The Gap between Visualization Research and Visualization Software (VisGap) (2020)
C. Gillmann, M. Krone, G. Reina, T. Wischgoll (Editors)

Visualization in Notebook-Style Interfaces

J. Schmidt[1] and T. Ortner[1]

[1]VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH, Vienna, Austria

Abstract

Visualization research has always stressed the need for visual tools for data exploration and sense making. Despite the fact that many visualization technologies are available nowadays, their application in modern data science workflows is limited. One of the manifold reasons behind this is the development of visual analytics tools as standalone applications, featuring the complete pipeline from data loading to visualization. Other tools are targeted towards specific use cases (e.g., data wrangling), but to not provide standardized interfaces for import and export. This does not reflect the approach of stitching together several tools as it is employed in data science workflows nowadays. In this paper we outline the differences between standalone tools and notebook-style workflows for a specific use case for time series analysis. The outcomes demonstrate the benefits of notebook-style interfaces for tracking the steps in a data analysis workflow in a narrative way, for reporting, and for collaboration. We therefore argue that not considering the current developments towards the increased application of notebook-style interfaces for data science will lead to a reduced application and acceptance of visualization techniques in these domains. We outline the barriers for the integration of visualization techniques in narrative workflows, and describe directions for future research.

1. Introduction

The landscape of data analytics workflows has drastically changed within the past years. The emergence of data science as a term and as a research field has led to many users in diverse domains being confronted with new needs for data analytics. Advances in sensor technologies, embedded devices, and mobile applications make it possible to capture large amounts of data from many different contexts [Ber16], with many users who want to make more of this data in the future.

Data science can be defined as a "*concept to unify statistics, data analysis, machine learning and their related methods*" in order to "*understand and analyze actual phenomena with data*" [Hay98]. As such, data science comprises the interdisciplinary integration of techniques from mathematics, statistics, computer science, and information science [PGL*11]. In their way to discovering new insights from data, data scientists have to undergo five different stages [KPHH12], namely finding datasets suitable for the analysis (*Discover*), bringing data into a desired format (*Wrangle*), verifying the quality of the data and understanding its structure (*Profile*), using data for model building (*Model*), and reporting the results to stakeholders (*Report*).

In the past, the aim when developing visual analytics applications was often driven by providing a holistic interface for data loading, data handling, visualization, and handling of insights. Successful tools like Tableau [Tab20] and MS Power BI [Mic20b] follow this intended goal, by providing standalone applications that support major parts of the data analytics workflow. Especially users without computational background greatly benefit from such standalone approaches. Other visual analytics tools are targeted towards specific use cases or workflow steps to support, for example, data wrangling [KPHH11].

Data scientists are used to working with several different tools [AZL*19], since no available solution covers all aspects of the data analytics workflow. This way data science in many cases converged to stitching together several different tools in every workflow stage to achieve the intended final data analysis goal. Scripting languages like Python and R have become more popular than ever [BLH15], since they are easy to access (i.e., open source), easy to learn, and are supported by a large community providing help and additional plugins. In this rapidly changing environment, standalone and/or isolated visual analytics applications are not widely accepted. Visual analytics systems so far hardly provide standardized interfaces to export and re-use actions (e.g., selections, filtering) taken in the provided visualizations. This had led to a gap between the multitude of visualization techniques being developed in research [ML17], and the visualization techniques actually applied by data scientists.

Data scientists, and other scientists working with data, prefer narrative approaches to keep track of their data workflow. Literate programming tools [KRA*18] such as notebooks help data scientists to record their steps and decisions in a data analysis workflow. They allow scientists to save whole workflows and in this way make decisions and results reproducible. Figure 1 depicts an example using notebooks as a literate programming tool. Notebooks consist of smaller elements, typically called *cells*, which contain smaller portions of code. The cells can be executed independently, but vari-

ables may be shared across them. All cells together describe the complete data analysis workflow and all steps that have been executed. Literate programming sessions can be saved and restored so that users can recall the workflow at any time.

