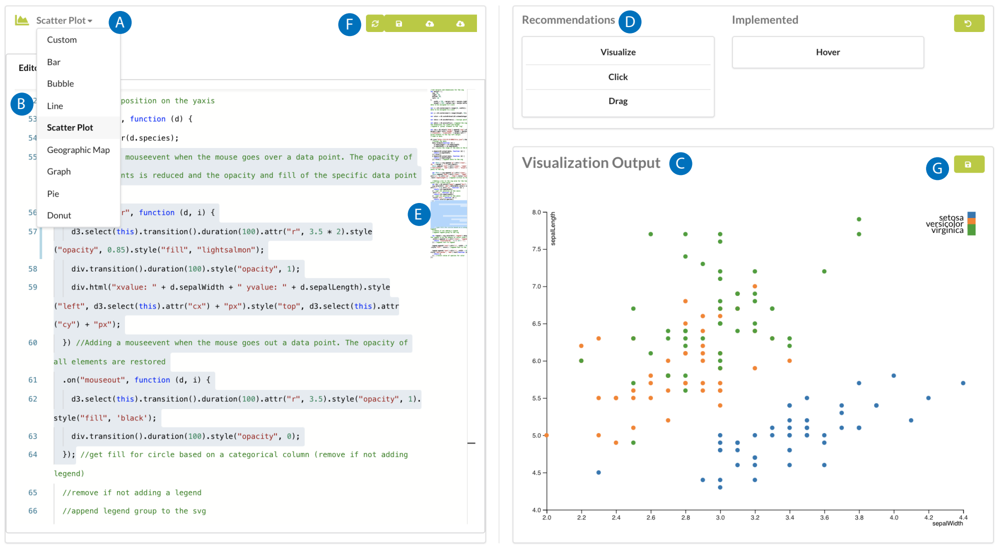
User-Driven Programming Support for Rapid Visualization Authoring in D3

HANNAH K. BAKO, University of Maryland
ALISHA VARMA, University of Maryland
ANUOLUWAPO FABORO, University of Maryland
MAHREEN HAIDER, University of Maryland
FAVOUR NERRISE, University of Maryland
JOHN P. DICKERSON, University of Maryland
LEILANI BATTLE, University of Washington



```
position on the yaxis
, function (d) {
r(d.species);
 mouseevent when the mouse goes over a data point. The opacity of
nts is reduced and the opacity and fill of the specific data point

r", function (d, i) {
    d3.select(this).transition().duration(100).attr("r", 3.5 * 2).style
("opacity", 0.85).style("fill", "lightsalmon");
    div.transition().duration(100).style("opacity", 1);
    div.html("xvalue: " + d.sepalWidth + " yvalue: " + d.sepalLength).style
("left", d3.select(this).attr("cx") + "px").style("top", d3.select(this).attr
("cy") + "px");
  }) //Adding a mouseevent when the mouse goes out a data point. The opacity of
all elements are restored
  .on("mouseout", function (d, i) {
    d3.select(this).transition().duration(100).attr("r", 3.5).style("opacity", 1).
style("fill", 'black');
    div.transition().duration(100).style("opacity", 0);
  }); //get fill for circle based on a categorical column (remove if not adding
legend)
  //remove if not adding a legend
  //append legend group to the svg
```

Fig. 1. An example of an interactive visualization designed using the Minerva system. A user can select a visualization such as a *scatterplot* to create from the **Template Panel (A)**. The code responsible for creating the *scatterplot* is displayed within the **Editor (B)** and the corresponding visualization rendered in the **Visualization Panel (C)**. A list of recommended interactions is displayed to the user in the **Recommendation Panel (D)**. A user can then choose a recommended interaction such as a *Hover* and the code responsible for implementing the interaction is added to the interface and highlighted for the user to view and modify **(E)**. Once a user is satisfied with their visualization, they can export the code and visualization using the **Controls (F)** and **(G)**.

D3 is arguably the most popular language for programming visualizations online. However, D3 requires a level of programming proficiency that fledgling analysts may not yet possess. Instead of asking users to switch to simpler languages or direct manipulation tools, we propose *automated programming support features* to augment users' existing skill sets. To this end, we mine D3 examples from the web and programmatically translate them into three automated features: **template completion** to quickly program initial visualizations; **recommendations** to suggest complementary interactions for a users' D3 program; and **code augmentation** to

Authors' addresses: Hannah K. Bako, hbako@cs.umd.edu, University of Maryland; Alisha Varma, alishav@umd.edu, University of Maryland; Anuoluwapo Faboro, afaboro@umd.edu, University of Maryland; Mahreen Haider, mhaider1@umd.edu, University of Maryland; Favour Nerrise, fnerrise@umd.edu, University of Maryland; John P. Dickerson, john@cs.umd.edu, University of Maryland; Leilani Battle, leibatt@cs.washington.edu, University of Washington.

2022. Manuscript submitted to ACM

implement recommended interactions with a single click. We demonstrate these features in Minerva, an automated development environment for programming interactive D3 visualizations. In a user study with 20 D3 users, we find that Minerva enables participants to program interactive visualizations in D3 with significantly less time and effort. All our data and code is shared on OSF:https://osf.io/97mru/?view_only=f259e07ca33441128d5fdc1fd862c369.

CCS Concepts: • **Human-centered computing → Visualization toolkits**.

ACM Reference Format:
Hannah K. Bako, Alisha Varma, Anuoluwapo Faboro, Mahreen Haider, Favour Nerrise, John P. Dickerson, and Leilani Battle. 2022. User-Driven Programming Support for Rapid Visualization Authoring in D3. 1, 1 (December 2022), 22 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Interactive visualizations are essential to gaining a deeper understanding of complex data [46]. Visualization specification languages are the primary substrate for creating interactive visualizations, because they enable users and systems to programmatically express a seemingly infinite range of visualization and interaction designs [40]. At the forefront of these languages is D3 [10], which is known for being one of the most influential and expressive languages for implementing interactive visualizations in the browser [4, 10, 20, 21, 25].

However, despite its popularity and benefits, D3 is still difficult to use, requiring extensive programming experience that many users may not (yet) possess [40, 47]. For example, creating a simple D3 visualization requires that in addition to knowing D3, a user has knowledge of basic web development languages (HTML, CSS), the Document Object Model (JavaScript), and oftentimes server side scripting to fetch data from remote sources. Some visualization tools circumvent this complexity by providing higher-level abstractions, such as Vega-Lite [48] and Plotly [28]. Others eliminate the need for language-based specification altogether by providing direct manipulation interfaces, such as Data Illustrator [34], Lyra [12, 59], Voyager [55, 56], and Data2Vis [13]. However by abstracting or eliminating specifications, these tools also sacrifice expressiveness, user control, and creativity [26, 40], all strengths of the original D3 language [10]. This challenge points to a core question that drives our work: *how can we help users harness the expressive power of a language like D3, regardless of how comfortable they are with the complexities of the language?*

One solution is to abstract popular visualization examples into general-purpose *templates* [39], and generate programming recommendations based on these pre-defined structures (e.g., [39, 54]). However, for complex languages like D3 that mix imperative and declarative specification [49], creating templates is not a straightforward process [31]. Furthermore, prior work focuses on creating and using templates within the boundaries of a specific system [21, 39], and it is not yet clear how templates can be generalized to code written *outside* of these pre-defined boundaries.

In this paper, we propose a *data-driven* and *user-driven* approach to the generation, recommendation, and integration of visualization code templates. With our approach, we identify and extract common template structures using the code of actual D3 users mined from the web. Then, we leverage our understanding of how D3 users apply these templates in the wild to develop *automated features* to (1) determine which code snippets to recommend to a user as they write code in real time, and (2) integrate these code snippets automatically into live user code. As a result, we can avoid abstracting away the complexities of D3 while still providing transparent and customizeable recommendations to implement a user's desired functionality, even for user code that does *not* follow a pre-defined visualization template. We designed our approach to be general purpose and applicable to most visualization specification languages, as long as the following properties hold true: there is a sufficient number of visualization examples from which meaningful code templates could be extracted; and the language can be represented as an Abstract Syntax Tree to support our program analysis techniques. As an example, both Vega [50] and Vega-Lite [48] meet these requirements.

We demonstrate the utility of our approach through Minerva (shown in Figure 1), an interactive programming environment for rapid implementation of interactive visualizations in D3. To understand how people implement visualizations in D3, we perform a detailed analysis of D3 examples shared on Bl.ocks.org [8] and Observable [9]. Based on our findings, we derive general-purpose templates for common visualization and interaction types implemented in D3. Using these templates, we developed three new automated support features: *template completion* to pre-populate a selected visualization template using appropriate data attributes from an uploaded dataset; *interaction recommendation* to suggest complementary interactions to add to the user's visualization, which we infer from the code's SVG output; and *code augmentation* to automatically update the user's code to implement a selected interaction recommendation, even when the user is *not* using a pre-defined visualization template.

In summary, this paper makes the following contributions:

- We present an **analysis of common programming patterns** for D3 visualizations and interactions based on 1,500 D3 examples from the web (section 4).
- We provide a **suite of general-purpose templates** for common D3 visualizations and interactions (section 6).
- We develop a **model for recommending interactions** to implement within a D3 program (section 7) and a **code augmentation component** to automatically integrate recommendations into the user's code (section 8).
- Finally, we implemented these features in the Minerva system and evaluate Minerva through **a user study with 20 D3 users** (section 9). Minerva enabled participants to implement interactive D3 visualizations in less time and with fewer iterations compared to the typical D3 implementation workflow used by participants (section 10).

