web-based VA systems for notebooks called NOVA [19] converts web applications developed with diverse languages and frameworks to notebooks for easier use.

2.2 Computational Notebooks

Researchers use computational notebooks to combine code, visualizations, and text in single documents in order to support collaborative data analysis [12] [1] [2] [20]. Notebook platforms for data science use include Databricks, Azure, Colab and Jupyter [3]. Whereas, visualization notebook platforms include Observable Notebook. In our analysis, we used Jupyter and Observable as two main ingredients because they are the most popular notebook platforms for implementing visual analytics notebooks based on our experience and by sampling papers from visual analytics conferences such as IEEE VIS.

2.2.1 Jupyter Notebook

Jupyter Notebook is a scientific interactive framework where data scientists can embed code, text, mathematical equations, and interactive graphs [17]. Jupyter cells are either code cells and markdown cells. Code cells support independent implementation, running code and providing feedback. Markdown cells contain documents and display diverse contents, such as images, tables, and links. Two major advantages are: 1) literate programming [6], the layout of the combination with code, output, and textual description, which help users gain an explicit overview and improve the readability of the logic process, and 2) easy to modify and debug, reducing the code implementation for repeated experimentation by decomposing into cells [13].

2.2.2 Observable Notebook

Observable Notebook is an online platform for writing JavaScript code mainly for visualization with packages like D3. While Jupyter Notebook executes code by passing it to an external kernel, Observable Notebook runs its code in the browser in a reactive manner [10]. Cells re-execute automatically when their dependent variables change, which saves users time and frustration since cells are run automatically without user invocation and notebook state maintains consistency.

2.3 Visual Analytics in Notebooks

In the past few years, visual analytics researchers have increasingly used computational notebooks to develop visual analytics methods for data science. For example, Fujiwara and Wei implemented ULCA (Unified Linear Comparison Analysis) on Jupyter Notebook to find similarities and differences between datasets [5].

Also, new platforms have been built to better support visual analytics notebooks. For example, a popular Python package named NotebookJS [11] enables notebook creators to embed interactive visualizations implemented with JavaScript into Python-based Jupyter Notebooks. We include it in our cases study due to its popularity. Another library called B2 [21] presents visualizations in an adjacent frame in a Jupyter notebook. AntEater automatically tracks data in the cells and presents visualizations of the data with minimal code.

2.4 Andromeda

We chose Andromeda [8, 15] as a reference example because it is a representative method that involves both analytics (such as dimension reduction) and interactive visuals (such as sliders and plots). Andromeda is an interactive visual analytics algorithm for the analysis of high-dimensional data. It provides coordinated multiple views for users to interact with: the "Attribute" view and the "Observation" view. Users perform dimension reduction with a weighted multidimensional scaling algorithm (WMDS) on the data by adjusting the weights via sliders in the Attribute view, and see the projection of the reduction result on the 2D plane in the Observation view. Alternatively, users can drag the points in the Observation view to perform inverse projection (Inverse MDS) and see how the weights change in the Attribute view.

3 Andromeda implemented in 3 Platforms

In this section, we describe the implementation and interactions of three Andromeda notebooks developed using three platforms mentioned in the Introduction. These 3 implementations form the basis of our comparison of the platforms.

3.1 Jupyter Andromeda

The implementation of the Jupyter Andromeda (J-Andromeda) utilizes various software packages to support the use of visual analytics and data science. Packages such as Scikit-learn, Pandas, and Numpy were used to perform data processing and WMDS. Moreover, we used Matplotlib to build a draggable scatter plot to display 2D projections of high-dimensional data. More importantly, button-controlled events are supported by the Python package ipywidgets, which allows users to interact with the notebook. In addition, users can perform interactions, shown in Figure 1, such as dragging sliders, clicking buttons, and dragging dots in the scatterplot. The following interactions can be performed individually or as combinations: 1) Weighted Dimension Reduction: users can drag the sliders to change weights of data attributes, and click the "Apply New Weights" button to render the 2D scatterplot. 2) Inverse Dimension Reduction: users can drag points on the scatterplot and click the "Inverse" button to execute the inverse-DR algorithm, and show the resulting attribute weights in the bar chart. Then users can click "Copy to Sliders" button to re-render the scatterplot by applying the obtained weights. 3) Reset Plot: users can reset the weights and scatterplot by clicking the "Reset Plot" button.