Figure 1: *Notebook application. Here two cells in a Kaggle notebook are shown, containing Python code to load a timeseries dataset and an inline visualization to plot the data.*

The integration of visualization techniques in common narrative programming interfaces (e.g., Jupyter notebooks in Python) is, however, still very limited [Sch20]. Consequently, in most of the cases data practitioners end up using only well-known, basic charts while they refrain from applying advanced techniques. This is especially true for the interactive exploration of data [BE18]. At the same time, data scientists express their interest in applying new visualization techniques [Mee19] and show great interest in trying out alternatives [LBE20]. We therefore argue for a stronger focus on **notebook-style visualization techniques** in visualization research which seamlessly integrate into existing literate programming environments, by

- employing the graphical programming features provided by the narrative programming environments,
- enabling fast visual representations,
- providing interfaces for loading data, and
- providing interfaces for exporting results from user interactions.

In this paper we outline an exemplary data analysis use cases and show how it could be solved by either using a standalone system or notebooks (Section 2). From the results and by analyzing related work we could identify current barriers for including visualizations in notebook-style environments (Section 3), and could define directions for future research (Section 4).

2. Data Science Use Case

Time series data describes datasets consisting of measured values of points in a temporal order [BJ94]. In many cases time series are recorded by built-in sensors in Internet-of-Things (IoT) or industrial production applications, or through observing long-running processes such as financing. Time series datasets can easily become large and therefore pose interesting challenges for visualization research [AMST11]. Typical use cases when working with this type of data are statistical analysis, pattern search, anomaly detection, and modeling and forecasting.

We chose a time series analysis use case to be able to address many important aspects of data analysis processes. Time series analysis usually involves large data records and tasks of a highly cyclic and repetitive nature.

2.1. Task Description

In the use case described in this paper we perform a pattern search task on a set of time series. The use case consists of five tasks. In the following, the tasks are described, alongside with references to the data science workflow steps [KPHH12] they are performed in:

(1) Import: Loading the data (*Wrangle*)
(2) Structure: Understanding the structure of the data (*Profile*)
(3) Patterns: Detecting interesting patterns (*Profile*)
(4) Pattern Search: Performing pattern search and examining the results (*Profile*)
(5) Report: Saving the results for reporting (*Report*)

The *Discover* stage is not represented here since we used predefined datasets, and since this stage is not of major interest for visual analytics research. Three out of five tasks are assigned to the *Profile* stage of the data science workflow, since we would like to emphasize the interactive exploration of data in this use case.

2.2. Use Case Execution

We performed the use case twice, once using a standalone visual analytics tool providing dashboards with linked views (Visplore [PTMB09]) and a Python-based notebook (Jupyter [KRKP*16]). A comparison of the two tools can be seen in Figure 2. In the following the tasks and the differences between the two tools are described:

(1) Import

As a first step the data had to be loaded. The time series datasets were stored as CSV files with columns representing the time series and rows referring to timestamps. The first of the two used datasets contained 526 rows (timestamps) and 37 columns (time series), and the second dataset consisted of 504,768 rows (timestamps) and 22 columns (time series). A side-by-side comparison of the data loading mechanisms of the two tools we used can be seen in Figure 3.

Standalone: In a standalone tool users can rely on the data import functionalities provided by the system. In the case of Visplore we could make use of the built-in CSV import functions. Many standalone tools, including Visplore, also provide previews of the loaded data and simple data wrangling tasks on import (e.g., removing missing values).

Notebook: In the case of notebooks the data import has to be done manually. In our case Python provided built-in CSV loading capabilities, which nevertheless required to write some lines of Python code. Checking the loaded data and data wrangling, if necessary, has to be executed manually by the user.

Figure 2: *Tools used for the time series use case. We used a dashboard of the standalone visual analytics tool Visplore [PTMB09] with linked views ("Standalone Tool") and a Python Jupyter [KRKP*16] notebook ("Notebook") to solve the tasks of the use case. The complete analysis dashboard of the standalone tool is shown, but only a part of the complete notebook could be displayed here, due to the horizontal alignment of the notebook.*

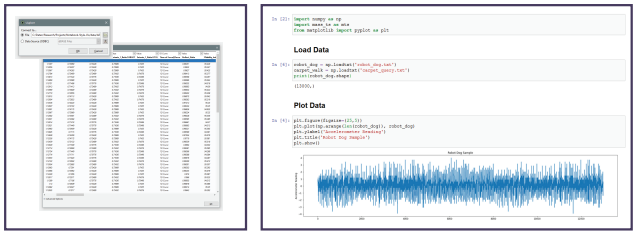

Figure 3: *Import. (left) The standalone tools provides built-in data loading, and preview functionalities, and in this case even simple data wrangling functionalities (e.g., removing missing values). (right) In notebooks the data import has to be done manually, and data checking can be achieved by plotting the data.*

(2) Structure

After loading the data, users need to become familiar with the structure and the quality of the data at hand. In this use case understanding the data meant checking how many time series are available, checking the length of the time series, and detecting time series with an interesting structure (i.e., repetitive or cyclic patterns).

