

Integrated Visualization Editing via Parameterized Declarative Templates

Andrew McNutt
University of Chicago
Chicago, IL

Ravi Chugh
University of Chicago
Chicago, IL

ABSTRACT

Interfaces for creating visualizations typically embrace one of several common forms. *Textual specification* enables fine-grained control, *shelf building* facilitates rapid exploration, while *chart choosing* promotes immediacy and simplicity. Ideally these approaches could be unified to integrate the user- and usage-dependent benefits found in each modality, yet these forms remain distinct.

We propose *parameterized declarative templates*, a simple abstraction mechanism over JSON-based visualization grammars, as a foundation for multimodal visualization editors. We demonstrate how templates can facilitate organization and reuse by factoring the more than 160 charts that constitute Vega-Lite’s example gallery into approximately 40 templates. We exemplify the pliability of abstracting over charting grammars by implementing—as a template—the functionality of the shelf builder Polestar (a simulacra of Tableau) and a set of templates that emulate the Google Sheets chart chooser. We show how templates support multimodal visualization editing by implementing a prototype and evaluating it through an approachability study.

CCS CONCEPTS

• **Human-centered computing** → **Visualization systems and tools**; **Graphical user interfaces**.

KEYWORDS

Information Visualization, Declarative Grammars, Templates, User Interfaces, Ivy, Systems

ACM Reference Format:

Andrew McNutt and Ravi Chugh. 2021. Integrated Visualization Editing via Parameterized Declarative Templates. In *CHI Conference on Human Factors in Computing Systems (CHI ’21)*, May 8–13, 2021, Yokohama, Japan. ACM, New York, NY, USA, 18 pages. <https://doi.org/10.1145/3411764.3445356>

1 INTRODUCTION

Every user interface design involves compromise. Which tasks should be made easy at the expense of making other tasks cumbersome or even impossible?

There are several common user interface modalities for creating visualizations, each with distinct trade-offs [20]. *Chart choosers*

(as in Excel) allow users to rapidly construct familiar visualizations at the expense of flexibility. *Shelf builders* (as in Tableau) facilitate dynamic exploration but can obstruct the construction of specific chart forms or the addition of visual nuances. *Textual programming* is highly expressive but can impede rapid exploration.

A modality which may be well suited to the initial stages of a task, may become cumbersome in subsequent phases. For example, while a low-configuration approach for rapid data exploration might suffice at the beginning of their work, the user might subsequently require a more flexible—even if higher friction—interface that allows them to tune specific details of their chart. Graphical user interface (GUI) systems, such as chart choosers and shelf builders, can leave experts wanting more precise control over the chart creation process, while textual systems can leave novices in need of assistance. Without the ability to move between modalities, users are stuck with each interface’s shortcomings. None of the existing single-mode interfaces simultaneously achieve each of several goals (as in Fig. 2): ease of use (G1), explorability (G2), flexibility (G3), and ease of reuse (G4). These deficits can force users to switch tools across their analysis [32], or compel proficient users to seek ad hoc solutions that are difficult to repeat in different contexts.

Ideally, interfaces of varying complexity could be integrated such that both novice users (for whom chart choosers are often best suited [21]) and experts (whose most profitable interface will vary) obtain the benefits of each modality as their tasks require. Unfortunately this territory remains under-explored, as visualization systems tend to prefer one-size-fits-all designs. Declarative visualization grammars are an enticing starting point as they provide significant flexibility for specifying visualizations as text (G3). However, they lack the abstraction mechanisms found in full-featured programming languages. This paper considers the question:

Can we extend declarative grammars with abstraction mechanisms for reuse (G4), in a way that facilitates explorability (G2) as in shelf builders and ease of use (G1) as in chart choosers?

To answer to this question, we propose an abstraction mechanism called *parameterized declarative templates*, or simply *templates*, which extend textual, declarative grammars—specifically, those in which specifications are defined using JavaScript Object Notation (JSON) [15]—with mechanisms for reusing chart definitions. As depicted in Fig. 1h, templates abstract “raw” declarative specifications with parameters that specify data fields for a visual encoding (e.g. Color) and design parameters (e.g. height and width), making them more easily reused (G4). Templates can be rapidly instantiated for different data sets, shared among communities, and modified to taste—alleviating barriers to opportunistically [8] leveraging the rich body of grammar-based charts found online. Templates, which are essentially functions in a simple programming language with variables and conditionals, are described formally in Sec. 4.2.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CHI ’21, May 8–13, 2021, Yokohama, Japan

© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-8096-6/21/05...\$15.00

<https://doi.org/10.1145/3411764.3445356>

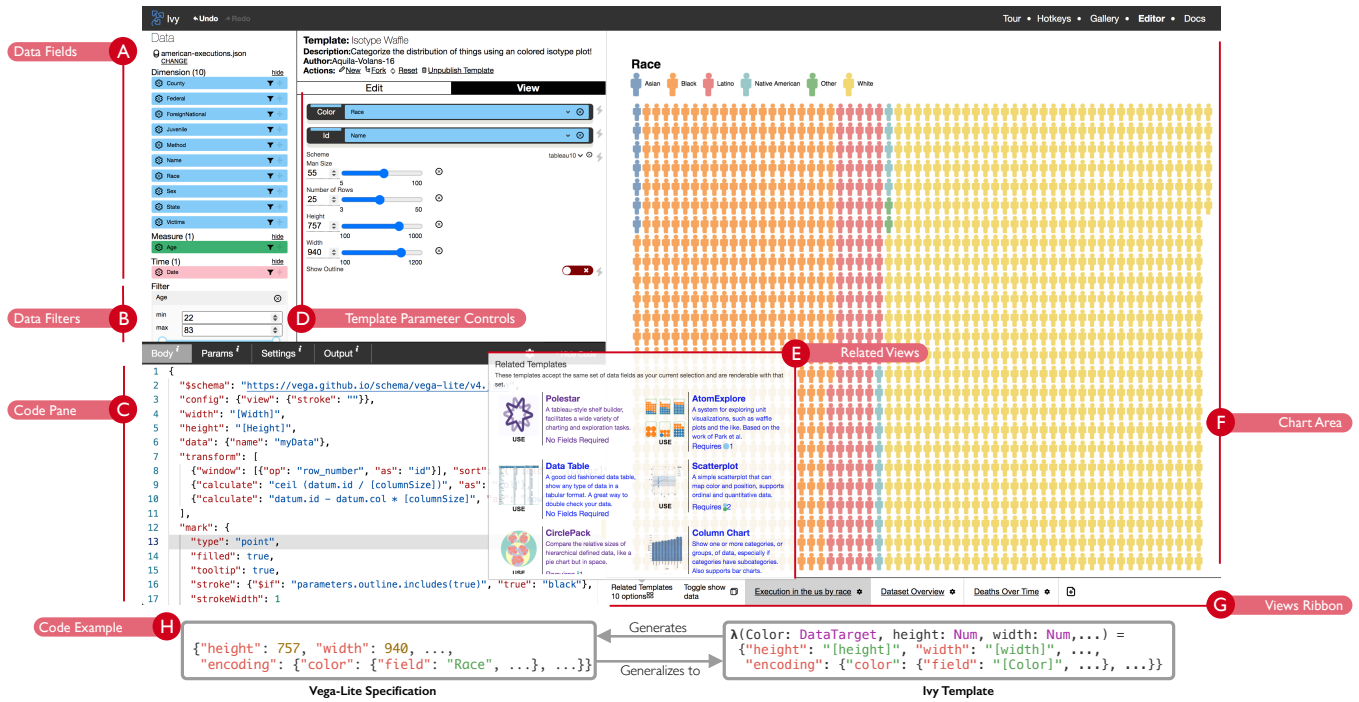


Figure 1: The Ivy visualization editor combines textual specification with GUI-based shelf building and chart choosing. Here, an “Isotype Waffle” template is used to visualize the people executed in the USA under the death penalty by race since 1977 [9]. Templates are functions with typed parameters that abstract JSON specifications in declarative visualization grammars.

We systematically apply this idea in a prototype visualization editor, called Ivy, in which templates are created and instantiated through text- and GUI-based manipulation (G1). Templates abstract over arbitrary JSON-based visualization grammars—our implementation currently supports Vega, Vega-Lite, Atom [62], and a toy data table language—allowing users to easily move between them (G3). We implemented an Ivy template that recreates the functionality of the shelf builder system Polestar [97, 98], that serves as the default view of the system, facilitating explorability (G2). Our user interface design is described in Sec. 4.3.

The systematic application of templates furthermore enables two notable interaction features beneficial to exploration (G2). *Catalog search* utilizes standard type-based compatibility checks to implement an extensible—if simple—recommendation system. *Fan out* facilitates rapid exploration within a template by juxtaposing multiple chart configurations on demand. These features are described in Sec. 4.4.

To evaluate how templates may serve as a foundation for multi-modal visualization editing, we performed two investigations. We considered how they might usefully reproduce and compress extant families of charts, first by factoring the 166 unique examples that constitute the Vega-Lite gallery into 43 templates, and then by reconstructing the 32 charts of the Google Sheets chart chooser as 16 templates. We then conducted a small approachability study of our prototype Ivy system which demonstrated that, with some training and guidance, users are able to create and instantiate templates by mixing the available modalities. We present these results in Sec. 5.

Although the main ideas regarding templates—abstraction over value definitions—and their instantiations are familiar concepts, the contribution of this paper is to explore and evaluate how these ideas may help integrate what are currently disparate approaches for creating visualizations. The result of this exploration is our Ivy prototype, which exhibits the promise of this approach. This composition of modalities provides several potential opportunities, including borrowing and adapting external examples (which many shelf builders lack), rapid exploration of parameter combinations (which programmatic interfaces often lack) and self-service chart creation (which chart choosers typically lack). Compared to textual programming in existing declarative grammars, the abstraction layer provided by templates simplifies the process of chart reuse and therein reduces code clones (as in our chart gallery reproductions).

Our prototype is available at <https://ivy-vis.netlify.app/>. The appendix further details our evaluation and implementation, while other supplementary materials can be found at <https://osf.io/cture/>.

2 RELATED WORK

Software systems for creating visualizations can be classified into a variety of interaction modalities [20, 59]. Among these, we aim to bridge three of the most common: chart choosing, shelf building, and textual programming. We now discuss declarative grammars and visualization editors, as they relate to the key ideas in this paper: (1) to endow declarative visualization grammars with basic abstraction mechanisms, and (2) to design a multimodal UI based on templates for creating and editing visualizations.



Figure 2: Chart making modalities have distinct strengths and weaknesses. Using *parameterized declarative templates*, Ivy strives to combine the strengths of several common modalities in this design space.

2.1 Declarative Visualization Grammars

Declarative grammars of graphics have proven extremely popular [23, 24, 42, 45, 49, 53, 62, 64, 66, 76–78, 80, 81, 83, 91, 92, 94, 95], with different approaches providing more expressiveness than others for particular tasks. Compared to full-featured, procedural visualization programming languages [6, 7, 35], declarative visualization languages trade fine-grained control over how to render a view for concise, expressive means to specify what to render. The idea behind parameterized declarative templates is to tilt these specification languages slightly back towards the full-featured languages, by adding some basic programming abstraction mechanisms.

Many declarative grammars [42, 45, 49, 62, 76–78, 81] adopt JSON as their specification format, which allows chart definitions to be easily consumed by various environments and tools. Vega and its surrounding ecosystem exemplify these benefits, which we seek to amplify by making JSON-based grammar specifications easier to reuse and explore. A related effort is Harper et al.’s [25] system for converting D3 charts [7] into reusable Vega-Lite specifications, which they also dub templates. Their templates provide abstractions related to ours, but are restricted to a limited subset of D3 charts. Ivy templates abstract arbitrary JSON and thus support any JSON-based visualization grammar.