3.2 Observable Andromeda

The Observable Andromeda (O-Andromeda) is implemented with JavaScript code which support both data processing and visual analytics use. Differences exist compared to J-Andromeda: 1) due to limited analytics libraries, the dimension reduction method uses D3's force directed graph algorithm to simulate weighted MDS, 2) HTML widgets are used to create sliders and buttons, and 3) plots were built using the package d3.js. In addition, users can interact with sliders, buttons and the scatterplot in a similar way as J-Andromeda, but slightly different due to improved performance. For example, the "Apply New Weights" button is no longer needed. When users drag the sliders, the scatterplot is immediately re-rendered in real-time. In addition, the "Inverse" button is no longer needed. Thus, when users select and drag at least two points on the scatter plot, the inverse projection algorithm automatically executes and updates the bar plot displaying new weights.

3.3 Jupyter with Embedded JavaScript Andromeda

The implementation of the Jupyter with Embedded JavaScript Andromeda (JEJ-Andromeda) involves a python package called "NotebookJS", in which JavaScript code can be embedded into Jupyter Notebook to achieve bidirectional communication between Python (data science use) and JavaScript (visual analytics use). First, based on the slider weights, the projection data is computed in the python kernel using the sklearn library for dimension reduction. Then, the JavaScript code retrieves the data and uses D3 to render the scatterplot within a cell in the Jupyter notebook. Conversely, when users drag dots on the scatterplot, the coordinates of the dragged dots are recorded and delivered from the JavaScript environment to the Python kernel for computation. NotebookJS enables this cyclical communication for interactions. The interactions of the JEJ-Andromeda is similar to J-Andromeda.

4 Comparison of the Three Andromeda Notebooks

During implementing and testing the three versions of Andromeda notebooks, some major differences were found in various aspects, such as design logic, coding difference, and usability.

4.1 Design Logic

The design logic refers to how to architect notebooks that merge data science with visual analytics. It is mainly impacted by five deep-level differences in Table 1: 1) Cell Execution, 2) Control Flow, 3) Interactions with Widgets, 4) Bidirectional Communication, and 5) Button Usage.

	Jupyter	Observable	Jupyter+JS
Cell Execution Manner	manual	reactive	manual
Cell Execution Order	linear	topological	linear
Interaction with Widgets	ipywidgets	html widgets	ipywidgets
Bidirectional Communication	python callbacks	reactive value updates	JavaScript-triggered python callbacks
Button Usage	give control	stop cyclical reference	give control

Table 1. Design Logic Comparison

4.1.1 Cell Execution Manner

Neither the J-Andromeda nor the JEJ-Andromeda cells will be executed unless users run them manually. A typical compilation procedure for a cell consists of 1) selecting the cell and 2) clicking the Run button in the top menu or pressing Shift-Return on the keyboard. Since users are most likely unaware of which cells need to be compiled to perform certain functions, visual analysis notebooks often require the creation of additional buttons to invoke the desired functions. However, due to the reactive nature of O-Andromeda's cells, O-Andromeda's cells are executed automatically, which means that no action is required to run the cell if there are any changes or modifications to the data.

4.1.2 Cell Execution Order

Since Jupyter cells execute linearly, the J-Andromeda runs codes from top to bottom of the notebook. A significant consequence is that a cell will not run successfully if its referenced variables are not declared in the preceding cells. In the JEJ-Andromeda, the JavaScript code will keep running to receive the processed data and render the plot since the first run, though the Python cells run similarly to the J-Andromeda. O-Andromeda uses a topological order to run cells, determined by unit references, which means that no matter how the cells are arranged, variables will be updated as soon as their references change.

4.1.3 Interaction with Widgets

Integrating cells into methods triggered by widgets like buttons is necessary for users to interact with notebooks. Software packages are available for creating widgets, such as ipywidgets for J-Andromeda and JEJ-Andromeda, and HTML widgets for O-Andromeda. The ipywidgets package provides Jupyter widgets in the Ipython kernel to register callback functions. The HTML widgets are familiar to people who know basic web development concepts, such as HTML, DOM (Document Object Model), and CSS. After the declaration of the HTML widget variables, the variable names will be referenced in cells which need to run whenever updated. Thus, the reactive capability runs the dependent cells whenever users interact with the widgets.