Standalone: Visplore is targeted towards the analysis of large time series data and therefore provides many functionalities for assessing the quality of the time series data at hand [ASMP17], and for comparing time series. In this use case we concentrated on the line plots that were used to understand the structure of the time series data. Plotting the data in a temporal order (time on the x-axis) reveals a temporal structure, and plotting several time series in the same view with different colors can be used for comparison. The linked-view environment in a standalone system allows to quickly change between different views (e.g., clicking through the time series and plotting them in another view), and allows users to rapidly get a concise overview of what is available in the data.

Notebook: Every time the user wants to see a time series in the data in more detail, the code in one of the cells has to be changed and executed, so that the corresponding line plot showing a time series is updated. Another possibility is to plot the time series below each other. Getting an overview of the data this way is therefore less intuitive than with the standalone tool and also requires more manual adjustments and code sequences.

(3) Patterns

After deciding for a specific time series in the data, the users needed to find the patterns that are of interest for them. Such patterns could represent, for example, unusual peaks, certain periods in the data, periodic or cyclic behavior, or other time series structures representing a semantic meaning. Detecting patterns of interest required a more detailed analysis of the data. This included zooming into certain parts of the time series line plots, marking interesting regions, and comparing them.

Standalone: All plots in the linked-view environment provided functionalities for zooming and selecting certain regions. This is a common feature provided by many standalone tools. The selected patterns could be compared by marking them and placing them side-by-side.

Notebook: The basic line plots in the Jupyter notebook provided zooming possibilities, but no interactive possibilities for selection and filtering. The detection of interesting patterns and keeping track of discovered features therefore were highly manual tasks. As an advantage of the narrative style, storing patterns in different cells helped to keep track of what has already been discovered.

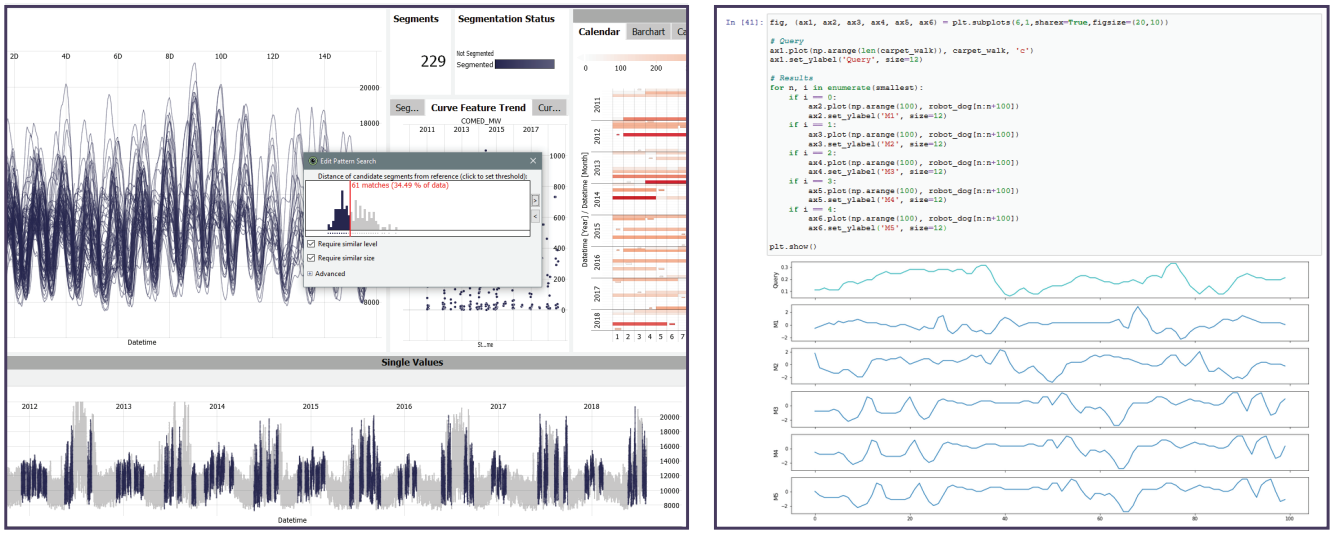

Figure 4: *Pattern search. (left) The pattern search can be started within the dashboard. Comparing different results (e.g., when changing search patterns or algorithm parameters) is not supported by default. (right) Due to the the narrative style of notebooks, all steps in the workflow are preserved. In this case different results using the same search query are shown below each other.*