2.2 User Interfaces for Visualization

2.2.1 Chart Choosers. This prevalent technique starts with selecting a desired chart form from a (potentially large) set of chart types, and then customizing it through a (usually limited) set of options. They are ubiquitous among analytics tools, such as spreadsheets, and are often found in visual analysis (VA) environments [2, 13, 60, 82]. This workflow facilitates an approach to charting held by many novices [21], but can also lead to premature commitment [22].

Ivy provides a gallery of templates, some of which constitute single distinct chart forms, as in conventional chart choosers. Some systems, like Ivy, provide APIs for extending the selection of charts [17, 56], but, unlike Ivy, they tend not to allow these changes from within the tool. Some systems [13, 56, 90] provide social features, wherein users can create, share, and modify charts. Ivy users can publish, fork, and remix templates through a shared template server.

2.2.2 Shelf Builders. In shelf builders, users map data columns to visual attributes, typically in a manner that is motivated by a principled visualization framework, such as VizQL [24] or the *Grammar*

of Graphics [95]. Tableau [80] and Charticulator [67] are prominent examples of this paradigm. Data fields of the current dataset are often represented as draggable “pills” that can be placed onto “shelves,” each of which denotes different aspects of the visualization (such as horizontal position or color).

We designed the GUI controls in Ivy for choosing arguments for template parameters to resemble the visual conventions of Polestar [97, 98], itself a facsimile of Tableau [80]. Shelf builders support a range of tasks, including presentation [30, 34, 73, 74] and exploration [80, 97, 98]. We facilitate the latter by constructing an Ivy template—IvyPolestar—that emulates and expands upon the Polestar application. This “default” template provides a familiar shelf builder interface upon application startup.

Many VA tools are backed by JSON-based declarative grammars (such as Lyra [74, 102] or Voyager [97, 98]), yet do not allow users to modify the underlying specifications. Restricting modifications to what can be created within such tools misses a significant opportunity, as there is a wealth of online knowledge and examples that ought to be utilized [8]. In commercial VA environments (such as Tableau), the disconnect from the underlying specification can cause fine-grained editing capabilities to be relegated to deeply-nested drop-down menus, which can preclude feature utilization. We address these issues by making our templates malleable, so that users can adapt the interface to their needs.

2.2.3 View Exploration in Shelf Builders. Shelf builders often prominently feature affordances for rapid data exploration, often in the form of recommendation systems [97, 98]. Lee describes the current state of the art of these subsystems [46]. Our template-based architecture gives rise to simplified analogs to these “smart” features, but which still facilitate view exploration goals.

Our *catalog search* is an extensible variation of Tableau’s Show Me feature [54], enabling users to create examples that are available for subsequent recommendation. Like Show Me, catalog search uses data-role based matching to identify potential alternatives. Unlike Show Me, however, catalog search does not internalize domain-specific knowledge of the underlying grammars—the recommendations do not consider, for example, known relationships between human perception and chart configuration.

Our *fan out* enables users to rapidly search across both data and design space by simultaneously juxtaposing parameter selections of interest. Similarly to catalog search, this feature is simple but achieves many of the same goals as additional prior systems [1, 11, 52] that often include sophisticated domain-specific knowledge, by enabling users to rapidly explore alternatives in a low cost manner (i.e. in a simple design gallery [55]) and to manipulate collections of examples simultaneously (à la Juxtapose [26]). This technique most closely resembles René’s [18] combinatorial design exploration, although extended to encompass data-based variations.

2.2.4 Text-Based Editors. There is a long history of UIs that combine textual chart programming with a system for rendering those charts. Computational notebooks, such as Jupyter Notebook [44], facilitate chart construction through tight feedback loops between code and rendered charts. In a similar manner to Ivy templates, papermill [61] allows analysts to parameterize computational notebooks which can then be run in a non-notebook environment. Observable [5] provides a reactive-programming platform for creating

rich web-based visualizations, which users are encouraged to fork and remix. Wood et al. [99] take a literate programming approach to the visualization design process by enabling authors to build Vega and Vega-Lite charts in a Markdown document.

Most closely related to our work is the Vega-IDE [86], which augments textual specification of Vega and Vega-Lite charts with debugging tools [31], and Chart Builder [12], which allows users to edit Vega-Lite charts through a GUI or text, but not both. We borrow much of Vega-IDE interface design in the text-editing portion of our system; for instance, we make use of Microsoft Monaco's JSON-Schema [63] based support for JSON grammars as a way to provide linting and validation. Beyond standard text-editing features, we also implement extensible heuristic rewrite rules that (automatically) suggest abstractions of JSON values into parameters. This eases the flow of converting a declarative specification into a reusable template.

2.2.5 Multimodal Editors. Several prior works have explored combinations of user interface modalities for creating visualizations. Liger [72] mixes together shelf-based chart specification and *visualization by demonstration*. Hanpuku [4], Data-Driven Guides [41], and Data Illustrator [50] combine visual editor-style manipulation with visual chart specification or textual programming. Victor [89] explored a prototype that combined a spreadsheet with direct manipulation and manual view specification. His system enabled highly expressive visualization construction, whereas we focus on supporting analytic tasks.

Tools such as Jupyter Widgets [38] and Shiny [70] provide mechanisms for parameterizing analysis code in an ad hoc manner, however these lack many of the graphical affordances for exploration found in many visual analytics systems.

Systems including mage [40], Wrex [14], and B2 [100] expand on these ideas by intermingling text and graphical specification in computational notebooks. Sketch-n-Sketch [28] takes a bidirectional approach to combining (non data-driven) visual editing with textual programming (in a full-featured, procedural language). Re-Vize [32] seeks to support multiple modalities by chaining together analysis tools through a Vega-Lite-based API.

Our investigation in this paper—to integrate chart choosing, shelf building, and textual specification—is complementary to these efforts. This combination of modalities offers a rich feature-space (such as our *catalog search* and *fan out* features), enables educational opportunities (by presenting a synchronized view between GUI and perhaps unfamiliar textual grammars), and supports a variety of data exploration tasks.