4.1.4 Bidirectional Communication

Bidirectional communication between data and visualizations in our notebooks means both data and visualization depend on each other and form a cycle. The J-Andromeda notebook achieves the communication by executing callback functions observed by widgets along with updating the global variable "weights". Once the slider widgets are dragged, new "weights" will be applied to the original data by clicking the "Apply Weights" button and new scatter plot will be generated. Inversely, after the layout of the scatter plot changes by dragging points on it, clicking on the button widget to execute the inverse DR algorithm leads to the update of the "weights" variable. It can be used to trigger a new cycle by clicking the "Apply Weights" button. Communications in O-Andromeda automatically happen between the data and visualization, because of the reactive manner of observable notebook. The JEJ-Andromeda notebook works like a combination of both platforms mentioned above. The python widgets are interacted by users to change weights of attributes and process data with callback functions, while the JavaScript code serves to obtain the processed data to render plots and send the plot data back to python code automatically and continuously.

4.1.5 Button Usage

Buttons in all three notebook platforms are used to execute corresponding functions or cells. Due to the low-speed performance of graph rendering speed and the inverse MDS algorithm, the buttons in J-Andromeda and JEJ-Andromeda are used to divide the whole process into small chunks, which helps users to avoid taking too much time to wait for the process to be completed. Additionally. users can gain control over the execution of small steps of the running algorithm without having to run all the relevant cells manually. In Observable, buttons are not needed for solving the performance issues like in Jupyter. This is because Observable executes the interactive functions so quickly that the results can be displayed in real time as the user interacts. However, button widgets in O-Andromeda are used to avoid the cyclical reference error, which will be discussed in section 5.2.2.

4.2 Coding Difference

4.2.1 Background Knowledge

To implement J-Andromeda, data scientists must be familiar with 1) Python syntax; 2) common Python packages such as Pandas, NumPy, Scikit-learn, and Matplotlib; 3) Jupyter notebook cells; and 4) creating interactions in Jupyter. O-Andromeda requires them to know 1) JavaScript syntax; 2) packages for data processing and visualization such as D3; 3) HTML, CSS, and DOM; and 4) Observable reactive features. However, for JEJ-Andromeda, they should be familiar with both Python and JavaScript syntax and packages.

4.2.2 Debug: print vs console.log

Debugging contributes to successful exploratory operation of the notebook. Data scientists often need to check the values of various variables for debugging or understanding the processing details. Simply typing "print(variable_name)" in J-Andromeda can check the data values within a cell or function and "console.log(variable_name)" in O-Andromeda to view the values by opening the browser console. In addition, the JEJ-Andromeda utilizes both "Print" and "Console.log" methods for debugging. The "Print" method requires only one step to see the result, which is to run the line of code. The "Console.log" method, on the other hand, requires more steps: 1) run the code, 2) inspect the web page, and 3) open the console to see the result.

4.2.3 NCSL

NCSL, a measurement to the difficulty of programming by counting the the number of non-comment, non-blank lines of code [9], can be beneficial when the differences in implementing functions are not caused by programmers' coding knowledge but by the limitations of the platforms. In Table 2, the O-Andromeda takes least lines of code for implementing the user interactions due to D3 and reactive execution. Moreover, the JEJ-Andromeda requires more lines of code to achieve the bidirectional communication, due to NotebookJS transitions between Python and JavaScript. The J-Andromeda Notebook takes the most lines of code for implementing the draggable scatter plot. However, both J-Andromeda and JEJ-Andromeda use less lines for data processing, by exploiting common data-science libraries such as sklearn and pandas.