(4) Pattern Search

Pattern search in time series refers to the process of taking a search pattern as input and detecting all occurrences of this pattern in a given time series. More advanced algorithms for pattern search also allow users to detect patterns of varying length or intensity. The results of a pattern search are a list of timestamps at which the patterns occurs in the given time series. In this use case we used one of the patterns identified in the previous step as input. The two different interfaces of the tools when performing the pattern search are shown in Figure 4.

Standalone: Visplore provides a simple pattern search algorithm based on Dynamic Time Warping [MÖ7] to detect occurrences of a pattern in a time series. Users can select the input pattern and vary a similarity threshold. The resulting segments of the pattern search are marked with color in the time series plot. Additionally, the detected patterns can be overlayed in one plot. Varying the similarity threshold can be done by using a slider, and a histogram in the background reveals how the variation will affect the results. The analysis functionalities of standalone tools are naturally limited to the algorithms that have been integrated into the tools. Varying parameters results in an update of the linked views, which does not allow for keeping track of the changes. Comparing different parameter settings would require manual steps by the users (e.g., by saving states).

Notebook: Python provides several different libraries for time series analysis, also including pattern search capabilities. Since Jupyter notebooks are closely integrated into the programming environment, all available Python libraries and extensions can be tested. We used the Python *mass-ts* [Pyt20] library with GPU acceleration to do the pattern search. For displaying the results, manual steps were necessary by the user. When trying out different parameter settings, we used different cells and therefore could keep track of the settings we already tried. We also could easily compare the different results by placing them below each other in one plot. It would further also be easily possible to integrate other algorithms into the notebook in a similar manner. In an iterative workflow and when trying out alternatives, notebooks better preserve the story of the previous steps (*provenance*), which can even easily be shared with others (*collaboration*).

(5) Report

Reporting the results of a data analysis can be targeted towards different types of end users. When reporting to external stakeholders and users from different domains, usually simple visualizations are used to summarize the findings and explain the consequences. When discussing with other data scientists, more elaborate visualizations can be employed, since the other parties are usually interested in understanding the details, and also the steps that were taken to achieve the results.

Standalone: Standalone tools such as Visplore usually provide possibilities for automatically exporting images, screenshots, and maybe even PDF reports to show the current results of a data analysis workflow. For being able to communicate the steps and decisions they made to create certain results, users have to manually keep track of their analysis workflow.

Notebook: The narrative style of notebooks naturally creates a log of the complete workflow, so that notebooks can be used as a direct communication and collaboration tool for data scientists. When creating reports for other stakeholders, users have to manually design new plots and align them properly (e.g., in a PDF document).

3. Notebook-Style Visualizations

Both standalone tools and notebooks were suitable applications for fulfilling the tasks of the time series analysis use case in Section 2. Dashboards with linked views allow for a very quick and intuitive initial exploration of the data, to get an overview of what is contained in the dataset, and to better understand its structure. Notebooks provide a more flexible environment supporting iterative analysis and evaluation of different analysis algorithms, since notebooks are closely integrated into the corresponding programming environment. The cell structure and narrative approach of notebooks allows for keeping track of the analysis steps in the workflow, something that is lost in the linked-view environment of standalone tools.

3.1. Existing approaches

In our use case we applied Python *Jupyter* notebooks [KRKP*16], which are a famous modern application employing the literate programming paradigm. Jupyter's goal is to achieve a computational narrative by combining human language, live code, and its results into one narrative document. Visualizations for specific use cases (e.g., network visualization [RLW*17]) have already been proposed with the help of Jupyter notebooks.

For web-based applications, *Observable* [Obs20] provides JavaScript notebooks for instant feedback, where it is very straightforward to add interaction and animations. The notebooks can be shared in a web-based manner. *distill* [dis20] overcomes the limitations of static documents like PDF files by providing the possibility to integrate interactive content into text documents. Most of the documents are web-based, but distill also features the integration into Jupyter notebooks.

