Linting for Visualization: Towards a Practical Automated Visualization Guidance System

Andrew McNutt* Gordon Kindlmann†

University of Chicago
Department of Computer Science

ABSTRACT

Constructing effective charts and graphs in a scientific setting is a nuanced task that requires a thorough understanding of visualization design; a knowledge that may not be available to all practicing scientists. Previous attempts to address this problem have pushed chart creators to pore over large collections of guidelines and heuristics, or to relegate their entire workflow to end-to-end tools that provide automated recommendations. In this paper we bring together these two strains of ideas by introducing the use of lint as a mechanism for guiding chart creators towards effective visualizations in a manner that can be configured to taste and task without forcing users to abandon their usual workflows. The programmatic evaluation model of *visualization linting* (or vis lint) offers a compelling framework for the automation of visualization guidelines, as it offers unambiguous feedback during the chart creation process, and can execute analyses derived from machine vision and natural language processing. We demonstrate the feasibility of this system through the production of `vislint_mpl`, a prototype visualization linting system, that evaluates charts created in matplotlib.

Index Terms: Human-centered computing—Visualization—Visualization systems and tools; Human-centered computing—Visualization—Visualization design and evaluation methods

1 INTRODUCTION

Data visualization is essential for scientific and technical communication. The impact of a paper often hinges on the clarity of its images [22], and outside academia, poorly produced charts may have catastrophic consequences [29, 38]. Unfortunately, knowing how to make good visualizations seems to be easier than reliably putting that knowledge into practice. Visualization experts can package their knowledge into collections of guidelines, such as VisGuides [10]. Visualization creators then face the task of internalizing, remembering, and consistently applying the guidelines throughout their work. Alternatively, visualization design knowledge can be externally embodied in systems that provide automated chart recommendations [24, 39, 43], including commercial tools like Spotfire [34] and Tableau [35]. These tools, however, usually must own the entire visualization and analysis process, which may fail to reach sophisticated users in their native computing environment.

The number and variety of collections of vis design guidelines speaks to the activity and progress in visualization as a growing discipline. Creators of visualizations have many collections of best practices to choose from [13, 28, 38]. Chen et al. [5] highlight the importance of developing open venues for such collections of guidelines as a pathway for advancing the field of visualization in general, a call to action which is answered by Diehl et al. [10] with VisGuides. Guidelines can come as pithy maxims, such as "maximize data ink ratio" [37] or as general heuristics like "use color to maximize perceptive effects" [36]. While collections of guidelines can adapt to task and context, and grow with new best practices, it still places a cognitive burden on the chart creator to understand and remember all of the rules. For instance, a scientist may not understand what "Chart Junk" is, and why to avoid it [37].

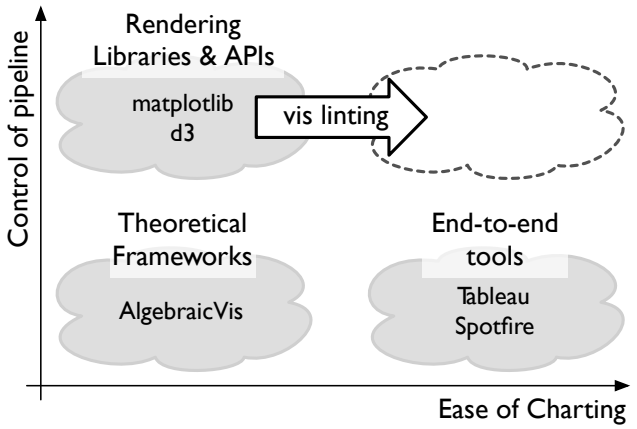

Figure 1: Automated recommendations from end-to-end systems simplify the process of making a good visualization, but provide their users a low granularity of control. In contrast, rendering libraries can provide a high degree of programmatic control but do not usefully guide users towards good visualization design. We see visualization linting as one way to smoothly guide users of visualization libraries towards the best possible results.