	Function	Jupyter	Observable	Jupyter+JS
Data Processing	WMDS	3	17	3
Data Processing	Inverse MDS	47	56	47
Visualization	Draggable Scatter Plot	32	11	12
Interaction	Reset Plot Button	7	2	7
Interaction	Bidirectional Communication	4	0	28

Table 2. NCSL for Important Functions

4.3 Usability

4.3.1 Interactive Performance

To compare efficiency of each notebook, we performed a case study on a numeric dataset with 49 rows and 36 columns to measure the speed, as shown in Table 3, for specific methods such as Scatter Plot Rendering, Dragging Dots, and Inverse MDS. The rendering speed of the scatter plot is calculated by the time difference between the start and end of the plot rendering function. The dragging speed of points is collected by observing the time difference between the processing of consecutive dragging events. In addition, the inverse MDS function is measured for the time cost in one run. The O-Andromeda takes the least time for both the scatter plot rendering and dragging dots method, while J-Andromeda performs the slowest. The speed of dragging dots is very slow in the J-Andromeda notebook compared to JavaScript, which can be frustrating for users. Additionally, O-Andromeda takes the least time to run the identical inverse MDS algorithm, presumably because JavaScript execution has been highly optimized in most browsers.

Function	Jupyter	Observable	Jupyter+JS
Scatter Plot Rendering	30-50ms	0-2ms	0-20ms
Dragging Dots	90-115ms	10-25ms	10-25ms
Inverse MDS	900-1100ms	10-30ms	900-1100ms

Table 3. Speed Measurements

4.3.2 Notebook Organization

Due to the linear execution in Jupyter, visualizations are displayed after the data is imported and processed. However, a data scientist may want to display the visualization at the very beginning of the notebook. By creating special empty placeholder cells during initialization and filling them later in the code, the visualization can be placed at the top of J-Andromeda and JEJ-Andromeda with extra lines of code. In contrast, O-Andromeda is much more flexible, as cells can be placed in any order without affecting the execution. As a result, O-Andromeda benefit users by placing all the interfaces, including visualizations, buttons, and sliders, at the very beginning of the notebook.

5 Discussion

5.1 Considerations

We share our considerations of the three platforms to assist data scientists in building their own visual analytics notebooks. The Jupyter platform works the best when data science packages are heavily used and less use of visualization. The Observable platform is the winner when users have high expectation on visualizations, such as fast response time, flexible plot design, CSS styles, and freedom of arranging cells. Furthermore, the Jupyter embedded with JavaScript platform will benefit users who are familiar with both languages and necessary packages to take advantages of both sides. In any of the cases, users will have to deal with workarounds to overcome the disadvantages of each platform as listed below.

5.2 Challenges

Challenges require research on novel tools that support better ways to implement visual analytics notebooks.

5.2.1 Jupyter Notebook

One challenge with visual analytics in Jupyter is that the developer needs to integrate buttons into the notebook design so that the user can easily execute the necessary code to re-compute visualizations when needed. Otherwise the user will need to know which cells to re-execute. Interaction sequences must be decomposed into smaller chunks due to slow performance. Creating buttons requires writing non-trivial code that is irrelevant to the data science task. Furthermore, data processing code must be wrapped into functions that can be invoked by such buttons. Special functions and placeholder cells are needed to proximate visuals from multiple cells. Finally, debugging the values of variables inside of widget-observed functions is difficult, since the result cannot be displayed when users simply apply the "print" method. It requires additional code such as creating placeholders or passing values to global variables.

5.2.2 Observable Notebook

When a cyclical interactive algorithm is built in Observable, an endless loop is caused by cyclical references causing users to lose control of the notebook. An example from Andromeda is shown in Figure 2. Because of the reactive nature, each procedure runs automatically and causes an infinite loop. This problem is exacerbated by the fact that analytical methods such as dimension reduction are sometimes non-deterministic, meaning they get a different result each time around the loop, so the reactive framework keeps firing. Unfortunately, such cyclical methods are often needed in interactive visual analytics. One way to stop the cycle is to introduce buttons into the notebook design (like the "Copy to Sliders" button) that require the user to click to invoke the next cycle.

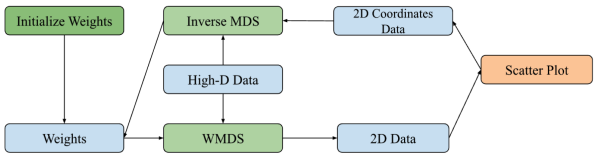

Fig. 2. A Reference Loop of Variables and Functions in Andromeda

5.2.3 Jupyter Notebook Embedded with JavaScript

In JEJ-Andromeda, JavaScript automatically keeps retrieving the processed data and rendering the plot and sending updated coordinates data to the Python side. Automatic re-rendering can interrupt the interactive process. Each communication of data between python and JavaScript involves expensive translation to/from text format. Furthermore, Javascript processes are repeatedly spun off, without a way to terminate them from the notebook. Thus, JavaScript will not only send the coordinates data from the current run but also from previous runs after users rerun the notebook. Users must refresh the notebook web page to stop the previous JavaScript runs. Improvements are needed to allow better communication and control over JavaScript execution.