The *litvis* system [WKD19] supports the process of literate visualization by focussing on low-cost design exposition. litvis aims to integrate design exposition in a way that minimizes the time needed for designers to formulate the exposition, making it easier to adopt into the design process. The litvis system handles live documents that contain the textual narrative in the form of markup code written in Markdown and optionally styled with CSS. The branching possibilities in litvis allow arranging documents into a document tree to explore different ideas and analysis directions.

In addition to technical systems, the term *literate analytics* was coined by Mathisen et al. [MHK*19]. Literate analytics describes a novel approach which not only captures the final state of an analysis, but instead documents the whole analytic process with all possible dead-ends along the way in a comprehensive narrative. This is very close to what can also be achieved with notebooks, but is hardly supported – and requires manual efforts – in current standalone visual analytics systems.

3.2. Visualization in Notebooks

The reasons why visualization techniques are not targeted towards the integration into notebooks yet are manifold. There is also little research or related work on the integration of visualization in notebooks yet. Here we list some of the technical limitations and barriers we identified towards a better integration of visualization:

The possibilities for *graphical programming* in notebooks are limited. Access to the graphics card is not possible at all times and depends on the libraries provided by the narrative programming environment. This definitely limits the possibilities for accelerating the graphical representations and requires using simpler visual mappings. Furthermore, less screen space is available for the visualizations, and full screen representations are hardly possible.

Similar to graphical programming, also the *interaction possibilities* are limited in a notebook environment. Keystrokes will most likely be occupied by the notebook environment and are not available to be used in the visualization.

The *computational power* which can be enforced depends on the narrative programming environment. Applications requiring high computational efforts will be slower in scripting environments than if the same analysis would be implemented in a high-performance programming language like C++, and in an application targeted towards the intended use case.

Documentation, giving examples for using visualizations, and user guides increase the *engagement in the data science communities*. Since the daily duties of a researcher usually do not allow for much extra time which could be spend on the tasks mentioned above, the developed prototypes are not well integrated into the narrative programming environments used by data scientists yet.

4. Directions for Future Work

The overall application of visualization in notebooks is still limited. We argue for enforcing this direction, to not lose touch with data scientists and the interesting use cases and task descriptions they can provide us with. Through analyzing the possible barriers for notebook-style visualization (Section 3.2), we could identify the following possible directions for future work:

Placing Visualizations in the Data Workflow Supporting the data interfacing capabilities provided by the programming environments will ease the integration of different types of visualizations into notebooks and narrative contexts. At the same time, we should work on standardized data interfaces for visualization for both input and output data. Input data is defined as the data used for the visualization. Output data is defined as data created by interacting with the visualization (e.g., selections, filtering). An interesting direction for future work is the definition of bidirectional channels between the visualizations and the notebook environment.

Development of new Domain-Specific Languages Domain-specific languages (DSLs) can be used to formulate complex visualizations in a concise and expressive way [RBGH14]. As such, DSLs for notebook-style visualizations like Vega-Lite [SMWH17] provide a more usable and scalable integration of different notebook-like environments. In the future, new DSLs for specific domains, such as biology or geology, need to be defined to let experts specify *what* they want to do and see, instead of being forced to focus on *how* they can achieve their goal. Well-designed DSLs remove complexity from scripting, while allowing for high-performance optimizations oblivious to the user [RO12], which shall increase the applicability of visualization techniques in various domains.

Standards for User Interactions In visualization research, we should work on standards for interaction mechanisms that should be supported by notebook-style visualizations. This way users will know in all cases how to interact with the visualizations, and how to create new output data from visualizations.

Guidance To many domain experts it is not natural to formulate their goals in code. Guidance systems can help users to navigate through libraries and functions by providing intelligent suggestions similar to IntelliSense [Mic20a]. While IntelliSense is indispensable for writing code, a recommendation system can go well beyond showing parameter info or member lists. We envision, for instance, visualization widgets helping in specifying a filter query while writing it or providing a quick preview of the selected data before executing the query. We believe that guidance systems will significantly reduce entry barriers for users applying visualizations in notebooks.

Incremental Evaluation and Execution Context Tasks can be computationally expensive, may involve large amounts of data, require the computation of supporting data structures, or all three. A good DSL can keep the user oblivious to these details, nevertheless, changes to input parameters may result in performance problems. Incremental evaluation can mitigate these problems with reevaluating only parts of derived data, data structures, visualizations that have actually changed [SASS15]. In terms of computation, memory, and bandwidth it is often desired to specify the execution context of code cells if they should be evaluated at the server, the client browser, or graphics card.