3 MOTIVATING USE CASES

Here we describe how two hypothetical users—Axel, a novice user of visual analytics systems, and Tabitha, a visual analytics power user—carry out two example workflows in Ivy. A motivation for our work is to allow users like Axel and Tabitha to collaborate and benefit from each other's efforts by working within a common system—aiming to supplant single-modality tooling that contributes to *designer-developer breakdowns* [48].

3.1 Axel: A Novice User

Axel wants to make a chart showing death penalty executions in the USA for a report he is writing. After loading a relevant dataset, Axel browses templates that have been created by others. Axel benefits from how other members of his team can easily grow the collection of templates to adapt to the team's changing visualization needs.

Unsure of what values are in the dataset, he selects a univariate bar chart template and uses fan out (Sec. 4.4.2) to view all of the dimension fields simultaneously. As he views these summaries, he gradually removes fan out options that are not interesting or do not serve his task. He eventually selects the option to display "race". He notices that there are 16 related templates (Sec. 4.4.1) that he could use. The "Isotype Waffle" catches his eye. He selects it to view a fully formed Isotype colored by "race". The chart does not have quite the right dimensions so he adds width and height variables to the template, which allows him to make small incremental adjustments from the GUI to produce the final version (Fig. 1f). He takes a screenshot and adds it to his report.

3.2 Tabitha: An Expert User

Tabitha is interested in exploring the Gapminder dataset [69], so she loads the data and selects the IvyPolestar template. She explores the data using the familiar row and column abstractions (as in Fig. 5) to investigate iterative hypotheses. After viewing several scatter plots and bar charts, Tabitha wants to see a hierarchical representation of the part-to-whole relationships between region, country, and GDP; a Sunburst chart comes to mind.

Unable to find a satisfactory chart in the Ivy gallery, she browses the Vega gallery and finds one, but the way that it works with data does not quite match her intended usage. She creates a blank template in Ivy and pastes in the example from the Vega gallery. She is then presented with a series of automatically generated suggestions (Sec. 4.3.3) on how she might template her chart—clicking through these creates new shelves as appropriate. She text-edits the data transformation logic to accommodate her desired functionality. She uses the built-in debugging tools to view the results of the current data state, iteratively developing her transformations. She adds parameters for controlling width, height, color scheme, and other aesthetic values, at which point she is happy with the template and decides to share it. She clicks the publish button to make the template available on the community server, ensuring that she and others can reuse her work in the future. Finally, she instantiates the Sunburst for her dataset and takes a screenshot.

4 SYSTEM DESIGN

We now describe the design and implementation of Ivy. First, in Sec. 4.1, we introduce to the components of our system with a narrative walkthrough. Next, in Sec. 4.2, we present a rigorous formulation of *parameterized declarative templates*—a grammar-agnostic mechanism for abstracting JSON-based specifications—in a small programming language. Then, in Sec. 4.3, we describe our UI design for selecting and instantiating templates, designed to obtain the benefits and familiar aesthetics of chart choosing and shelf building. Finally, in Sec. 4.4, we describe how the systematic application of templates enables beneficial forms of view exploration.



Figure 3: Users can transform example chart specifications into reusable templates. Here we show the process of taking (A) a bar chart of interest and (B) its corresponding Vega-Lite spec, and transforming it into (C) an Ivy template. Using (D) the corresponding GUI shelf builder pane, the user can (E) fan out across parameters of interest to see alternatives simultaneously.

4.1 Narrative Walkthrough

We now introduce the technical components of our system with a narrative description of our hypothetical user Tabitha constructing a reusable template from an example.

She begins by loading a population dataset in Ivy. She then copies in the “Aggregate Bar Chart” [84] from the Vega-Lite documentation (Figure 3b). Figure 3a shows the output visualization.

After pasting in the code into the *template body* (via the Body tab of Fig. 1c), several automated suggestions are provided on how the data fields could be abstracted as *template parameters*. Clicking through the suggestions replaces the “age” and “people” data fields (Figure 3b, lines 8,10) with two new *DataTarget* parameters (Figure 3c, lines 7,13). Tabitha renames the generated parameters to *xDim* and *yDim* which automatically replaces their uses (enclosed by “escape” brackets) with `[xDim]` and `[yDim]` (Figure 3c, lines 23–24). She uses the settings popover (Fig. 6) to specify their allowed data roles—she could have also done so using the Params text box (Fig. 1c). These configurations result in a shelf builder style user interface which she uses to explore her data set. She then uses these shelves to specify values for the *xDim* and *yDim* parameters (Figure 3d). These *value settings* are then *applied* to the *template body* to produce a JSON specification, which is then transformed into a visualization by a language-specific rendering function, in this case, by Vega-Lite.

Next, she wants to chart populations for different years, so she replaces the constant value 2000 (Figure 3b, line 5) with `[year]`

(Figure 3c, line 20), referring to a new template parameter that abstracts over the choice of year (lines 9–12). Intending to apply this template only to datasets with decennial measurements, Tabitha specifies that year should be chosen from among a new Enum type, called “allowedValues”, comprising census years (Figure 3c, line 11). (She chooses not to abstract the filtering predicate, which would have resulted in a both more general and more complex template.)

Then, Tabitha considers whether and how to sort the bars. Knowing the appropriate Vega-Lite option, she adds a “sort” field with a conditional to the template body (Figure 3c, line 23). She then creates a new template parameter, *sort* (line 8), and configures it such that if *sort* is set to true, then the “sort” value in the resulting Vega-Lite specification is set to “-x”, which, in rendered Vega-Lite, sorts the bars in order of increasing value. If *sort* is set to false, then the resulting spec contains no “sort” field—as in the original specification (Figure 3b).

Lastly, Tabitha wants to enable control of the color of the bars in her chart. To do so, she first defines a color parameter (Figure 3c, line 14), and then adds a reference to this parameter on line (Figure 3c, line 21). This allows her, or other users of this template, to pick between colors preferred by her organization.

Her template now satisfactorily prepared, she is ready to explore her dataset by specifying parameter values through the GUI (as in Figure 3d). In addition to specifying individual value settings, she can explore multiple options simultaneously using *fan out* (Figure 3e), which juxtaposes multiple views to consider (Sec. 4.4.2).