6 Conclusion and Future Work

Merging data science and visual analytics tools together in notebooks is important as computational notebooks become more popular. In this paper, we presented and compared three platforms to explore the challenges of creating notebooks with visual analytics features. Full detailed results are available in Liu [7]. Although we analytically compared the three platforms on various aspects, how actual users are satisfied with each version under difference task settings is unknown. Therefore, it will be valuable to measure users' satisfaction by conducting empirical user studies.

REFERENCES

[1] S. Alspaugh, N. Zokaei, A. Liu, C. Jin, and M. A. Hearst. Futzing and moseying: Interviews with professional data analysts on exploration practices. *IEEE transactions on visualization and computer graphics*, 25(1):22–31, 2018.

[2] A. Batch and N. Elmqvist. The interactive visualization gap in initial exploratory data analysis. *IEEE transactions on visualization and computer graphics*, 24(1):278–287, 2017.

[3] S. Chattopadhyay, I. Prasad, A. Z. Henley, A. Sarma, and T. Barik. What's wrong with computational notebooks? pain points, needs, and design opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1–12, 2020.

[4] K. A. Cook and J. J. Thomas. Illuminating the path: The research and development agenda for visual analytics. Technical report, Pacific Northwest National Lab.(PNNL), Richland, WA (United States), 2005.

[5] T. Fujiwara, X. Wei, J. Zhao, and K.-L. Ma. Interactive dimensionality reduction for comparative analysis. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):758–768, 2021.

[6] D. E. Knuth. Literate programming. *The computer journal*, 27(2):97–111, 1984.

[7] H. Liu. Comparison of computational notebook platforms for interactive visual analytics: Case study of andromeda implementations. *MS Thesis, Dept of Computer Science, Virginia Tech*, 2022.

[8] H. Liu, Y. Bian, and C. North. Andromeda in jupyter: Interactive inverse dimension reduction. In *Proceedings of the 4th Workshop on Visualization for AI Explainability*, 2021.

[9] D. J. Lubinsky. Measuring software size by distinct lines. In *Proceedings Fourteenth Annual International Computer Software and Applications Conference*, pp. 403–404. IEEE Computer Society, 1990.

[10] S. N. Pattanaik and A. Benamira. Teaching computer graphics during pandemic using observable notebook. In *Eurographics*, 2021.

[11] J. Piazentin Ono, J. Freire, and C. T. Silva. Interactive data visualization in jupyter notebooks. *Computing in Science Engineering*, 23(2):99–106, 2021. doi: 10.1109/MCSE.2021.3052619

[12] 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*, pp. 1–12, 2018.

[13] T. Sarkar. Writing machine learning code more productively. In *Productive and Efficient Data Science with Python*, pp. 85–111. Springer, 2022.

[14] J. Schmidt and T. Ortner. Visualization in notebook-style interfaces. In *VisGap@Eurographics/EuroVis*, 2020.

[15] 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):1–36, 2018.

[16] R. K. Soni. The big picture of full stack web development. In *Full Stack AngularJS for Java Developers*, pp. 1–27. Springer, 2017.

[17] A. Suárez, M. A. Alvarez-Feijoo, R. Fernandez Gonzalez, and E. Arce. Teaching optimization of manufacturing problems via code components of a jupyter notebook. *Computer Applications in Engineering Education*, 26(5):1102–1110, 2018.

[18] A. Y. Wang, A. Mittal, C. Brooks, and S. Oney. How data scientists use computational notebooks for real-time collaboration. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–30, 2019.

[19] Z. J. Wang, D. Munechika, S. Lee, and D. H. Chau. Nova: A practical method for creating notebook-ready visual analytics. *arXiv preprint arXiv:2205.03963*, 2022.

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

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