The maturation of visualization as a discipline is also reflected in the high-quality software that it has produced for managing the entire process of creating visualizations. One concern with end-to-end tools is how their design expertise is implicitly encoded in the architecture of the software; another is how the software is typically agnostic to task. This is problematic because vis design guidance can and should change with task [31], with the context of the audience or the application, and with advances in scientific studies of visualization methods. The effectiveness measure of APT [24], for example, uses a ranking based on Cleveland and McGill [7] embedded directly in the structure of the evaluation component. Inflexibility of vis guidance is a risk with more recent tools as well, such as CompassQL [42], which applies its rules regarding expressiveness implicitly, as well as SeeDB and Zenvisage [39], which embed their guidelines into their recommendation axes in such a way that the end user is unaware of which guidelines are at play. Draco democratizes these approaches by allowing its users to compose their own collections of guidelines through a constraint-based formalization, although users are encouraged to use carefully preconstructed collections [27]. Draco hides the underlying reasoning about various visualization choices, and precludes the production of unusual or novel visualizations as each chart type must be encoded into the recommendation model independently.

*e-mail: mcnutt@uchicago.edu
†e-mail:glk@uchicago.edu

We propose to follow a middle road between these approaches, in which visualizations are created with the aid of automation without requiring that users give up their usual scientific data workflow or their ability to tailor guidelines to specific needs. Tarrell et al. suggest that as systematic enumerations of guidelines emerge, such as VisGuides, we should be able to engage these rules programmatically [36]. Interestingly, Meeks borrows from software engineering the term *linting*, a method of automated software quality checking, to introduce the idea of "vis linting", along with a collection of associated lint rules (though without any concrete implementation) [26]. We seek to realize the aspirations of these works by creating a prototype of a programmatic interface for visualization evaluation by linting, not unlike the spell check for visualization called for by Moritz et al. [27]. Figure 1 schematically illustrates how we position vis linting as an automated analysis that respects existing scientists' workflows, while empowering them to make better visualizations.

Increasingly, the basic medium in which scientists and data scientists do and present their research is the *computational notebook*, such as those supported by Project Jupyter [21]. For example, the notebook at `https://github.com/minrk/ligo-binder` documents, in an accessible way, the extensive data processing required for the first observation of gravitational waves [23]. Another ubiquitous tool of visualization creation in scientific contexts is matplotlib, a Python library common in Jupyter notebooks [16]. We aspire to harness the open-source activity and enthusiasm around matplotlib and Jupyter to disseminate automated guidance about good visualization design, so our initial work has focused on matplotlib.

Our contributions here are conceptual and practical. First, we refine the idea of vis linting in terms of rules that are either purely computational, or that embody algebraic visualization design concepts. Second, we introduce `vislint_mpl`, a Python program for evaluating matplotlib visualizations, which demonstrates the practical feasibility of vis linting. Only a small number of lint rules have been implemented to date, but we are working now on adding more. Our `vislint_mpl` prototype, along with a collection of possible lint rules and implementation strategies, is available at `http://github.com/mcnuttandrew/vislint_mpl`.

2 Visualization Lint

2.1 Background of Lint

Since the late 1970s, lint systems have been used widely to enforce coding style and to catch rudimentary programming bugs [17]. A linter is a program that analyzes another program by applying an extensible collection of rules. The set of rules, and their parameter settings, can be configured to taste and selectively ignored as the situation might require. Linters are either run on demand, similarly to unit tests, or as a background process that provides graphical hints to the programmer as they work, similarly to a spell checker.

2.2 Lint for Visualization

Visualization lint, much like traditional lint, it is an automated framework for evaluating a visualization relative to a collection of predefined rules or guidelines. Beyond the static analysis that is conventionally understood as linting, we also envision analysis that executes the chart-creation code[1]. A visualization linting system is thus a configurable collection of functions (lint rules) that each take in a visualization, or code that produces a visualization, and evaluates it as passing, or else provides an explanation of why it fails. The language of lint for visualization is a natural evolution of the checklist approach to visualization guidelines delineated by Tarrell et al. [36]. Effective rules, like effective guidelines, are granular and can be evaluated unambiguously.