Templates	$t ::= \begin{cases} \text{func} = \lambda(x_1:T_1, \dots). e \\ \text{lang} = \mathcal{L}, \text{metadata} = \dots, \\ \text{symbols} = [y_1, \dots], \end{cases}$	Template Application	$t(S) = v$
Atomic Values	$a ::= s \mid n \mid b$	$t.\text{lang} = \mathcal{L} \quad t.\text{func} = \lambda(x_1:T_1, \dots). e \quad S = x_1 \mapsto a_1, \dots$	
JSON Values	$j ::= a \mid \{s_1:j_1, \dots\} \mid [j_1, \dots]$	A. Substitute Arguments $Se = e'$	B. Evaluate Conditionals $\llbracket e' \rrbracket_{\text{Ivy}} = j$
Expressions	$e ::= a \mid \{s_1:e_1, \dots\} \mid [e_1, \dots]$		C. Render Visualization $\llbracket j \rrbracket_{\mathcal{L}} = v$
Variables	$ x \mid y$	Evaluation of JSON Expressions	$\llbracket e \rrbracket_{\text{Ivy}} = j$
Conditionals	$ \text{if } p \text{ then } e_1 \text{ (else } e_2)$	Atomics	$\llbracket a \rrbracket_{\text{Ivy}} = a \quad (1)$
Predicates	$p ::= s \text{ (where } \llbracket s \rrbracket_{\text{JS}} = b)$	Objects	$\llbracket \{s_1:e_1, \dots, s_n:e_n\} \rrbracket_{\text{Ivy}} = \cup_i \llbracket (s_i:\llbracket e_i \rrbracket_{\text{Ivy}}) \rrbracket_{\text{Ivy}} \quad (2)$
Settings	$S ::= x_1 \mapsto a_1, \dots$	Lists	$\llbracket [e_1, \dots, e_n] \rrbracket_{\text{Ivy}} = [\llbracket e_1 \rrbracket_{\text{Ivy}}, \dots, \llbracket e_n \rrbracket_{\text{Ivy}}] \quad (3)$
Data Roles	$R ::= \text{Measure} \bullet \mid \text{Dimension} \bullet \mid \text{Time} \bullet$	Evaluation of Conditionals	$\llbracket e \rrbracket_{\text{Ivy}} = j \text{ or } \perp$
Param. Types	$T ::= \text{DataTarget } args \mid \text{MultiDataTarget } args \mid \text{String } args \mid \text{Number } args \mid \text{Boolean } args \mid \text{Enum } args \mid \text{Text } args \mid \text{Section } args$	$\llbracket \text{if } p \text{ then } e_1 \text{ (else } e_2) \rrbracket_{\text{Ivy}} = \llbracket e_1 \rrbracket_{\text{Ivy}} \quad \text{if } \llbracket p \rrbracket_{\text{JS}} = \text{true} \quad (4)$	$\llbracket \text{if } p \text{ (then } e_1) \text{ else } e_2 \rrbracket_{\text{Ivy}} = \llbracket e_2 \rrbracket_{\text{Ivy}} \quad \text{if } \llbracket p \rrbracket_{\text{JS}} = \text{false} \quad (5)$
Spec. Lang.	$\mathcal{L} ::= \text{Vega} \mid \text{Vega-Lite} \mid \text{Atom} \mid \text{Table} \mid \dots$	$\llbracket \text{if } p \text{ then } e_1 \rrbracket_{\text{Ivy}} = \perp \quad \text{if } \llbracket p \rrbracket_{\text{JS}} = \text{false} \quad (6)$	
Views	$v ::= (\mathcal{L}\text{-language specific rendering})$	Evaluation of Conditional Fields	$\llbracket (s:e) \rrbracket_{\text{Ivy}} = \{s:e'\} \text{ or } \emptyset$
		$\llbracket (s:e) \rrbracket_{\text{Ivy}} = \emptyset \quad \text{if } \llbracket e \rrbracket_{\text{Ivy}} = \perp \quad (7)$	$\llbracket (s:e) \rrbracket_{\text{Ivy}} = \{s:\llbracket e \rrbracket_{\text{Ivy}}\} \quad \text{otherwise} \quad (8)$

Figure 4: The Ivy template language is composed of an abstract syntax grammar (left) and evaluation rules (right).

4.2 Template Language Design

Templates provide a simple set of abstractions over JSON-based grammars. Put simply, a template is a function specified in a superset of JSON, which includes variables and simple control flow operators, that when applied to arguments produces a chart in a particular visualization grammar. Templates are grammar-agnostic as they abstract arbitrary JSON specifications. We provide a full description of templates to highlight exactly how they make declarative grammar specifications reusable (G4)—by combining multiple specifications into a single template—and easier to use (G1)—by demarcating the arguments for manipulation.

Formally, we define a *template* t as a function that applies a set of N **template parameters**, x_i with type T_i , to a **template body** e to generate a *view* v . In addition to the *function* itself, a template is a record that defines: the output JSON specification *language* \mathcal{L} of the template body, *metadata*, and a list of zero or more template-specific constant *symbols* that the body may refer to (Sec. 4.2.2). Our simplified model of JSON [15] values j comprises *atomic values* a —literal strings s of type `String`, numbers n of type `Number`, and booleans b of type `Boolean`—records of string-value pairs, and lists of values. The abstract syntax of templates is defined in Fig. 4.

4.2.1 Template Bodies. Beyond JSON literals, a template body, or *expression* e , may employ two basic programming constructs. A *variable* refers to a template function parameter x or a template symbol y , wherever a JSON literal value might normally appear. A *conditional expression*, written $\text{if } p \text{ then } e_1 \text{ (else } e_2)$, is dependent on the result of *predicate expression* p , which is a “raw” JavaScript

code string that evaluates to a boolean value. The else-branch is optional, which supports optionally-defined fields, per Sec. 4.2.3.