2 RELATED WORK

Our work draws from four research areas: visualization authoring tools, recommendation tools, interactive visualizations and template and code generation. In this section, we review the relevant literature in each topic.

2.1 Visualization Authoring Tools

There is a plethora of visualization authoring tools available to users for producing complex visualization designs, including specification based toolkits (e.g [10, 28, 48, 50]), automated tools (e.g [12, 52, 59], PowerBI), or graphic driven tools (e.g [34]). Specification languages are expressive, but require considerable programming skills for users to fully utilize them [13, 27, 53]. On the other hand, graphic driven tools eliminate the need for programming by allowing the users to use interactive widgets within a GUI to design visualizations. However, by eliminating the need for specifications, they take away the control from the user [26] and still require great levels of effort for users to learn how to use them effectively [47]. Moreover, for visualization authors who want to use specification based authoring tools, these tools may not be of much help. Our approach aims to overcome this limitation by focusing on how to simplify the process of using these visualization languages instead of abstracting the visualization implementation process from it's users.

2.2 Recommendation Tools

Visualization recommendation systems assist users by providing suggestions of tasks or visualizations that help users achieve their visualization design goals [37, 55, 56]. Recommendation systems are either rule-based or model-based. Rule-based recommendation systems depend on established guidelines [7, 36] for data encoding to make recommendations [22, 55, 56]. Model-based systems however use a mixture of expert guidelines, visualization data, and machine learning techniques to determine the recommendations for users [13, 27, 35, 42, 57]. The models designed by these systems are typically trained on a dataset that captures the needs and interest of the target users at the time the data was collected. As a result, most model based recommendation systems ignore the contextual data that comes from factoring user intent and evolution of user interest within recommendation systems [19, 53].

A popular approach to model prediction problems includes the use of Markov decision problems (MDPs) [6]. MDPs are used for the modeling of stochastic decision problems where there are a (typically finite) number of states and actions that an actor can take. Shani et al. advocate for the use of MDPs to solve prediction problems as MDPs take into account the long term effects and value of user actions into account when picking a recommendation to display to users [51]. This form of reinforcement learning could be used to extend already existing modelling techniques to allow for further customization and support within recommendation systems. Given its success as a modelling technique in real-world recommendation and prediction problems [2, 18, 51], we model our recommendation system as an MDP.

2.3 Interactive Visualizations

Interactions are critical because they allow users to elicit a deeper understanding of the data presented [14]. Various visualization typologies classify interactions based on how interactions can be used in visualizations as well as in the creation of visualizations. Heer and Shneiderman provide a taxonomy of 12 task types for interactions within a visualization [23]. These tasks are categorized into three high-level categories: (1) Data & View Specification tasks that enable exploration of data by specifying views of interest, (2) View Manipulation tasks to highlight patterns and hypotheses and (3) Process & Provenance tasks to support iterative data exploration. Brehmer and Munzner [11] bridge the gap between high-level and low-level interaction task[s] classification by connecting complex high-level tasks as a sequence of simpler low-level tasks. These complex tasks are organized by the intent of the task being performed i.e why is a task being performed, how are these tasks carried out and what are the inputs and results of the task.

Research in the visualization community continues to highlight the importance of interactions [58]. However, recommendation systems focus on visualization authoring and only support a basic set of interactions within visualizations such as tooltips and click [37, 55]. One tool that attempts to remedy this problem is Interaction+ [41], which extracts information about visual objects from web pages, and uses this information to provide a variety of interactions which users can use to manipulate the existing visualization. While Interaction+ provides a novel approach to solving the interaction problem, the interaction implementation process is still abstracted from users. Minerva helps visualization authors gain a better understanding of how to incorporate interactions directly into specifications, by giving authors access to the actual code for the interactions. To model the space of possible interactions, we incorporate tasks from Brehmer and Munzner's typology [11] into Minerva's model design to provide a suite of interactions that users can implement.

2.4 Templates and Code Generation

Code reuse has been a strongly advocated and documented programming practice [32]. Templates have been leveraged to promote code reuse in programming by systems like MICoDe [33]. Within the visualization community, some tools utilize templates to generate visualizations [30, 38, 39] and style templates [21]. Recent work by McNutt et al. proposes using parameterized declarative templates, as a means of abstracting declarative visualization grammars towards promoting reuse, visualization exploration and ease the process of creating visualizations [39].

Templates have also been combined with Code synthesis algorithms to create tools that automatically infer a user's intent based on their direct manipulation input and generates the corresponding code output [15, 31, 54, 59]. Tools like Wrex [15], Falx [54], and Mage [31] provide readable code in response to users' interactions with GUI widgets while Lyra 2 [59] only focuses on using direct manipulation to make static charts interactive without exposing the underlying code to users. Our work extends this line of research of applying templates and code synthesis techniques for implementing visualizations. However our approach contributes techniques to automatically combine (1) multiple templates into a single interactive visualization and (2) integrate templates into already existing code.

3 MOTIVATING USAGE SCENARIO FOR MINERVA

Sandra is a student who wants to create some interesting visualizations for the final project in her data science course. Although this course was broad in scope, it lacked depth in terms of learning interactive visualization techniques. Sandra has heard that D3 can support a wide variety of interactive visualizations and experience with D3 could be a nice addition to one's resume. However, she has limited knowledge of D3. She wants to use D3, preferably with a simple dataset to start, such as the Iris dataset [1]. Sandra decides to use our open source tool Minerva, shown in Figure 1, to program her visualizations. Sandra goes to the Minerva website, imports her dataset, and chooses a scatterplot from Minerva's Template Panel (Figure 1A). Minerva automatically generates the D3 code for a scatterplot from the Iris data as shown in the Editor (Figure 1B), and the resulting visualization is rendered in the Visualization Panel (Figure 1C). Then, Minerva recommends a ranked list of interactions commonly implemented in scatterplots (Figure 1D).

Sandra clicks on the hover interaction from the recommendation panel and Minerva automatically extends her current D3 program with the corresponding code, highlighting the new code block and providing a summary of its behavior in the Editor (Figure 1E). The new code displays additional information about a data point when a hover event is triggered. Some of the data points are really close to each other; Sandra adds a zoom interaction from the interactions on the recommendation panel so she can zoom into tightly packed sections of the visualization. Once Sandra completes her visualization, she exports the code from the editor as a JS file and SVG of the visualization for her personal records.

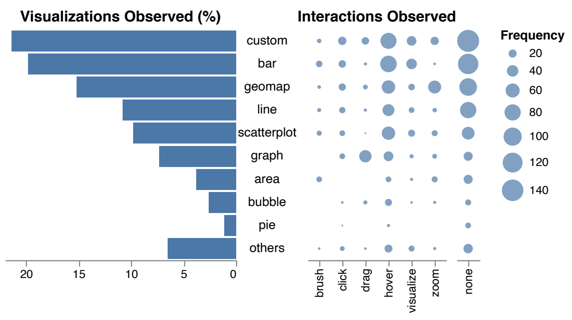
Using Minerva, a new D3 user like Sandra can quickly and iteratively implement interactive visualizations. Minerva eases the cumbersome process of writing D3 code, enabling such users to explore how different visualization and interaction types are programmed in D3 using a single interactive interface. Although this example focuses on a new D3 user (Sandra), it can also easily apply to a more advanced user. For example, a professional analyst who has used D3 in several projects may find it tedious to edit an old example from a previous project, and may still prefer to fill in one of Minerva's templates instead. We discuss the experiences of both new and more advanced D3 users in section 10.

4 ANALYZING EXISTING D3 DESIGNS

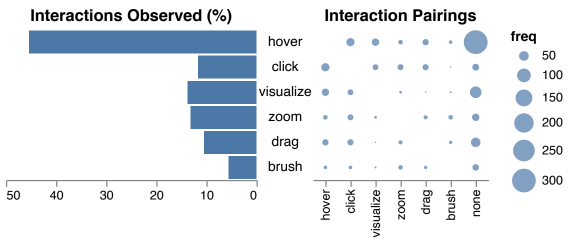
Simplifying the process of implementing visualizations using D3 requires that we first gain an understanding of *what types of visualizations D3 users are creating, and how these visualizations are implemented.* In this section, we discuss our process for collecting and analyzing implementations of D3 examples shared on the internet to answer these questions. We describe the data extraction and classification process as well as results and implications of our qualitative analysis.

4.1 Building a Corpus of D3 Examples

To understand how visualization users implement visualizations, we need to capture the breadth of visualizations implemented using D3. The first step of our analysis process was to collect a sufficiently large corpus of examples from the internet. We adopted Battle et al.'s approach of identifying "islands" of usage for D3, or specific websites with thousands of D3 examples [4]. We initially considered three sources: bl.ocks.org [8], Observable [9] and GitHub [44]. Given that the resulting corpus will be used as training data for automated features, we need the data to be of sufficiently high quality i.e. the code needs to compile, and produce at least one visualization. To collect and assess the data, we first downloaded a sample of 500 examples from each corpus to manually assess the quality of each data source. To maintain quality within our dataset, a visualization or interaction is only classified if (1) the example explicitly imports D3 and (2) the code in the example results in a rendered visualization in a browser. We found that the examples sourced



(a) Common Visualization-Interaction Pairings from Bl.ocks.org.



(b) Common Interaction Pairings from Bl.ocks.org.