Linting automates the application of common sense ideas and principles. While an individual test may seem trivial (e.g. **require-chart-title**), adhering to a large collection of potentially unfamiliar rules can be cumbersome for a non-expert chart maker. We argue that it is easier to follow a rule if conformance to that rule can be mechanistically and immediately tested. For example, a standard guideline is to avoid coloring schemes that are ambiguous to colorblind viewers (**colorblind-friendly-colors**). The availability of colorblind-friendly options in chart creation systems like Datawrapper [1] demonstrate that authors need help identifying which color palettes are in fact colorblind-friendly. Yet this rule is also often overlooked, and perhaps it would be better respected if chart creation tools automatically checked for it. Similar automation could also implement rules that are usually ignored, such as enforcing the use of grayscale-friendly color schemes (**printable-colors**). It is not a great leap to restructure the guidelines presented in the Visualization Guideline Repository [40] and other locations as a collection of lint rules. As a demonstration of this concept, we include a rule list in our `vislint_mpl` repo that combines the guidelines of VisGuides, Meeks's lint rules [26], and some novel rules of our own.

Not all guidelines apply in all situations; rules suitable for one task may be incongruous for another [31]. For instance, Meek's **require-annotation** rule would appropriate for visualizations serving an *introduce* [28] purpose, but inappropriate for purely exploratory data analysis. We thus suggest that as creators of visualization methods and curators of best practices, we can assemble different rule collections for a range of common tasks. Users can then select a rule set based on their intended task, and benefit from the associated guidance generated by vis linting.

Spell checkers do not try to modify the content of text; they only help improve its quality. Analogously, vis linting does not try to create or substantively change a visualization, as a recommendation system might; it only seeks to improve the visualization the user is already building. For instance, vis lint would not prevent the construction of unusual or exotic charts types, such as Chernoff faces, instead simply providing tunable and ignorable commentary [6]. Linting has value independent of (and complementary to) vis recommendation systems, which might develop a model of the user over time, or suggest an ideal chart based on the given data, as described by Vartak et al. [39]. A linter operates within a fixed set of rules that do not change with user or data, it merely applies those rules in a rigorous and consistent manner. We suggest that such clear and opinionated-but-configurable feedback is a good method for steering scientists towards better charts, as it both reduces the mystery of the black-box recommendation system, and provides an informative enumeration of visualization design principles.

2.3 New Types of Lint Rules

The fact that lint works via execution of a program creates possibilities for either new vis guidelines, or new ways of reliably computing vis properties for evaluation with respect to a guideline, that would otherwise challenge purely human evaluation and application. We highlight two new categories of vis lint rules: computational and algebraic. At this stage we are outlining the types of analyses possible with linting, rather than defining exactly when or how these particular rules should be executed. For the computational rules, listed below, our thinking is that basic legibility and intelligibility tasks can be performed by machine, and charts failing these tests will likely be problematic for human observers.

- **legible-text**: Text in a chart should be legible enough that well-

[1] We suggest that differentiating between dynamic and static evaluation in contemporary usage is somewhat arbitrary, as they tend to be used in similar patterns. For example, unit tests, which must be dynamically evaluated, may either be run on demand, or as a background process (c.f. the "watch mode" of the Jest javascript test framework [12]). On the other hand, some linters can evaluate dependency graphs, which cannot be done statically within dynamically evaluated languages. We adopt the terminology of lint to convey the application of a pre-defined (though extensible) set of rules, rather than program-specific creation and evaluation of tests in the context of unit testing.

trained optical character recognition, such as Tesseract [33], should be able to read all labels correctly (as compared to the label text passed to the API). Our current implementation of this rule works on especially clear labels, but will need further training to handle labels overlying other marks.