Fig. 3c shows the concrete syntax for variables and conditionals in Ivy, which implements the abstract syntax of Fig. 4. Variables are “escaped” with square brackets (e.g., “[year]”). Following similar support in other tools [36], conditionals are written as shown in Fig. 3c line 23. Full-featured languages typically provide more extensive abstraction mechanisms, but—as we show in Sec. 5—even just variables and conditionals suffice for a wide variety of use cases.

4.2.2 Template Parameters. The N parameter declarations x_i with types T_i serve two purposes: to abstract over data fields and stylistic choices in the definition of a visualization and to define GUI elements that allow users to specify argument values for these parameters. For example, a “switch” widget is drawn for each parameter of type `Boolean`, a dropdown menu for each `Enum`, and a Tableau-style “shelf” for each `DataTarget` (Fig. 3d). Fig. 4 defines several parameter types T . The data target types `DataTarget` and `MultiDataTarget` range over data columns (identified with strings) in the current dataset as well as template-specific symbols (also encoded with strings in our implementation). Symbols are template-specific values that can be used to instantiate parameters. For example, IvyPolestar defines a count symbol that induces a targeted channel to take on a count aggregation.

Each type T_i contains type-specific arguments $args_i$, for example, the minimum and maximum allowed values for a `Number` or the values comprising an `Enum`. The data parameter types—`DataTarget` and `MultiDataTarget`—carry a configuration field to define a *data role* R , discussed further in Sec. 4.3. Each type T_i also contains an

optional boolean expression p_i that determines whether the GUI should display controls for that parameter x_i —for example, displaying a sort direction widget only when a boolean `sort` parameter is set to true. Fig. 4 elides details of these type-specific arguments.

As parameters are arguments to a function, they must also have values. We refer to a user’s choice of argument values a_i as *settings* S . Settings are typically set using the GUI, but can also be specified as text (Fig. 1c, “Settings” tab).

4.2.3 Evaluating Templates to Visualizations. Finally, we produce visualizations by applying the function described by template t to the parameter settings S , following the three steps found in Fig. 4 A–C. **First** (Fig. 4A), the argument values a_i are substituted for the parameters x_i in the template body e using straightforward substitution. For example, this transforms a snippet of the body in Fig. 3c, `{"y": {"field": "[yDim]"}, ...}` with settings `{"yDim": "age", ...}` to be `{"y": {"field": "age", ...}}`.

Second (Fig. 4B), the resulting expression is evaluated to produce a JSON value, transforming any conditionals into JSON values. This is done by a straightforward recursive traversal, following Eqns (1)–(3). Whenever a conditional is encountered, it is executed by evaluating the JS-based predicate ($\llbracket \cdot \rrbracket_{JS}$ referring to JS evaluation), and then replacing the conditional with either the corresponding then- or else-branch as appropriate. Eqns (4) and (5) handle the two usual cases for conditionals. Eqn (6) handles the case when the predicate evaluates to false but no else-branch is provided; returning a \perp value, indicating that the conditional is to be deleted. Eqns (7) and (8) explicitly provide a mechanism to delete conditionals that return \perp in record values. This is how, for example, Fig. 3c line 23 optionally defines a “sort” field. Our implementation generalizes this rule to arrays by viewing them as numerically indexed records.

The **third** (Fig. 4C) and final step is to use the generated JSON to render a visualization. This is done by, for a specific visualization language \mathcal{L} , using the corresponding rendering function $\llbracket \cdot \rrbracket_{\mathcal{L}}$ to interpret the JSON value and render the resulting view—typically as an HTML Canvas or SVG. Our current implementation supports four languages \mathcal{L} , namely, Vega, Vega-Lite, Park et al.’s Atom [62], and a simple table language. However, as we discuss in Sec. 6.1.1, Ivy’s language support is designed to be extensible.

4.3 User Interface Design

Equipped with the notion of templates, we next describe the user interface design of Ivy. As shown in Fig. 1, the application consists of two panes, one for chart editing and another for chart viewing. The chart editing pane contains a data column filled with Tableau-style “pills” representing data columns, and an encoding column with “shelves” for those pills to be placed upon. This encoding column can be used to instantiate (i.e. provide arguments for) the parameters of the template, or to edit the GUI of the current template. The editing pane also includes a code editor which can manipulate the current template and UI state textually. Per Saket et al. [72], we facilitate multimodal interaction by tightly synchronizing these views, such that changes in one modality are instantly reflected in the other.

Below, we describe how Ivy supports the *creation*, *selection*, and *application* of templates to produce charts.

4.3.1 Template Selection as Chart Choosing. The root of Ivy is a template gallery, which aims to achieve Satyanarayan et al.’s [78] vision for a system “allowing users to browse through designs for inspiration, or adapt them for their own visualizations.” This avoids the blank canvas problem [75] and supports ease of use (G1).

The gallery is populated with a library of system-provided templates, as well as templates created by Ivy users (stored on a communal server). Simpler templates allow users to jump quickly to familiar visual forms (such as line charts or bar charts), while more sophisticated templates privilege thinking with their data [68] (as in IvyPolestar). The gallery is present both as a homepage for the application, independent from the visualization editor, as well as an intermediate view while creating new view tabs.

Each template is accompanied by a set of user-defined examples, namely, settings chosen by users to instantiate the template, with data bindings and output renderings with respect to a collection of predefined datasets. These examples serve as “crowd-sourced documentation” for how individual templates operate. Furthermore, this adds an element of opportunistic programming: to create templates, users can borrow small snippets—such as a well-formatted list of color schemes—and use them in their own creations.

4.3.2 Template Application via Shelf Building. After selecting a template and uploading a dataset of interest, the user is presented with a shelf builder-style GUI for setting the template parameters and specifying basic data filters. We choose this GUI design seeking to exploit the same affordances that drive the explorability (G2) of shelf builders. Specifically, our design closely follows that of the Polestar shelf builder system, which Wongsuphasawat et al. [97, 98] constructed as a simulacra of Tableau to serve as a baseline comparison in the development of their recommendation-based exploration systems. While emulating Polestar is a relatively small threshold to overcome in the context of visualization systems in general, it demonstrates the promise of our template-based approach. Systems such as Tableau or PowerBI possess features that—although substantially larger and more complex—are not substantially *different* than those in Polestar.