5. Conclusion

In this paper we discussed the gap between common data science workflows in notebooks and advanced visualization techniques. We outlined a data science use for the analysis of time series data and performing pattern search in a standalone and a notebook tool. The results of our use case show that both standalone applications and notebooks are suitable for an explorative analysis of the data. Notebooks, however, are better integrated in the corresponding programming environment and therefore provide direct access to the underlying extensions and plugins. The narrative style of notebooks is also better suited for keeping track of the workflow and for reporting the findings. We then discussed the main barriers for integrating visualization into notebooks, which are mainly the limited graphical programming possibilities and the lack of standardized data interfaces. Ideas on how these challenges can be tackled in the future are discussed. We are aware that addressing only one use case based on time series data and only two tools does not cover a full evaluation of many possible solutions one could think of. Jupyter notebooks are a representative literate programming tool, though. In the future, we would like to explore more use cases, and evaluate in detail the advantages and disadvantages of data analysis using standalone or narrative solutions.

Acknowledgments

VRVis is funded by BMK, BMDW, Styria, SFG and Vienna Business Agency in the scope of COMET - Competence Centers for Excellent Technologies (854174) which is managed by FFG.

References

[AMST11] AIGNER W., MIKSCH S., SCHUMANN H., TOMINSKI C.: *Visualization of Time-Oriented Data*, 1 ed. Human–Computer Interaction Series. Springer-Verlag London, 2011. doi:10.1007/978-0-85729-079-3. 2

[ASMP17] ARBESSER C., SPECHTENHAUSER F., MÜHLBACHER T., PIRINGER H.: Visplause: Visual Data Quality Assessment of Many Time Series Using Plausibility Checks. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (2017), 641–650. doi:10.1109/TVCG.2016.2598592. 3

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

[BE18] BATCH A., ELMQVIST N.: The Interactive Visualization Gap in Initial Exploratory Data Analysis. *IEEE Transactions on Visualization and Computer Graphics 24*, 1 (Jan 2018), 278–287. doi:10.1109/TVCG.2017.2743990. 2

[Ber16] BERTINO E.: Introduction to Data Science and Engineering. *Data Science and Engineering 1* (2016), 1–3. doi:10.1007/s41019-016-0005-1. 1

[BJ94] BOX G. E. P., JENKINS G. M.: *Time Series Analysis: Forecasting and Control*, 3rd ed. Prentice Hall PTR, 1994. 2

[BLH15] BARLAS P., LANNING I., HEAVEY C.: A survey of open source data science tools. *International Journal of Intelligent Computing and Cybernetics 8* (2015), 232–261. doi:10.1108/IJICC-07-2014-0031. 1

[dis20] DISTILL: Distill is dedicated to clear explanations of machine learning. https://distill.pub/, 2020. [accessed 2020-03-08]. 5

[Hay98] HAYASHI C.: What is Data Science ? Fundamental Concepts and a Heuristic Example. In *Data Science, Classification, and Related Methods* (1998), Springer Japan, pp. 40–51. 1

[KPHH11] KANDEL S., PAEPCKE A., HELLERSTEIN J., HEER J.: Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada, May 7–12 2011), CHI '11, ACM, pp. 3363–3372. doi:10.1145/1978942.1979444. 1

[KPHH12] KANDEL S., PAEPCKE A., HELLERSTEIN J. M., HEER J.: Enterprise Data Analysis and Visualization: An Interview Study. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (2012), 2917–2926. doi:10.1109/TVCG.2012.219. 1, 2

[KRA*18] KERY M. B., RADENSKY M., ARYA M., JOHN B. E., MYERS B. A.: The Story in the Notebook: Exploratory Data Science Using a Literate Programming Tool. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada, April 21–26 2018), CHI '18, ACM. doi:10.1145/3173574.3173748. 1

[KRKP*16] KLUYVER T., RAGAN-KELLEY B., PEREZ F., GRANGER B., BUSSONNIER M., FREDERIC J., KELLEY K., HAMRICK J., GROUT J., CORLAY S., IVANOV P., AVILA D., ABDALLA S., WILLING C.: Jupyter Notebooks – a publishing format for reproducible computational workflows. *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (2016), 87–90. doi:10.3233/978-1-61499-649-1-87. 2, 3, 5

[LBE20] LIU J., BOUKHELIFA N., EAGAN J. R.: Understanding the Role of Alternatives in Data Analysis Practices. *IEEE Transactions on Visualization and Computer Graphics 26*, 1 (2020), 66–76. doi:10.1109/TVCG.2019.2934593. 2