Fig. 2. Results from analysis carried out on the bl.ocks.org dataset (a) presents the frequency of the top visualization types and the interactions implemented within them. Visualizations with 1% and less frequency are captured in the *"others"* category [Heatmap(1.0%), Voronoi(0.9%), Tree(0.8%), Sankey(0.8%), Sunburst(0.8%), Donut(0.7%), Parallel Coordinates(0.3%), Box Plot(0.2%), Word Cloud(0.3%), Hexabin(0.2%), Radial Chart(0.2%), Stream Graph(0.2%) and Waffle Charts(0.08%)] (b) shows the number of times different interaction types were implemented in the same visualization

from GitHub were of poor quality often containing incomplete code. 49.2% of the examples sourced from Observable were unrelated to D3 containing Vega-Lite visualizations[1], tutorials on data science tools[2], etc. In addition, unlike the (now deprecated) bl.ocks.org repository, exported Observable examples are designed to operate within the Observable environment rather than standard web projects, making it difficult to programmatically analyze these notebooks for D3 API usage. In contrast, the bl.ocks.org examples were of consistently higher quality than both the GitHub and Observable examples[3]. We believe this difference in quality stems from the maturity of the bl.ocks.org repository compared to Observable, as well as its intended use as a robust repository of D3 examples compared to GitHub. For these reasons, we focus on analyzing examples from bl.ocks.org for our analysis.

4.2 What visualization types do D3 users implement?

We qualitatively coded the corpus manually classifying what visualization(s) and interaction(s) are implemented in each example. Visualizations were classified using the classifications of prevalent D3 visualizations observed by Battle et al [4]. This resulted in 21 visualization types which we refer to as *standard* visualizations. We coded a total of 1500 D3 examples scrapped from Bl.ocks.org which resulted in 1228 viable visualizations for further analysis. Of these viable visualizations, 21.4% were diagrams, art, or highly specialized visualizations, such as a departures board for flights, braille clocks, candle stick graph, etc., which we classify as *custom* visualizations.

From our 1228 viable visualizations, 996 (81.1%) were standard D3 visualization types such as scatter plots, area charts, voronoi diagrams, etc.. Bar charts (19.8% n=251), Geographic maps (15.2% n=192), and line charts (10.8% n=137) were most prevalent in our sample, shown in Figure 2a. The top 5 standard visualizations i.e. Bar Charts, Geographic Maps, Line Charts, Scatterplots and Graphs account for 80.1% of all the non-custom visualizations implemented with 16 other visualization types accounting for 19.9% of examples in our sample highlighted in Figure 2a.

Our analysis of the Bl.ocks.org examples highlight that even though there are a lot of custom visualizations being implemented by users, majority of the visualizations still conform to the standard D3 visualizations on the internet.

[1]https://observablehq.com/@aldo/tuftes-charts-in-vega-lite
[2]https://observablehq.com/@thisistaimur/warc-study-analysis
[3]The coded examples are available on OSF: https://osf.io/97mru/?view_only=f259e07ca33441128d5fdc1fd862c369.

However, only 5 visualization types account for the vast majority of these standard visualizations. This suggests that we can support the needs of D3 users by focusing on the most popular visualizations from our data.

4.3 What interaction types do D3 users implement?

We classify interactions within these visualizations using Brehmer and Munzer's typology of visualization task[s] [11]. Of the 1500 examples we explored, 659 (43.9%) contained interactions. 859 interaction implementations were identified within these interactive examples, falling into 6 general widget types: Brush, Click, Drag, Hover, Visualize, and Zoom. The most popular interactions overall were Hover (n=390), Visualize (n=118) and Click (n=100), shown in Figure 2b. We observe the following categories of interactions in our dataset:

- **Select:** highlighting data points to emphasize salient information, often using "*Brush*", "*Hover*", and "*Click*" widgets. This category of interactions was the most common implementation we observed, showing up in 62.6% of all interactions implemented.
- **Encode:** changing which data attributes are encoded in a visualization, often using a GUI widget; we use the term *"Visualize"* to represent all such encoding interactions. Of the 859 interaction implementations we identified, 13.7% implemented an interaction in this category.
- **Navigate:** moving the focal point of a visualisation to center on a specific subset of points, often using "*Zoom*" and "*Pan*" widgets. 13.2% of the interactions implemented in the examples fell into this category.
- **Arrange:** sorting and organizing marks within the visualization, such as by using a "*Drag*" widget. 10.5% of the examples implemented the Drag interaction as a means of arranging visual elements in a visualization.

This analysis also revealed that certain interaction types were often implemented together in the examples. Examining these associations, we identify 39 distinct pairs of interactions with "Click and Hover" as the most frequent pairing representing 14% of all occurrences Figure 2b.

We observe a high number of interactions being implemented within visualizations. This is particularly encouraging as it highlights that users may follow recommended best practices when designing their visualizations [58].

4.4 How are these visualizations being Implemented?

A key step in clarifying the needs of visualization users is to understand how they implement visualizations. We qualitatively analyzed the D3 examples to understand how visualization users implement similar types of visualizations. For each visualization and interaction type, we examined the code structure of different visualization and/or interaction type implementations. We identified the API calls used within each implementation, the organization of code and the syntactical correctness of each example.

At the low level, users' programs vary for visualizations of the same type, such as by variable names, whitespace used, and so on. However, when the code is examined at the structural level, the overall structure and API calls remain the same. For example, the default code structure across visualizations begins with defining the boundaries and creating an SVG where subsequent visualization elements will be housed. This is followed with code that fetch data files and finally code to implement and bind the svg elements. The baseline code for the specific visualizations follows almost the same structure with changes made to fit the code to the specific dataset the user is trying to visualize. This creates many opportunities for code reuse, a common implementation strategy observed within the D3 community [5, 25].

To expand upon our observation, we measure how often popular interactions are implemented for each visualization type. We find that when broken down to individual visualization types, the types of interactions implemented vary with for all visualization types. Overall, Hover was the most frequently implemented for 19 of the 22 visualization types considered as seen in Figure 2b. Comparing across visualization types, we find that the percentage of examples with interactions for a particular visualization type varied greatly. For example 49% of the line chart examples implemented had no interactions implemented whereas for graph examples 82% of the examples had at least one interaction implemented. Comparing for the sets of interactions implemented for each visualization type, we see that the set of interactions implemented also varies. Using the line and area chart examples, in Figure 2 we see that line charts have a wider set of interactions implemented (brush, click, drag, hover, visualize and zoom) compared to area charts with only two interaction types (brush and hover).

Our analysis of how D3 users implement visualizations highlights a recurrent theme of widespread code reuse by users. We also observed that the set of acceptable interactions varies based on the specific visualization being implemented. Additionally, for certain visualization types (e.g. Graphs) including at least one interaction was the convention indicating that for certain D3 implementation[s], users could benefit from using multiple modes of interactivity to navigate and explore their visualizations. For example, the user may (for graphs) or may not (for bar charts) need to consider incorporating interactions and may want to use specific sets of interactions to match existing D3 examples.

4.5 Design Considerations

Analysing how users implement visualizations gives us insight into the interests, workflow and challenges that stem from using D3. These insights in tandem with previous work and our personal experience working with D3, provide design considerations for Minerva. We discuss how each of these considerations are incorporated into Minerva below.

C1: Code component reuse. Code reuse is a documented and advocated skill within the programming and D3 community [25]. Our analysis of D3 code on the internet in subsection 4.4 highlights similarity in overall structure of D3 code for visualizations types. D3 users tend to implement D3 visualizations in consistent ways, which are dictated by existing examples with only a few D3 visualizations deviating significantly from these established examples. As a result, we can speed up the D3 implementation process by automating the adoption of these established coding conventions for each visualization type and interaction type. Based on this premise we synthesize code *templates* from existing D3 examples to capture the already existing usage patterns of D3 users towards making our template easy to understand. This makes it easy for users working with our templates to quickly associate a code segment in the flow of a template with the specific task it performs. For example the definitions of the dimensions of a visualization are typically the first set of variables to be defined within a template. We also provide detailed descriptions of the purpose of each line of code via comments. These curated templates are enable as a feature within the Minerva system.

C2: Use numbers to drive adaptive recommendations. Our analysis in subsection 4.4 shows that a high number of D3 examples allow their end users to deeply explore the underlying data through interactions; a very important use of interactivity in visualizations [14]. However, we observed that there are many implementation conventions for different possible combinations of visualizations and interactions. D3 users interested in using interactions would have to sift through numerous examples to figure out the appropriate implementation convention for their visualizations. To save them the time and effort, we can instead automatically provide them with a curated list of appropriate interactions to implement for a given visualization type via recommendations. In this way, D3 users can focus more on implementing the program at hand, rather than having to scour the internet for relevant D3 examples to guide them.

The D3 community is one that is strong and thriving; new visualizations are being implemented and shared almost every day. The large number of D3 users provides an ample community from which data can be drawn upon and used to power recommendation engines. Minerva leans on the data generated by this community to power it's recommendation model by actively learning from the behaviour of its users to provide D3 users with recommendations of potential interactions for the current visualization a user is working on.

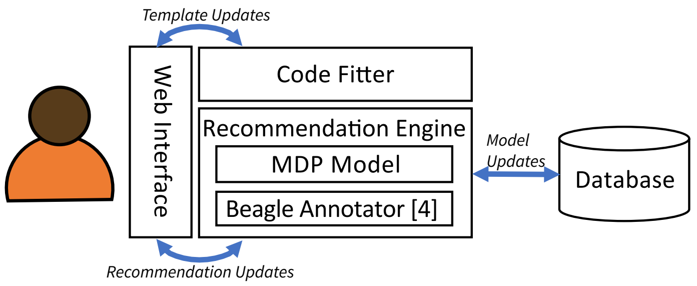


Fig. 3. Minerva's architecture. The Minerva interface (Figure 1) captures user requests for interactions and recommendations. Using the Code Fitter, it generates the appropriate D3 specification and renders the results for the user via the Interface. Requests are also sent to the prediction model to receive recommended interactions for the user.