To select data parameters (i.e. `DataTable` or `MultiDataTable`) of interest, users drag-and-drop from a list of data fields, color-coded according to their *data roles*, onto encoding shelves, as in Fig. 1a, d. Following prior work [1, 80, 97] roles include **Measure** (quantitative fields), **Dimension** (nominal or ordinal fields), and **Time** (temporal fields). When a dataset is loaded, we make heuristic guesses about the role for each column, which the user can later modify. We use roles in Ivy to construct a naive automatic *Add to Shelf* feature (akin to Tableau’s *Add to Sheet* [54]), except ours is simply based on order and data role. If a template has three `DataTargets`, the first of which allows only a **Measure** while the latter two allow anything, clicking *Add to Shelf* on a **Dimension** will add it to the second parameter.

In addition to the visual aesthetics of Polestar (and hence that of Tableau), we also emulate the *functionality* of its shelf-building interface through a “default” library template called IvyPolestar (shown in Fig. 5). The only features not replicated are the Automatic Mark Type—implementation of which, though possible in Ivy, was beyond the scope of the paper—and the chart bookmarks—which we replaced with a notion of view tabs.

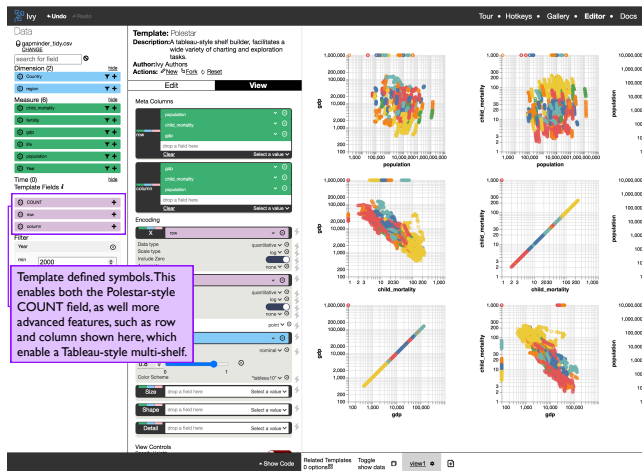


Figure 5: Our IvyPolestar template reproduces and extends the functionality of the Polestar shelf builder. Here template specific symbols are used to make a SPLOM.



Figure 6: In edit mode, users can modify templates both through the shelf builder GUI and the textual code pane.

A notable feature in Polestar is a *count* symbol that is visually similar to normal data fields and induces count aggregations on channels without a selected data column. To model this feature, we define a template-specific symbol (cf. [Sec. 4.2.2](#)), *count*, which IvyPolestar uses to implement the corresponding functionality. IvyPolestar introduces two additional symbols, *row* and *column*, that enable faceting by data columns in the manner of Tableau’s multi-shelves. Placing *row* or *column* symbols on any shelf creates a *MultiDataTarget* which acts as a wrapper around Vega-Lite’s juxtaposition operators [76]. [Fig. 5](#) highlights this feature.

4.3.3 Template Creation and Text Editing. Templates can be created or modified in two ways, either by modifying the textual representation (Fig. 3c) or through GUI interactions (Fig. 6). The textual representation facilitates both small tweaks (Sec. 3.1), as well as creating new templates. For instance, users may copy code snippets found online—such as in language documentation or Stack Overflow—and templateize them to suit their task (Sec. 3.2). Templates can also

be created by “freezing” and refining the GUI state when interacting with an existing template. For example, a user might apply a full-featured template, such as IvyPolestar, to construct something resembling their desired chart, fork the text output as a new template (as in Fig. 1d), and then provide fine textual grained updates. As discussed previously, exposing textual representation to the end user furthers our flexibility (G3) and reusability (G4) goals.

To ease the construction of templates, Ivy uses domain-specific pattern matching and rewrite rules to suggest potential transformations to users. For instance, if a user were to find a chart in the Vega documentation that they wanted to copy, they would simply start a new template and paste the code into Ivy. The code pane then suggests ways to transform the code. For example, if a value in a Vega-Lite spec (such as in [Fig. 3b](#)) is used where a data reference is expected (e.g. `"field": "age"`), then Ivy suggests swapping `"age"` with a reference to a new parameter. [Fig. 6](#) shows an example of such rules. Rules are defined by Ivy developers, rather than Ivy users. We implemented rewrite rules for Vega-Lite, Vega, and Atom.

4.4 Template-Based View Search

The systematic formulation and application of templates allows us to emulate recommendation and exploration (G2) features found in a variety of existing charting systems as a consequence of our design. Here we highlight two such features that follow naturally from the use of templates: one arises by fixing the arguments and varying the template, and the other by fixing the choice of template and varying the arguments.

4.4.1 Extensible Recommendation with Catalog Search. The heterogeneity of user needs is often addressed in chart choosers by offering large and often diverse sets of charting options [54], which can be intimidating or difficult to utilize due to their volume.

The gallery in Ivy is equipped with *catalog search*, which allows users to search across the set of available templates based on compatibility with a set of specified columns of interest—specifically, by using a simple type-compatibility algorithm that compares template parameter types with the data roles of selected columns. This feature exists both in the gallery—where it acts as a type-based search mechanism—as well as ambiently throughout the system in the related templates tab (Fig. 1f)—where it acts as a simple alternative recommendation system. The current template is compared with each other template, yielding a *partial match*, a *complete match*, or *no match*. A partial match occurs when the selected data columns can be mapped to a template’s parameters. A match is complete if all required parameters are mapped. A complete match can translate the current selection without additional specification. This heuristic is further detailed in the appendix. When a match is selected, the current columns are mapped onto the new template, using a similar mechanism as our *Add to Shelf*, and the resulting chart is shown immediately.