- **no-complex-titles**: Having chart titles is important (c.f. Meek's **require-titles**), but chart titles should also not be too complex (depending on the audience). Our **no-complex-titles** rule judges title readability with a Flesch-Kincaid test [19], as implemented in the Python package textstat [2].
- **sufficient-data-ink-maximization**: Tufte's data-ink ratio is simple to understand but hard to measure by eye [37]. We intend to compute it by halving and doubling the amount of data charted, and measuring the pixel changes in the resulting images.
- **minimum-saliency**: One can set a minimum threshold saliency, for example if context requires that a chart catches the eye. This could be executed using the machine vision techniques discussed by Matzen et al. [25].
- **legible-graph**: Dunne et al. describe a mechanism for computationally monitoring graph legibility via easily computed metrics [11]. These can be framed as lint rules by placing thresholds on each of the metrics of interest, as with the crossing-angle and angular-resolution-min tests of Greadability [15].
- **visible-anomalies**: Correl et al. describe how chart configurations are susceptible to malicious settings [8]. This proposed rule handles the particular danger that a histogram with insufficient bins can hide gaps or outliers in a distribution [9].

Complementary to the computational lint rules above, the framework of Algebraic Visualization Design (AVD) [20] offers higher-level mathematical evaluations of visualization effectiveness. Some possible rules are:

- **data-dependent-chart**: charts should change when the data is changed, to avoid what AVD calls confusers. This can be tested by altering the input data (ideally in a way guided by the task at hand), and comparing the resulting images.
- **representation-invariance**: charts should not change when the data is transformed in superficial ways (e.g., re-ordering the numerical representation of categorical variables), to avoid what AVD calls hallucinators. Following the technique outlined by Karve [18], we currently implement this by permuting input data and comparing the resulting images.
- **no-connected-categories**: It makes no sense to draw connecting paths between per-category marks because there is no inter-category interpolation (whereas the same path drawing makes sense for connecting samples of an underlying continuous trend) [44]. This rule is a specific example of what could be generally described in AVD as failures of the data-visual correspondence principle.

3 Our Work

3.1 Implementation

Our prototype Python-based linter, `vislint_mpl`, evaluates visualizations created in matplotlib against a set of rules. After creating a matplotlib chart, the user passes the associated axes and figure objects, along with a configurable list of rules to be checked, to the `vislint_mpl` function. The user then receives a list of the rules that failed, along with an explanation of why. An example of the type of guidance provided can be seen in Fig. 2.

The implementation of the lint rules in our prototype typically involve the application of heuristics or inspection of various aspects of the matplotlib API. For instance, **representation-invariance** is

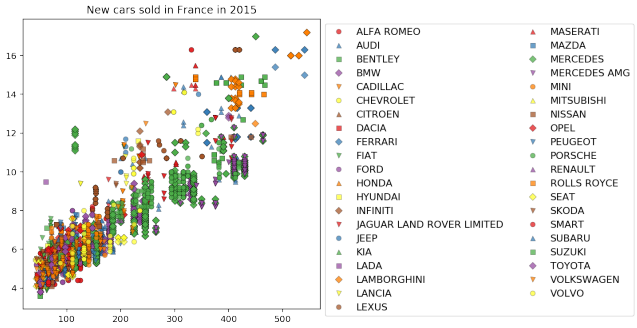

Figure 2: Given this chart, based on one in nb_266110.ipynb from the Github Jupyter notebook corpus [30], our `vislint_mpl` prototype finds failures of **representation-invariance**, **require-axes-labels**, **max-colors**, and **no-indistinguishable-series**, which should suggest the underlying problem to the chart creator: too many series. This could be fixed by splitting into small multiples with another variable, such as country of origin or vehicle type, or with interactive detail-on-demand. Whether or not there are automatable ways to create those solutions, the path to better visualizations can start with lint highlighting missteps.

implemented by permuting the order of both the series present and the order of the data within each of the series, and computing a pixel difference between the raster images of the resulting charts, while Meek's **maximum-pie-pieces** is implemented by inspecting the axes object for Wedge patches. We implement only a subset of the rules discussed above (and listed in our Github repository) to establish the proof of concept of our vis linting ideas.

We centered our initial work on matplotlib as it is ubiquitous among scientists (a recent survey [30] found that around half of the 1.25 million Python notebooks on Github imported matplotlib), and because it enables automated analysis on the elements of visualizations specifically. D^3, in contrast, can produce and manage entire web apps, and is less common among scientists [3]. With little or no modification to their current workflow, matplotlib users should be able to make use of this technology to produce better graphics.