C3: Familiar interfaces/workflows. Past research highlights a need for tools to be integrated into users workflows [15] and interfaces that users are familiar with [17]. Although our analysis does not lead directly to this conclusion, prior work suggest that visualization languages can potentially make a greater contribution to the visualization community at large by integrating more closely with users' established implementation workflows [5]. Furthermore, compatibility with other development platforms was one of the original design goals of D3 [10]. For these reasons, we prioritize compatibility with users workflow and other interfaces.

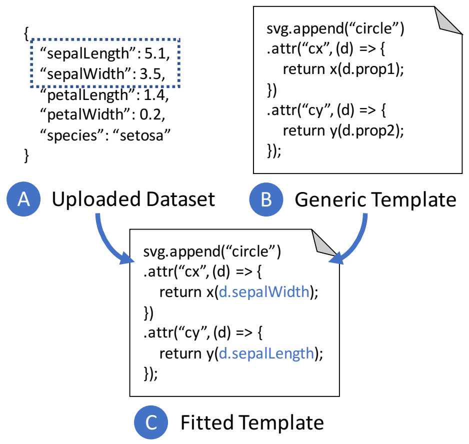
When designing visualizations in D3, visualization users typically write the specification within a code or text editor and then use a server to host their code as a web application rendered in the browser. In this manner the analyst has to consistently context switch, i.e. code writing mode and exploring the visualization in the browser. We lean on the live programming interface implemented in similar D3 editing interfaces [9] to implement this feature (see Figure 1). This enables users to actively monitor updates to their visualization and quickly catch bugs that have been introduced via their most recent code changes easing the challenges of debugging with languages like D3 [24]. In contrast to Observable's export feature, Minerva also allows users to directly export their code as pure JavaScript, allowing the exported code to be used directly in other development environments.

5 MINERVA SYSTEM OVERVIEW

Towards supporting our goal of helping users program D3 visualizations faster and more efficiently, we created three automation features to augment users skill sets. These features were implemented in a single interface called Minerva. In this section we describe the Minerva architecture depicted in Figure 3.

The Minerva web interface provides a user with an interactive graphical user interface through which they can interact with the Minerva system. The web interface consists of four core views: **Templates Panel** (Figure 1A), **Editor** (Figure 1B), **Visualization Panel** (Figure 1C) and the **Recommendation Panel** (Figure 1D). We selected some common visualization and interactions types from our analysis in section 4 to be supported in the Minerva system. Using our example in section 3, after Sandra uploads her dataset, she selects a bar chart from Minerva's list of templates, which triggers a request to the *Code Fitter* as shown in Figure 3. The *Code Fitter* (see section 8) then loads the corresponding template and fits it to Sandra's dataset. Sandra can see the updates to her code in the Editor.

Minerva provides recommendations of interactions that a user can implement in the current visualization. When a user selects a recommendation, the *Prediction Model* (see section 7) receives information from the interface about the current visualizations state to perform the necessary analysis to make predictions on recommended interactions for users. If recommendation requests are made for custom visualizations, the interface first sends the SVG elements for



```
{
  "sepalLength": 5.1,
  "sepalWidth": 3.5,
  "petalLength": 1.4,
  "petalWidth": 0.2,
  "species": "setosa"
}
```

```
svg.append("circle")
.attr("cx", (d) => {
   return x(d.prop1);
})
.attr("cy", (d) => {
   return y(d.prop2);
});
```

```
svg.append("circle")
.attr("cx", (d) => {
   return x(d.sepalWidth);
})
.attr("cy", (d) => {
   return y(d.sepalLength);
});
```

Fig. 4. When the user uploads a dataset (a) and selects a visualization template (b), the Minerva code fitter automatically selects compatible data attributes for the visualization and populates the template (c).

visualization to the *Beagle Annotator* [4] which makes predictions on what visualization the user is implementing. This information is subsequently used by the prediction model to generate interactions to be recommended to the users.

6 GENERATING D3 CODE FROM VISUALIZATION TEMPLATES

To facilitate our research goals, we rely on templates and programming techniques to automatically generate code for common D3 visualizations. This feature is housed in the Code Fitter component of the Minerva system. In this section, we discuss the techniques for synthesizing and automatically filling templates.

6.1 Constructing Templates

As with all programming languages, D3 has certain syntax to its specifications. We already observed in subsection 4.4 that most of the D3 code maintains the same structural organization. We also observed that the D3 API calls used within the examples remain consistent for visualization types. For example visualizations like scatterplots and bubble charts will contain calls to create circle marks. This common code organization and API calls are great indicators of the function of each line of code and the role they play in creating a visualization. Given these indicators, we analyse examples in our corpus to identify common code sections for each visualization type. These sections were manually extracted and combined to generate a working generic implementation of that interaction that can be modified to fit a dataset. These generic implementations are used as templates in the Minerva systems.

The process of creating templates for each interaction follows a similar format. Once key sections of code are identified for an interaction, we compile these sections into a generic template. We account for variations in how a specific interaction may be implemented for certain visualization types in these templates.

6.2 Fitting Templates to Datasets

Generic templates provide starter code for designers. However, users would still need to manually modify the code to bind their data for the visualizations to make sense. Minerva eliminates this burden by automatically fitting the code to bind provided data attributes to visual representations before presenting it to users. Given a selected visualization, the Minerva's Code Fitter first parses through the dataset to find the appropriate data attributes and then uses Abstract Syntax Tree (AST) manipulation to transform the generic template into customized specification a user can utilize.

Selecting Appropriate Attributes. Each visualization type supported by Minerva has constraints for the type of data it can encode. A line chart supports mapping of sequential and quantitative data while a Geographic Map requires geographic data for mapping. While quantitative data can be used for both linear and categorical mapping (discrete data), ordinal or nominal data can only be utilized for categorical mappings. Our Code Fitter considers each attribute in the source dataset to identify if the data type of an attribute is a suitable match for the specified visualization. The first match identified (and not currently implemented) is utilized as input to be mapped in the template (see Figure 4A). Users are able to manually change these attributes themselves in the *Editor*. For example, when Sandra requests for a scatter plot, the code fitter parses through her dataset to identify the data type for each attribute. The code fitter iterates through each attribute and arbitrarily identifies sepalLength, and sepalWidth attributes as the compatible data attributes for a scatter plot (Figure 4A). These attributes are selected to be visualized.

Modifying Templates with Attributes. Once the appropriate data attributes have been identified, the generic template Figure 4B is converted into its AST representation [29]. This AST is then traversed to find the nodes that are responsible for binding data to visual marks and replacing them with the selected data attributes. This new template is then returned to the user as custom D3 specification for the specified visualization. Continuing our example for a scatter plot, one of the data mapping can be found when determining the position of the circle marks. Our code fitter would modify such expressions so the name of the selected data attribute is passed as parameters for the call as seen in Figure 4C.

7 GENERATING RECOMMENDATIONS TO MAKE VISUALIZATIONS INTERACTIVE

Even when Sandra has chosen a visualization to create, she may still be unsure of how and whether to incorporate interactions such as Hover, Zoom, and Brush into her code. Users in a similar situation may need additional guidance on selecting appropriate interactions to incorporate. We expect that users would be interested in implementing similar types of interactions within their visualizations. So one approach to solving this is to provide recommendations of suitable interactions to users based on the implementation preferences of other D3 users. In this section, we provide an overview of the prediction models that drive the recommendation engine implemented in Minerva.

7.1 Predicting Which Interactions to Implement

Modeling Interaction States. To make recommendations, we first need an understanding of where the user is in her implementation process, and what her possible next steps may be in terms of selecting interactions to implement. We model the interaction selection process as a Markov decision process (MDP) [6]. The MDP predicts what interactions to recommend to the user based on the current state of the user's visualization code. We define the MDP for the Minerva recommendation engine as a set of *states S*, where each state $s \in S$ is a set of the interactions that have already been implemented in the user's code $\{i_1, i_2 ... i_n\}$. Given that the user could pick any number of interactions to implement, we create a separate state for every possible combination of interactions that we observed from Bl.ocks.org. Furthermore, this interaction set often starts out empty as we assume visualizations are initially static. For this reason, the empty state is always the initial state for our MDP model.

For any given state $s \in S$, our MDP model produces a ranked list of valid interaction recommendations, ordered by how likely they are to be implemented based on our Bl.ocks.org analysis in section 4.

Adjusting the Model Based on User Feedback. Given our interaction states, we model potential interaction implementation steps through a set of *actions A*, where each action $a \in A$ maps to a users reaction when an interaction i_r is recommended. The user can react to Minerva's recommendation i_r in one of four ways, given the user's current state s, and the corresponding future state s':

- **Accept** the recommendation by clicking on the corresponding button in the Minerva interface (see Figure 1D), thereby moving to state s', given by the equation:

$$P[s, i_r, s'] = \frac{observations(s')}{observations(s)}, \tag{1}$$

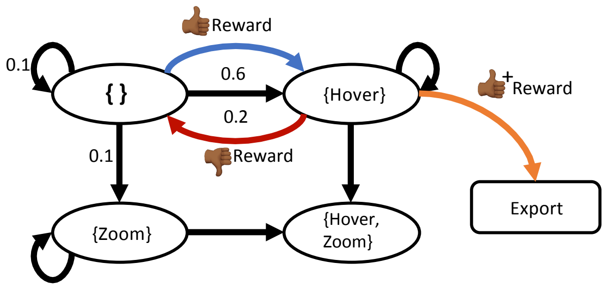


Fig. 5. The Minerva recommendation engine models user interaction state using a Markov decision process (or MDP) model. Here is a simplified example of how the Minerva MDP model is structured, where there are only two possible interactions zoom and hover, explained in subsection 7.2.

where *observations* computes the total observations of s (i.e., the combination of interactions) in our Bl.ocks.org dataset.