[MÖ07] MÜLLER M.: *Dynamic Time Warping*. Springer Berlin Heidelberg, 2007, pp. 69–84. doi:10.1007/978-3-540-74048-3_4. 4

[Mee19] MEEKS E.: 2019 Annual Data Visualization Survey Results. https://bit.ly/38GLV1D, Aug. 2019. [accessed 2020-02-15]. 2

[MHK*19] MATHISEN A., HORAK T., KLOKMOSE C. N., GRØNBÆK K., ELMQVIST N.: InsideInsights: Integrating Data-Driven Reporting in Collaborative Visual Analytics. *Computer Graphics Forum 38*, 3 (2019), 649–661. doi:10.1111/cgf.13717. 5

[Mic20a] MICROSOFT: IntelliSense in Visual Studio Code. https://code.visualstudio.com/docs/editor/intellisense, 2020. [accessed 2020-03-07]. 6

[Mic20b] MICROSOFT: Microsoft Power BI. https://powerbi.microsoft.com, 2020. [accessed 2020-02-29]. 1

[ML17] MCNABB L., LARAMEE R. S.: Survey of Surveys (SoS) - Mapping The Landscape of Survey Papers in Information Visualization. *Computer Graphics Forum 36* (2017), 589–617. doi:10.1111/cgf.13212. 1

[Obs20] OBSERVABLE: The magic notebook for exploring data / Observable. https://observablehq.com, 2020. [accessed 2020-03-01]. 5

[PGL*11] PARSONS M. A., GODØY Ø., LEDREW E., DE BRUIN T. F., DANIS B., TOMLINSON S., CARLSON D.: A conceptual framework for managing very diverse data for complex, interdisciplinary science. *Journal of Information Science 37*, 6 (2011), 555–569. doi:10.1177/0165551511412705. 1

[PTMB09] PIRINGER H., TOMINSKI C., MUIGG P., BERGER W.: A Multi-Threading Architecture to Support Interactive Visual Exploration. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1113–1120. doi:10.1109/TVCG.2009.110. 2, 3

[Pyt20] PYTHON: MASS (Mueen's Algorithm for Similarity Search). https://pypi.org/project/mass-ts/, 2020. [accessed 2020-03-04]. 4

[RBGH14] RAUTEK P., BRUCKNER S., GRÖLLER M. E., HADWIGER M.: ViSlang: A System for Interpreted Domain-Specific Languages for Scientific Visualization. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (2014), 2388–2396. doi:10.1109/TVCG.2014.2346318. 5

[RLW*17] ROSENTHAL S. B., LEN J., WEBSTER M., GARY A., BIRMINGHAM A., FISCH K. M.: Interactive network visualization in Jupyter notebooks: visJS2jupyter. *Bioinformatics 34*, 1 (2017), 126–128. doi:10.1093/bioinformatics/btx581. 5

[RO12] ROMPF T., ODERSKY M.: Lightweight Modular Staging: A Pragmatic Approach to Runtime Code Generation and Compiled DSLs. *Communications of the ACM 55*, 6 (2012), 121–130. doi:10.1145/2184319.2184345. 5

[SASS15] SCHULZ H.-J., ANGELINI M., SANTUCCI G., SCHUMANN H.: An Enhanced Visualization Process Model for Incremental Visualization. *IEEE Transactions on Visualization and Computer Graphics 22*, 7 (2015), 1830–1842. doi:10.1109/TVCG.2015.2462356. 6

[Sch20] SCHMIDT J.: Usage of Visualization Techniques in Data Science Workflows. In *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications* (Valletta, Malta, Feb. 27–29 2020), VISIGRAPP '20, SciTePress, pp. 309–316. doi:10.5220/0009181903090316. 2

[SMWH17] SATYANARAYAN A., MORITZ D., WONGSUPHASAWAT K., HEER J.: Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (2017), 341–350. doi:10.1109/TVCG.2016.2599030. 5

[Tab20] TABLEAU: Tableau. https://www.tableau.com, 2020. [accessed 2020-02-12]. 1

[WKD19] WOOD J., KACHKAEV A., DYKES J.: Design Exposition with Literate Visualization. *IEEE Transactions on Visualization and Computer Graphics 25*, 1 (2019), 759–768. doi:10.1109/TVCG.2018.2864836. 5