3.2 Future Work

Our current work has focused on enhancing chart creation for technically proficient scientists. A natural next step will be to tailor our linting tools to familiar environments like Jupyter, as this will enable us to answer Karve's [18] call for AVD automation in the context of notebook interfaces. We aim to reach these goals by developing a graphical feedback linting system for Jupyter in the vein of the pluggable lint system in Github's Atom [14]. Based on that, we could construct an interface that gives in-situ hints to users as they are creating visualizations, flagging issues and suggesting more effective alternatives, much like a spell check for visualization. Further, extending our linter to analyze notebooks, and the entire cells of code within, would enable greater insight than possible with analyzing just matplotlib objects.

We would like to implement more lint rules in a greater variety of categories, such as algebraic rules that better capture the user's intentions. This may be facilitated by moving to linting a grammar-based tool like ggplot [41], Vega-Lite [32], or react-vis [4], as manipulating a more refined API would enable more insightful evaluations. The language environments of these tools, however, lack the unusually rich functionality that has facilitated our initial work in Python for matplotlib (such as the ready availability of machine vision capability), which may slow functionality, and may preclude the linting of non-standard chart types.

4 Conclusion

We have described visualization linting as a way to guide scientists and other chart creators towards effective visualizations, and we showed its feasibility with a working prototype, `vislint_mpl`, which evaluates matplotlib graphics against a variety of rules. We have suggested the programmatic evaluation offered by linting is actionable, flexible, and extensible, because it provides concrete commentary during the vis creation process, can be configured to taste and task, and can execute analyses based on machine vision and natural language processing. The future development of vis linting tools will enable chart creators to produce better visualizations within their current workflows with minimal extra work.

References

[1] G. Aisch. Datawrapper now checks your colors, so you don't have to. `https://blog.datawrapper.de/colorblind-check/`, 2018.

[2] S. Bansal. textstat. `https://github.com/shivam5992/textstat`, 2018. Accessed: 2018-07-10.

[3] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[4] A. Bulyonov, A. McNutt, and Uber Vis Team. react-vis. `https://uber.github.io/react-vis/`, 2016.

[5] M. Chen, G. Grinstein, C. R. Johnson, J. Kennedy, and M. Tory. Pathways for theoretical advances in visualization. *IEEE computer graphics and applications*, 37(4):103–112, 2017.

[6] H. Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American statistical Association*, 68(342):361–368, 1973.

[7] W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984.

[8] M. Correll and J. Heer. Black Hat Visualization. In *Workshop on Dealing with Cognitive Biases in Visualisations (DECISIVe), IEEE VIS*, 2017.

[9] M. Correll, M. Li, G. Kindlmann, and C. Scheidegger. Looks Good To Me: Visualizations As Sanity Checks. *IEEE Transactions on Visualization and Computer Graphics*, 2018. [To appear].

[10] A. Diehl, A. Abdul-Rahman, M. El-Assady, B. Bach, D. Keim, and M. Chen. VisGuides: A Forum for Discussing Visualization Guidelines. In J. Johansson, F. Sadlo, and T. Schreck, eds., *EuroVis 2018 - Short Papers*. The Eurographics Association, 2018. doi: 10.2312/eurovisshort.20181079

[11] C. Dunne, S. I. Ross, B. Shneiderman, and M. Martino. Readability metric feedback for aiding node-link visualization designers. *IBM Journal of Research and Development*, 59(2/3):14–1, 2015.

[12] Facebook. Jest. `https://jestjs.io/en/`, 2018. Accessed: 2018-07-15.

[13] S. Few. *Information dashboard design*. O'Reilly Sebastopol, CA, 2006.

[14] Github. Atom: The hackable text editor. `https://atom.io/docs`. Accessed: 2018-06-28.

[15] R. Gove. Greadability.js: Graph layout readability metrics. `https://github.com/rpgove/greadability`, 2018. Accessed: 2018-07-10.