- **Export** the visualization to separate files (see (Figure 1E) and exiting the Minerva system. This is a cue that the user may be satisfied with her visualization and wants to save it for future use
$$P[s, i_r, export] = \frac{observations(s')}{\sum_{s \in S} observations(export|s)} \tag{2}$$
- **Undo** by adding and then removing an interaction using the undo button, meaning the user was dissatisfied
$$P[s, i_r, undo] = \frac{observations(s')}{\sum_{s \in S} observations(undo|s)} \tag{3}$$
- **Ignore** all of Minerva's recommendations I_r by not clicking on any recommendations in the interface, remaining in the same state s
$$P[s, i_r, s] = 1 - P[s, i_r, E] - P[s, I_r, s'] - P[s, i_r, undo] \tag{4}$$

Each time the user transitions to a different state, the probability distributions for outgoing transitions from s will be recalculated to account for this additional information. We normalize all corresponding probabilities to ensure that they sum to one. Note that since the total interactions that a user will implement is relatively small, these distributions can be computed easily.

Predicting Transitions Between Interaction States. We model how likely the user is to transition from one state s to another s' as transition probability between the two states. This is represented through a *transition function* T, which we use to compute the probability that an interaction i recommended in state $s_i \in S$ will lead to state $s_j \in S$, or $T(s_i|i_r, s_j)$. Given the user's current state $s \in S$, a recommended interaction from Minerva i_r, and a corresponding future state $s' \in S$, we define our transition function as the probability that a user will want to transition from s to s' if i_r is recommended:
$$T_r(s, i_r, s') = P[s, i_r, s'] \tag{5}$$
Note that an interaction will only be recommended if it has been observed before. However, we derive our states and transition probabilities using our observations from section 4, ensuring that our MDP model covers the majority of interactions that users often implement in D3.

When a user does or does not choose one of Minerva's recommendations, the MDP is rewarded or penalized accordingly. In this way, the MDP can adapt its behavior to better match the needs and behaviors of users. We model this behavior using a positive or negative *reward* R. We assume that the reward R for a user's reaction to recommendation i_r to be: a small positive number if the user **accepts** i_r, a large positive value if the **exports**, a large negative value if the user **undos**, and 0 if the user **ignores** i_r.

7.2 MDP Example.

We demonstrate the behavior of the Minerva recommendation engine using a simplified example based on our motivating scenario with Sandra from section 3. A simplified MDP model is depicted in Figure 5. In this example, let us assume that there are only two possible interactions that users will want to implement: zoom and hover. With two interactions, we will have four possible states for our model, shown in Figure 5: no interactions implemented (the initial state), zoom has been implemented, hover has been implemented, and all interactions have been implemented (the terminal state). The terminal state represents the scenario where Sandra has implemented all of the interactions possible.

Assuming Sandra starts in the initial empty state with her scatterplot, we can calculate initial transition probabilities, where four (fictitious) transition probabilities are provided in Figure 5, inspired by our analysis from section 4. The intuition behind these probabilities is as follows: Sandra will likely iterate on her scatterplot before incorporating any interactions (represented with probability 0.1), and Sandra is much more likely to implement hover than zoom interactions (probabilities 0.6 and 0.1, respectively). There is also a possibility that Sandra could implement a hover interaction but then change her mind and click on the undo button (represented with probability 0.2). For the empty state, our model provides a ranked list with two items, where the first item is hover, and the second item is zoom.

Now, suppose that Sandra clicks on the first recommendation of hover, representing the **accept** reaction. In this case, the MDP model receives a small positive reward for providing a useful recommendation, represented by the blue line in Figure 5. However, suppose that Sandra **undos** her last added interaction, this results in a large negative reward to the MDP model, represented by the orange line in Figure 5.

7.3 Evaluating the Model's Recommendation Accuracy

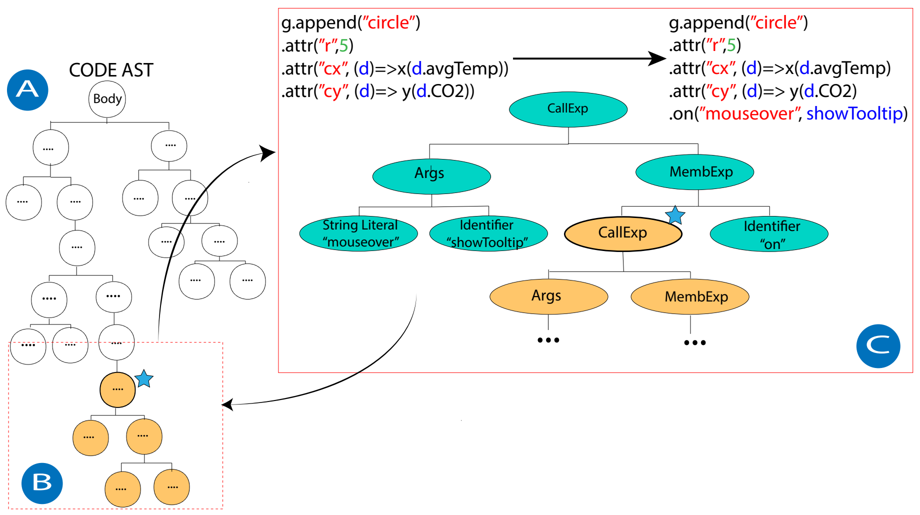
If the MDP model is not accurate in making predictions on training data, then its recommendations are unlikely to be useful in live D3 implementation sessions. Therefore, we test the MDP model accuracy prior to conducting a full evaluation of the Minerva system with actual D3 users (section 9), which we describe here. To ensure that Minerva's recommendations align with our observations from section 4, we performed leave-one-out cross validation [16] of the MDP model using our coded dataset. For each visualization example from our dataset, assuming the initial empty state, we classify Minerva's recommendations as correct only if they include the interactions implemented within this example. Overall, we find that the MDP model provides strong results, on average 76% accuracy. In some cases, we find that Minerva's accuracy can be improved, for example for zooming interactions (60% accuracy). These cases can be improved in a straightforward way in future work by adding more input observations for the model to train on. However, this evaluation does not take into account users' reactions to recommendations which, will improve the model's predictions over time.

7.4 Predicting Visualization types

We described how the recommendation engine supports interaction predictions when the user selects a visualization template first. However, our implementation in the Minerva system can also support recommendations even when the user does not select a template. To do this, Minerva utilizes the Beagle Annotator developed by Battle et al. [4] to enable real-time prediction of which type of visualization the user is currently implementing. With this annotator, Minerva can predict if a user is implementing a scatterplot, or some other visualization type by extracting the SVG elements of the visualization and running it through the Beagle Annotator's decision tree classifiers.

8 AUGMENTING USER CODE TO IMPLEMENT INTERACTIONS

Modelling users design goals help us understand how to recommend relevant interactions. However, these recommendations alone would not be very useful to visualization designers. Users would still have to search for and implement these



```
g.append("circle")
.attr("r",5)
.attr("cx", (d)=>x(d.avgTemp))
.attr("cy", (d)=> y(d.CO2))
```

```
g.append("circle")
.attr("r",5)
.attr("cx", (d)=>x(d.avgTemp)
.attr("cy", (d)=> y(d.CO2)
.on("mouseover", showTooltip)
```

Fig. 6. When a user selects a *Hover* interaction, the AST for the users code is generated (A). The Anchor node is identified in (B) and changed to include the new code for the interaction in (C).

interactions themselves. A major goal for our work is to reduce the challenge of matching visualization specifications with corresponding interaction specifications in existing D3 examples through automation. A user should be able click a single button to automatically add an interaction to her design.

Given a recommended interaction selected by the user, Minerva needs to be able to augment the existing code in the Editor (Figure 1B) to include the relevant code for this interaction. The code augmentation process begins with the creation of high quality templates for each interaction supported in the Minerva system (described in subsection 6.1). These generated templates are used to perform the necessary code transformations through graph rewriting operations on the Abstract Syntax Trees (ASTs) of the users code. This process can be broken down into four distinct steps (1) identify relevant variables, (2) generate templates, (3) identify anchor points, (4) update code

Identifying the Input Variables for the Interaction Template. Once the code has been converted to AST [43], Minerva identifies the components within the AST needed to make a viable template. The AST is traversed to search within functions or variable declarations for the needed components. For example if fitting a *Hover* interaction on a scatter plot, Minerva assumes that at the end of a *Hover* event, the color of a mark in the visualization would need to be restored to its original value. Hence it parses through the users code for the value assigned to the color of the target mark. If a variable is not found within the users code, a default value is assigned.

Populating the Interaction Template. Once all the needed variables have been identified, they are fitted into the templates AST. This process is similar to the modifying templates with attributes process explained in subsection 6.2. All API calls are updated with the right variable names and parameters as needed. For example in a hover interaction, the "mouseout" event will be updated to return the color of a mark to the original color specified in the code.

Identifying Anchor Points in the AST for Insertion. Anchor points signify nodes within the AST which need to be changed in order to accommodate the new code for the interaction. For example to add the Hover interaction, a new node has to be created and added to the code section responsible for creating visual marks. As such Minerva parses through the AST to identify the key node for the corresponding code sections. This node will represent the anchor point for this transformation (see Figure 6B).

Inserting the Populated Template into the AST. Once the template node has been generated, we insert it into the AST effectively transforming it into a new sub-tree. Inserting the template node involves either pre-pending, replacing or appending the new template to the anchor node forming a completely new sub tree. The nature of the template node and semantic correctness of the code are taken into consideration when determining which of the insertion actions to perform. For our *Hover* interaction, we would need to perform a replace action to make the template AST a parent node of the anchor node as seen in Figure 6C. Once the template has be inserted, the new AST is converted back into code and presented to the user, highlighting the newly added code blocks.

9 EVALUATION: USER STUDY DESIGN

In previous sections, we introduced three automated features implemented within Minerva with the aim of reducing the burden of visualization implementation for D3. In this section we focus on evaluating if these automated features lead to measurable improvements in how D3 users implement interactive visualizations [45, 47]. Our study was approved by the authors institutions IRB and preregistered on aspredicted.org[4].

Participants. We recruited 20 participants from mailing lists and social networking sites (4 female, 15 male, 1 non-binary) between the ages of 18 to 44. To ensure that all participants had enough basic understanding of D3 to be able to complete the study tasks under the time constraints, we required participants to have at least 3 months of experience working with D3.js. 11 of our participants had 3 months to 1 years experience, 3 had 2 - 3 years experience, 4 had 4 - 5 years experience and 2 were experienced D3 users with 5+ years of experience (see Table 1). Participants self-reported to be proficient in creating visualizations. Each subject spent approximately 1 hour and 30 minutes in our study and was compensated with a $20 Amazon gift card.

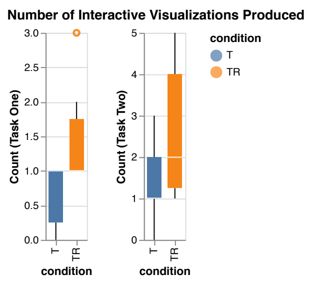
Tasks. To observe how participants use these automated features in various visualization implementation contexts, participants were asked to perform two tasks for the study [45, 55]. Note that we did not restrict what visualization or interaction types could be implemented; participants could implement whatever they chose as long as they met the requirements for each task.

- *Task One: Targeted visualization design:* For this task, participants were asked to use the Cherry Blossoms dataset to create a visualization meeting specific design requirements. The output visualization[s] had to meet two requirements: target users of the output visualization should be able to (a) identify correlations between two specific variables, and (b) interact with the individual data records represented in the visualization.
- *Task Two: Exploratory visualization design:* Participants were asked to choose between the popular Cars[5] and Iris [1] datasets, and then brainstorm what type of visualization they would like to implement. Then, participants were asked to implement this visualization and include at least one interaction in their visualization.

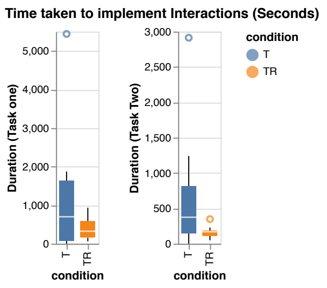
Experiment Conditions. Participants experienced two different Minerva setups across our tasks, resulting in four total experiment conditions. In one setup (baseline), participants only had access to Minerva's visualization templates, but not Minerva's interaction recommendations. In the second setup, participants had access to all of Minerva's features. Condition order was counterbalanced across participants. We tuned our experiment conditions through a pilot study, described below.

Pilot. We conducted a pilot study to fine tune our initial protocol. One important parameter we sought to tune was task difficulty, in particular whether participants should be asked to implement D3 visualizations entirely from scratch as one of the task conditions (the common case), or if participants should always be provided with Minerva's templates. In the pilot study, all participants (2 female, 1 male) were familiarized with Minerva and were asked to perform the

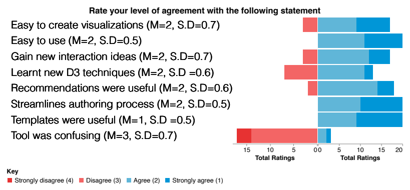
[4]Pre-Registration available at https://aspredicted.org/blind.php?x=cc46nd
[5]https://www.kaggle.com/abineshkumark/carsdata



(a) Count of interactive visualizations produced by participants. T = Templates only, TR = Templates + Recommendations



(b) Time taken by participants to implement a single interaction in visualization. T = Templates only, TR = Templates + Recommendations



(c) Distribution of responses to the post study questionnaire. All responses are on a 4 point likert-scale: 1 (strongly agree), 2 (agree), 3 (disagree) and 4 (strongly disagree).

Fig. 7. Results from analysis of data collected during evaluation of Minerva

tasks described above. For task one, we provided access to Minerva's visualization templates for two of the three pilot participants; the remaining participant implemented visualizations from scratch. Participants were able to use both the predefined templates and recommendation panel for task 2.

We observed that the two participants who had templates provided only used Minerva's templates while the third participant copied code from a sample visualization they found online. The participant who had to write their D3 code from scratch spent 45 minutes implementing a single visualization and was unable to implement any interactions, thus unable to complete the task. The participants who were provided with templates took less than 15 minutes to create a single interactive visualization for the same task. These results indicated that it would have been difficult to conduct a one-hour experiment if participants had to create visualizations from scratch, and the experiment time could be drastically reduced if Minerva's templates were always provided. Hence, we decided to provide Minerva's templates for all tasks and experiment conditions. However users could always opt to write their own code from scratch if desired.

Protocol. Participants were required to complete a demographic survey to determine eligibility for the study. Each session started with a Pre-study session where participants were familiarized with the study, and signed the consent form. They then proceeded to a training session where the experimenter walked them through the interface functionality via a prerecorded demo of the interface. Participants were then given 5 minutes to explore the full Minerva interface themselves and ask any question they had about the interface. Once participants were comfortable with the interface, they proceeded to complete our two study tasks. The experimenter limited guidance to only situations where the user reached a complete standstill and was unable to figure out the next step. Participants were allowed to use any external resource they wanted such as Stack Overflow or Bl.ocks.org. No time limits were placed for any of the tasks. Once all the tasks were completed, the participants completed a short exit survey and interview. Each session took an average of one hour to complete.

Data Collection. For each session, an experimenter observed the participants while they performed their tasks and took notes. Logs of participants' interactions with Minerva were also recorded through the Minerva back-end, which captured all click and keyboard events. We collected data from the post-task survey on participants' ratings of Minerva's features to capture participant feedback. Finally, video and audio were recorded to capture participants' interactions with the interface and responses to the exit interview questions.

Participant	Experience	T1 Condition	T1 Interactive Visualizations	T2 Condition	T2 Interactive Visualizations	New D3 Technique Learnt
1	3 months - 1 year	T	Scatterplot + Zoom	TR	Scatterplot + Visualize + Click + Zoom, Bar chart + Visualize, line + Brush	parsing/ formatting Dates in D3
2	3 months - 1 year	T	Scatterplot + Click	TR	Scatterplot + Click	Creating Axes on charts
3	4 - 5 years	T	Scatterplot + Hover	TR	Scatterplot + Brush + Zoom, + Drag + Visualize	-
4	3 months - 1 year	T	Bar chart + Hover	TR	Bar chart + Hover + Visualize	parsing/formatting Dates
5	3 months - 1 year	T	Scatterplot+ Hover	TR	Scatterplot + Hover + Visualize	Hover interaction
6	3 months - 1 year	T	Scatterplot + Hover	TR	Scatterplot + Visualize + Hover	Parsing/formatting Dates, Hover interaction
7	4 - 5 years	T	Scatterplot + Hover	TR	Scatterplot + Visualize	Parsing/formatting Dates (R)
8	5+ years	T	line + Hover	TR	Scatterplot + Hover + Visualize	inline parsing
9	4 - 5 years	T	Scatterplot + Hover	TR	Scatterplot + Hover; Scatterplot	-
10	2- 3 years	T	Bar chart + Hover	TR	Scatterplot + Visualize + Hover + Click + Drag	Force charts
11	2- 3 years	TR	line + Hover	T	Bar chart	-
12	3 months - 1 year	TR	Scatterplot + Hover	T	Scatterplot + Visualize	-
13	4 - 5 years	TR	Scatterplot + Hover	T	Bar chart + Visualize, Hover, Click	Creating axes, tooltips
14	3 months - 1 year	TR	Scatterplot + Hover	T	Scatterplot + Hover, Click	Parsing/Formating dates (R)
15	3 months - 1 year	TR	Scatterplot + Hover + Visualize + Click	T	Bar chart + Hover	Hover interactions
16	2- 3 years	TR	Scatterplot + Hover	T	Bar chart + Hover	-
17	5+ years	TR	Scatterplot + Click	T	Scatterplot + Click, Hover	Switching scales
18	3 months - 1 year	TR	Bar chart + Hover + Click + Visualize	T	line + Hover + Click, Scatterplot + Click	-
19	3 months - 1 year	TR	Scatterplot Visualize + Hover	T	Bar chart + Hover	Hover interaction
20	3 months - 1 year	TR	Scatterplot + Hover	T	Bar chart + Hover	axis definition

Table 1. Participants experience, interactive visualizations created for both tasks and new D3 techniques they described learning while working with Minerva. Data under the *"New D3 Technique Learnt"* column annotated with *"(R)"* are for participants highlighted in orange, are those who described not learning any new D3 technique but were reminded of some D3 technique. [Terms: T1 (Task One), T2 (Task Two), T (Templates Only), TR (Templates + Recommendations)]

10 EVALUATION: ANALYSIS & RESULTS

Here we analyze the results of our study focusing on the variety of visualizations implemented, survey responses and qualitative feedback. We fit linear mixed-effects models [3] to evaluate four hypotheses described in the subsequent subsections.