[16] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55

[17] S. C. Johnson. *Lint, a C program checker*. Citeseer, 1977.

[18] A. Karve. Visualization without guesswork. `https://www.youtube.com/watch?v=jtUjnIIDvEw`, 3 2017.

[19] J. P. Kincaid, R. P. Fishburne Jr, R. L. Rogers, and B. S. Chissom. Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for Navy enlisted personnel. Technical report, Institute for Simulation and Training, University of Central Florida, 1975.

[20] G. Kindlmann and C. Scheidegger. An algebraic process for visualization design. *IEEE Transactions on Visualization and Computer Graphics (Proceedings VIS 2014)*, 20(12):2181–2190, Nov. 2014. doi: 10.1109/TVCG.2014.2346325

[21] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, et al. Jupyter Notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pp. 87–90, 2016.

[22] P. Lee, J. D. West, and B. Howe. Viziometrics: Analyzing visual information in the scientific literature. *IEEE Transactions on Big Data*, 4(1):117–129, 2018.

[23] LIGO Scientific Collaboration and Virgo Collaboration. Observation of Gravitational Waves from a Binary Black Hole Merger. *Phys. Rev. Lett.*, 116:061102, Feb 2016.

[24] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions On Graphics (Tog)*, 5(2):110–141, 1986.

[25] L. E. Matzen, M. J. Haass, K. M. Divis, Z. Wang, and A. T. Wilson. Data Visualization Saliency Model: A Tool for Evaluating Abstract Data Visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):563–573, 2018.

[26] E. Meeks. Linting Rules for Complex Data Visualization. `https://www.youtube.com/watch?v=_KEl-Spdaz0`, 5 2017.

[27] D. Moritz, C. Wang, G. L. Nelson, H. Lin, A. M. Smith, B. Howe, and J. Heer. Formalizing Visualization Design Knowledge as Constraints: Actionable and Extensible Models in Draco. *IEEE Transactions on Visualization and Computer Graphics*, 2018. [To appear].

[28] T. Munzner. *Visualization analysis and design*. AK Peters/CRC Press, 2014.

[29] W. Robison, R. Boisjoly, D. Hoeker, and S. Young. Representation and Misrepresentation: Tufte and the Morton Thiokol engineers on the Challenger. *Science and Engineering Ethics*, 8(1):59–81, 2002.

[30] A. Rule, A. Tabard, and J. Hollan. Exploration and Explanation in Computational Notebooks. In *ACM CHI Conference on Human Factors in Computing Systems*, 2018.

[31] B. Saket, A. Endert, and C. Demiralp. Task-Based Effectiveness of Basic Visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 2018. [To appear].

[32] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017.

[33] R. Smith. Tesseract Open Source OCR Engine. `https://github.com/tesseract-ocr/tesseract`, 2018. Accessed: 2018-07-11.

[34] Spotfire. Spotfire. `https://spotfire.tibco.com/`. Accessed: 2018-07-10.

[35] Tableau. Tableau. `https://www.tableau.com/`. Accessed: 2018-07-10.

[36] A. Tarrell, A. Fruhling, R. Borgo, C. Forsell, G. Grinstein, and J. Scholtz. Toward visualization-specific heuristic evaluation. In *Proceedings of the Fifth Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization*, pp. 110–117. ACM, 2014.

[37] E. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1983.

[38] E. Tufte. *Visual Explanations*. Graphics Press, 1997.

[39] M. Vartak, S. Huang, T. Siddiqui, S. Madden, and A. Parameswaran. Towards visualization recommendation systems. *ACM SIGMOD Record*, 45(4):34–39, 2017.

[40] VisGuides. Visualization guidelines repository. `http://visguides.repo.dbvis.de/guidelines.html`, 2018. Accessed: 2018-06-28.

[41] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009.

[42] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Towards a general-purpose query language for visualization recommendation. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, p. 4. ACM, 2016.

[43] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2016.

[44] J. Zacks and B. Tversky. Bars and lines: A study of graphic communication. *Memory & Cognition*, 27(6):1073–1079, 1999.