10.1 Visualization Templates Enable Users to create a Higher Number of Executable Visualizations

To assess the efficiency of Minerva in helping users create visualizations, we hypothesized that users produce a higher number of executable visualizations when provided with visualization templates compared to when users create visualizations from scratch **(H1)**. However, based on our observations from our pilot study we could not run our study without templates and instead provided templates for all participants with an added option of creating visualizations from scratch if desired. While we do not formally test for H1, we have informal support for this hypothesis based on feedback from participants. We expected that some participants would want to write their own code to complete tasks and provided an option to support this. Contrary to our expectations, every participant in the user study defaulted to using the templates provided by Minerva even though they were aware of the custom code option. This provides support that templates indeed have the potential to improve the visualization implementation process. Participants found these templates to be helpful ($\mu = 1$, $\sigma = 0.5$, see Figure 7c) and shared that they often created personal collections of D3 templates to reuse for visualizations. For example, P9 with 4-5 years of D3 experience stated, *"So I have, like, all of my little coding projects here ... so I would take one of these, and probably just copy it and kind of just rely on my previously done projects"*. This indicates a pertinent need for a boilerplate code to be provided so that users have a baseline to quickly iterate over multiple visualizations.

10.2 Templates and Recommendations enable creation of more interactive, executable visualizations

We hypothesized that users produce a higher number of interactive visualizations provided with visualization templates and interaction recommendations, versus when only visualization templates are provided **(H2)**. To assess this hypothesis, we collected data on the number of interactions implemented for each visualization created during the study. We fit linear mixed effects models with experiment conditions and the order in which the condition was presented as the fixed effects, and participants and datasets were used as the random effects. We test for significance by comparing the full and null models using Likelihood ratio tests and find a significant effect ($\chi^2(3, N = 20) = 13.406, p = 0.011$). Templates+recommendations allowed participants create 1.9 more interactions per visualization compared to only 0.7 interactions when only using templates (see Figure 7a).

Participants found the recommendations provided by Minerva useful(Median(M)=2, $\sigma = 0.55$) and instrumental in helping them gain new ideas of interactions to implement (Median(M)=2, $\sigma = 0.64$) as seen in Figure 7c. Our qualitative analysis of participants' feedback highlight that the presence of recommendations in Minerva increased interactivity within created visualizations and assisted participants in exploring interactions that can be used within visualizations. For example, P7 with 4 to 5 years of D3 experience states *"Recommendations gave me new ideas of interactions to be implemented...interaction mode, really, I think helps explore the dataset"*.

10.3 H3: Templates and Recommendations enable faster creation of visualizations

We assess whether users spend less time implementing interactive visualizations when provided with templates and interaction recommendations, compared to when users only have access to templates **(H3)**. We recorded the time taken to fully implement a single interaction in a visualization for each participant. We fit linear mixed effects models with condition and condition ordering as a fixed effect and participant and datasets as the random effects. We find that the interface layout had a significant effect on the time taken to create visualizations ($\chi^2(1, N = 20) = 3.85, p = 0.0499$). Participants implemented interactions 11 minutes faster when both templates and recommendations were provided. In comparison, participants spent 16 minutes per interaction when only templates are provided (see Figure 7b).

Our survey results reveal that participants believe automated features streamline the visualization implementation process in D3 (Median(M)=1.5, $\sigma = 0.51$ see Figure 7c). Participants shared their previous workflow of having to first search for example interaction code online, before implementing interactions within their visualizations. This workflow makes adding interactions a tedious process and in some cases may discourage users from adding interactions completely. However, Minerva made it possible for interactions to be integrated with a single Click thereby streamlining the process. For example P19 with 3 months - 1 year D3 experience commented-*"Previously, I used [different text editor]... And it's more limited in terms of how much you can externally use the interactions. So I would have liked to use d3. But it was too hard to get it to work within my time constraints. So yeah, I think if I could have had this it would have made that possible."*

10.4 H4: Users' Find D3 to be More Useful and Interesting When Given Automated Features

Finally we hypothesized that users find D3 more interesting and more useful when provided with both visualization templates and interaction recommendations, versus when users only have visualization templates **(H4)**. Testing this would involve comparing how participants perceive D3 as a visualization tool before and after using Minerva. Another would be to examine if users were able to learn new D3 techniques after using Minerva.

To evaluate if participants found the features to be useful for increasing interest in D3, we asked participants to give examples of new D3 techniques they learnt while using Minerva. 12 Participants were able to identify at least one new D3 technique that they hadn't been exposed to before. For example P5 with 3 months - 1 year D3 experience, explained learning how to create interactions using only D3 *So actually, I did not know how to create a Hover in pure d3, I use HTML. So with Minerva, I learned how can you create such interactions without using any HTML.* 2 participants gave examples of techniques they were reminded of and 6 participants stated they did not learn anything new from the system.

We observed that majority of the participants who were able to learn new techniques were novice D3 users (3months - 1 year) (shown in Table 1). This indicates that our approach provides a great avenue for novice visualization users to learn how to use D3 to create visualizations. For P9 who instructs data students, teaching D3 is a difficult task as students would typically need to learn HTML, CSS, Javascript and D3 all at the same time, causing them to be overwhelmed. A tool like Minerva would be instrumental in helping his students focus on learning the basics of D3 without being burdened with learning all these other languages at once. *"I also teach people d3. And I think this would be great... Having* *something like this, that creates the chart and the code and comments the code, so you can read the code. They would like cry tears of joy to have this."*—P9, 4 to 5 years of D3 experience.

10.5 Opportunities for Refining Minerva

Participants clearly saw value in using a system like Minerva. For example, P5 said *"I could definitely see myself using this, I can try multiple templates pretty quickly. So it's like a rapid exploration."* However, our approach is not perfect, and participants made suggestions for further refinement. Participants expressed the need for insight on why certain recommendations are made, e.g., P10 said *"instead of having the recommendation shift, just give more insight as to why this is being recommended, basically"*. Participants also suggested including more widgets to support further customization of D3 code, e.g., P15 said *"...maybe adding something like the components, to support selecting an axis or legend. If those are also given as part of templates to completely customize code, that might be more helpful"*. Surprisingly, some participants did not seem to understand the value of interactions. For example, when asked if interactions were effective, P17 said *"I mean, if you want to add an interaction, this is definitely helpful ... But in general, I think I prefer non interactive visualizations"*. This finding suggest that our community still has work to do in conveying the importance of interactions to visualization creators.

11 DISCUSSION & FUTURE WORK

In this paper, we present a new approach to help users harness the expressive power of complex visualization languages like D3. We contribute three automated features to help users program D3 visualizations with less time and effort: (1) a suite of *data-driven and user-driven templates* for implementing common D3 visualizations and interactions, (2) a *recommendation engine* for suggesting complementary interactions to add to a D3 visualization, and (3) *automatic code augmentation* to incorporate suggested code snippets to live user code, even when users are *not* following a pre-defined template. We implemented our approach in a prototype system called Minerva. In a user study with 20 D3 users, we find that our automated features enable users to create interactive visualizations faster and in fewer iterations compared to the typical D3 development workflow (i.e., when automated features are not available).

Our research demonstrates how reusable templates can be extended to make complex visualization languages like D3 more accessible. Specifically, our research enhances existing D3 infrastructure (e.g., [9, 30]) and extends prior work in template development (e.g., [39]) by providing automated recommendations to help users go from programming static to interactive D3 visualizations, and code augmentation to automatically implement recommendations within live user code, even for user programs that fall outside the purview of existing visualization systems.

Even though our approach was designed with D3 in mind, the principles behind our work can applied to any visualization language. For example, if the target language can be represented using Abstract Syntax Trees and sufficient examples can be sourced, then our template designs and recommendation engine could be re-trained to support the target language. In this way, our approach can also apply to alternative languages such as Plotly [28], Vega-Lite [49] and Vega [50].

In the future, we envision a transfer learning strategy where inferred visualization best practices from one language could be translated to other languages. For example, we could adapt models trained on D3 examples to generate example visualizations and automated recommendations for Vega-lite [49]. However, a users' needs also depend on her experience. In the future, we aim to automatically detect a user's experience level through her interactions with her development environment (e.g., Minerva), which would enable us to reveal or alter the presentation of features to match her current abilities. In this way, the visible feature set can automatically evolve to meet the user's needs over time.

REFERENCES

[1] E Andersen. 1935. The Irises of the Gasp e Peninsula. *Bulletin of the American Iris Society* 59 (1935), 2–5.

[2] Ori Bar El, Tova Milo, and Amit Somech. 2020. Automatically Generating Data Exploration Sessions Using Deep Reinforcement Learning. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1527–1537. https://doi.org/10.1145/3318464.3389779

[3] Dale J Barr, Roger Levy, Christoph Scheepers, and Harry J Tily. 2013. Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of memory and language* 68, 3 (2013), 255–278.

[4] Leilani Battle, Peitong Duan, Zachery Miranda, Dana Mukusheva, Remco Chang, and Michael Stonebraker. 2018. Beagle: Automated extraction and interpretation of visualizations from the web. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–8.

[5] Leilani Battle, Danni Feng, and Kelli Webber. 2021. Exploring Visualization Implementation Challenges Faced by D3 Users Online. arXiv:2108.02299 [cs.HC]

[6] Richard Bellman. 1957. A Markovian Decision Process. *Indiana Univ. Math. J.* 6 (1957), 679–684. Issue 4.

[7] Jacques Bertin. 1983. *Semiology of Graphics*. University of Wisconsin Press.

[8] Michael Bostock. 2016 (accessed November, 2019). *Popular Blocks*. http://bl.ocks.org

[9] Michael Bostock and Melody Meckfessel. 2016 (accessed July, 2021). *Observable*. https://observablehq.com/

[10] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics* 17, 12 (2011), 2301–2309.

[11] M. Brehmer and T. Munzner. 2013. A Multi-Level Typology of Abstract Visualization Tasks. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2376–2385.

[12] H Carr, P Rheingans, H Schumann, Arvind Satyanarayan, and Jeffrey Heer. 2014. Lyra: An Interactive Visualization Design Environment. In *Eurographics Conference on Visualization*, Vol. 33. Citeseer, 10.

[13] Victor Dibia and Çağatay Demiralp. 2019. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE computer graphics and applications* 39, 5 (2019), 33–46.

[14] E. Dimara and C. Perin. 2020. What is Interaction for Data Visualization? *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2020), 119–129.

[15] Ian Drosos, Titus Barik, Philip J Guo, Robert DeLine, and Sumit Gulwani. 2020. Wrex: A Unified Programming-by-Example Interaction for Synthesizing Readable Code for Data Scientists. In *Proceedings of the 2020 CHI conference on human factors in computing systems*. 1–12.

[16] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The elements of statistical learning*. Vol. 1. Springer series in statistics New York.

[17] James Gale, Max Seiden, and Çağatay Demiralp. 2020. Sigma Worksheet: Interactive Construction of OLAP Queries. *arXiv preprint arXiv:2012.00697* (2020).

[18] Radhika Gaonkar, Maryam Tavakol, and Ulf Brefeld. 2018. *MDP-based Itinerary Recommendation using Geo-Tagged Social Media: 17th International Symposium, IDA 2018, 's-Hertogenbosch, The Netherlands, October 24–26, 2018, Proceedings.* 111–123. https://doi.org/10.1007/978-3-030-01768-2_10

[19] David Gotz and Zhen Wen. 2009. Behavior-driven visualization recommendation. In *Proceedings of the 14th international conference on Intelligent user interfaces*. 315–324.

[20] Jonathan Harper and Maneesh Agrawala. 2014. Deconstructing and Restyling D3 Visualizations. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (Honolulu, Hawaii, USA) *(UIST '14)*. Association for Computing Machinery, New York, NY, USA, 253–262. https://doi.org/10.1145/2642918.2647411

[21] J. Harper and M. Agrawala. 2018. Converting Basic D3 Charts into Reusable Style Templates. *IEEE Transactions on Visualization and Computer Graphics* 24, 3 (2018), 1274–1286. https://doi.org/10.1109/TVCG.2017.2659744

[22] Frederick Hayes-Roth. 1985. Rule-Based Systems. *Commun. ACM* 28, 9 (Sept. 1985), 921–932. https://doi.org/10.1145/4284.4286

[23] Jeffrey Heer and Ben Shneiderman. 2012. Interactive Dynamics for Visual Analysis. *Commun. ACM* 55, 4 (April 2012), 45–54. https://doi.org/10.1145/2133806.2133821

[24] Jane Hoffswell, Arvind Satyanarayan, and Jeffrey Heer. 2016. Visual debugging techniques for reactive data visualization. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 271–280.

[25] E. Hoque and M. Agrawala. 2020. Searching the Visual Style and Structure of D3 Visualizations. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2020), 1236–1245. https://doi.org/10.1109/TVCG.2019.2934431

[26] Eric Horvitz. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 159–166.

[27] Kevin Hu, Michiel A Bakker, Stephen Li, Tim Kraska, and César Hidalgo. 2019. Vizml: A machine learning approach to visualization recommendation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–12.

[28] Plotly Technologies Inc. 2015. *Collaborative data science*. Montreal, QC. https://plot.ly

[29] Joel Jones. 2003. Abstract Syntax Tree Implementation Idioms. *Pattern Languages of Program Design* (2003). http://hillside.net/plop/plop2003/Papers/Jones-ImplementingASTs.pdf Proceedings of the 10th Conference on Pattern Languages of Programs (PLoP2003) http://hillside.net/plop/plop2003/papers.html.

[30] Erick Katzenstein. 2016 (accessed August, 2021). *D3.Js/Live*. https://d3js.live/#/

[31] Mary Beth Kery, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wongsuphasawat, and Kayur Patel. 2020. mage: Fluid Moves Between Code and Graphical Work in Computational Notebooks. *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Oct 2020). https://doi.org/10.1145/3379337.3415842

[32] Miryung Kim, Lawrence Bergman, Tessa Lau, and David Notkin. 2004. An ethnographic study of copy and paste programming practices in OOPL. In *Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004. ISESE'04*. IEEE, 83–92.

[33] Yun Lin, Guozhu Meng, Yinxing Xue, Zhenchang Xing, Jun Sun, Xin Peng, Yang Liu, Wenyun Zhao, and Jinsong Dong. 2017. Mining implicit design templates for actionable code reuse. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 394–404.

[34] Zhicheng Liu, John Thompson, Alan Wilson, Mira Dontcheva, James Delorey, Sam Grigg, Bernard Kerr, and John Stasko. 2018. Data Illustrator: Augmenting Vector Design Tools with Lazy Data Binding for Expressive Visualization Authoring. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3173697

[35] Y. Luo, X. Qin, N. Tang, and G. Li. 2018. DeepEye: Towards Automatic Data Visualization. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 101–112.

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

[37] J. Mackinlay, P. Hanrahan, and C. Stolte. 2007. Show Me: Automatic Presentation for Visual Analysis. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1137–1144.

[38] Michele Mauri, Tommaso Elli, Giorgio Caviglia, Giorgio Uboldi, and Matteo Azzi. 2017. RAWGraphs: A Visualisation Platform to Create Open Outputs. In *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter* (Cagliari, Italy) *(CHItaly '17)*. Association for Computing Machinery, New York, NY, USA, Article 28, 5 pages. https://doi.org/10.1145/3125571.3125585

[39] Andrew M McNutt and Ravi Chugh. 2021. *Integrated Visualization Editing via Parameterized Declarative Templates*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3411764.3445356

[40] Honghui Mei, Yuxin Ma, Yating Wei, and Wei Chen. 2018. The design space of construction tools for information visualization: A survey. *Journal of Visual Languages & Computing* 44 (2018), 120–132.

[41] Min Lu, Jie Liang, Yu Zhang, Guozheng Li, Siming Chen, Zongru Li, and X. Yuan. 2017. Interaction+: Interaction enhancement for web-based visualizations. In *2017 IEEE Pacific Visualization Symposium (PacificVis)*. 61–70.

[42] Dominik Moritz, Chenglong Wang, Greg L Nelson, Halden Lin, Adam M Smith, Bill Howe, and Jeffrey Heer. 2018. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 438–448.

[43] Open Source. [n.d.]. *Babel*. https://babeljs.io/docs/en/

[44] Tom Preston-Werner, Chris Wanstrath, P.J Hyett, and Scott Chaconl. 2008 (accessed July, 2019). *GitHub*. https://github.com/about

[45] Donghao Ren, Bongshin Lee, Matthew Brehmer, and Nathalie Henry Riche. 2018. Reflecting on the evaluation of visualization authoring systems: Position paper. In *2018 IEEE Evaluation and Beyond-Methodological Approaches for Visualization (BELIV)*. IEEE, 86–92.

[46] Bahador Saket, Hannah Kim, Eli T. Brown, and Alex Endert. 2017. Visualization by Demonstration: An Interaction Paradigm for Visual Data Exploration. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 331–340. https://doi.org/10.1109/TVCG.2016.2598839

[47] Arvind Satyanarayan, Bongshin Lee, Donghao Ren, Jeffrey Heer, John Stasko, John Thompson, Matthew Brehmer, and Zhicheng Liu. 2019. Critical reflections on visualization authoring systems. *IEEE transactions on visualization and computer graphics* 26, 1 (2019), 461–471.

[48] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 341–350.

[49] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 341–350.

[50] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. 2016. Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (2016), 659–668.

[51] Guy Shani, David Heckerman, and Ronen I Brafman. 2005. An MDP-based recommender system. *Journal of Machine Learning Research* 6, Sep (2005), 1265–1295.

[52] Chris Stolte, Diane Tang, and Pat Hanrahan. 2002. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics* 8, 1 (2002), 52–65.

[53] Manasi Vartak, Silu Huang, Tarique Siddiqui, Samuel Madden, and Aditya Parameswaran. 2017. Towards visualization recommendation systems. *ACM SIGMOD Record* 45, 4 (2017), 34–39.

[54] Chenglong Wang, Yu Feng, Rastislav Bodik, Isil Dillig, Alvin Cheung, and Amy J Ko. 2021. Falx: Synthesis-Powered Visualization Authoring. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.

[55] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. 2016. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (2016), 649–658.

[56] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2017. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 2648–2659.

[57] Cong Yan and Yeye He. 2020. Auto-Suggest: Learning-to-Recommend Data Preparation Steps Using Data Science Notebooks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1539–1554. https://doi.org/10.1145/3318464.3389738

[58] J. S. Yi, Y. a. Kang, J. Stasko, and J. A. Jacko. 2007. Toward a Deeper Understanding of the Role of Interaction in Information Visualization. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1224–1231.

[59] Jonathan Zong, Dhiraj Barnwal, Rupayan Neogy, and Arvind Satyanarayan. 2020. Lyra 2: Designing interactive visualizations by demonstration. *IEEE Transactions on Visualization and Computer Graphics* (2020).