4.4.2 Exploring Encoding Variation through Fan Outs. Comparisons in visual analytics are often made temporally, requiring the analyst to hold mental reference to each of the values under consideration. To reduce this cognitive burden, Ivy users can *fan out* a template by applying multiple settings and rendering their output simultaneously. Fig. 3e and Fig. 7 display examples of this interaction.

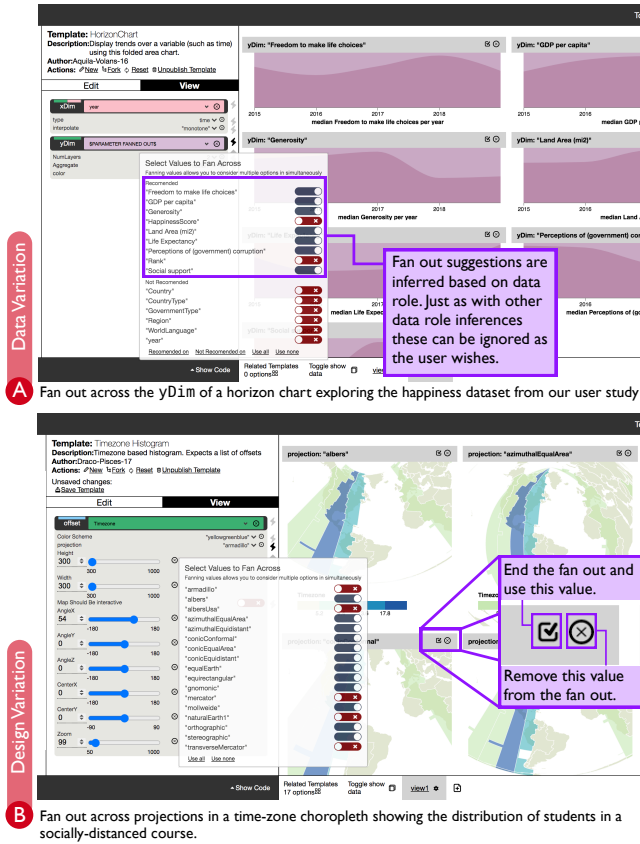


Figure 7: Users rapidly explore design and data alternatives via fan outs. After selecting parameter values of interest, they are shown all variations simultaneously.

To begin a fan out a user specifies sets of values that they wish to compare for each parameter of concern. This can be applied to both design and data parameters. For data parameters we provide suggestions of appropriate values based on the inferred data role and specified parameter role (Fig. 7a). We then compute a cartesian product of these sets and render a separate instance of the current template for each combination of values. Users then browse the resulting gallery, and can modify all of the combinations at once through the shelf-building UI, as well as remove values or select a combination to view in isolation (thus collapsing the fan out).

This approach allows users a low-stakes way to consider alternative chart configurations and rapidly explore the space of available design and data parameters.

5 EVALUATION

In the previous section, we discussed how our system design integrates and augments UI capabilities provided by existing chart choosers, shelf builders, and text-based editors. Here, we assess how well our template-based approach may work in practice by considering two questions: *Do templates facilitate organization and reuse of existing visualizations?* (Sec. 5.1), and *Is Ivy’s multi-modal UI approachable by real users?* (Sec. 5.2)

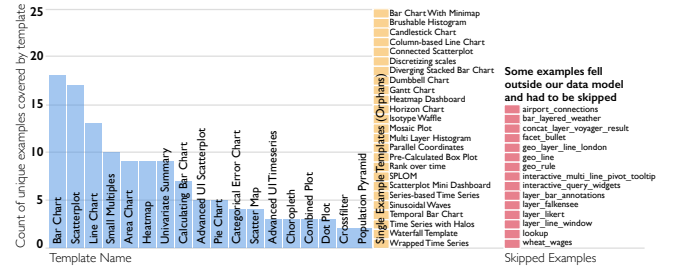


Figure 8: We reproduced the 166 examples found in the Vega-Lite gallery using 43 templates. Some examples had to be excluded (right) as they fell outside our data model, usually because they utilize multiple data sources, which Ivy does not currently support.

5.1 Templates for Existing Visualizations

We considered two chart corpora, the example gallery of Vega-Lite [88] and the chart chooser found in Google Sheets [19], looking for opportunities to factor related visualizations into templates, which yielded 3.5x and 1.8x *compression ratios*, respectively. Compression is the number of examples constructible by a given template. The reported average compression are computed by

$$\frac{|\text{Examples}| - |\text{Excluded by Data Model}|}{|\text{Templates}|} \quad (9)$$

In an example gallery, a larger ratio means less duplicated code (text) among examples. In a chart chooser, it means fewer chart forms with possibly more parameters. These results demonstrate that the simple template abstraction mechanisms enhance the flexibility (G3) of existing declarative grammars while improving their reusability (G4) by serving a variety of use cases.

While even just the basic abstraction mechanisms in templates—namely, variables and conditionals—are sufficient for merging all examples of each corpus into single templates, we strove to construct templates that are factored in reasonable ways, for example, by not depending on datasets having particular columns or features, and by considering what might plausibly be found in a chart chooser. This process prompted a number of minor system improvements, such as simplifying the conditional syntax and improving error handling, and suggested directions for future work on Ivy, such as a more sophisticated data model. Additional details of this analysis can be found in the appendix.

5.1.1 Vega-Lite Example Gallery. This gallery consists of 166 distinct specifications, reflecting years of iterative refinement and development by the Vega-Lite community. These examples highlight a breadth and depth of features exposed by the library, thus forming an ideal testbed for the utility of templates. We recreated this corpus with 43 templates, skipping 14 examples due to incompatibility with our data model (a single, flat input data table) for a 3.5x compression. Fig. 8 reports the frequency of each template.

We aimed to capture both the core content of each example (the major feature being demonstrated) as well as the resulting image. However, this was not always possible. We allowed minor text modifications and the inclusion of properties not present in the original example, as long as they did not affect the core of the example

(such as styling). It was sometimes necessary to forgo or modify some examples to accommodate our current implementation. For instance, a scatter-map of zip-code centroids colored by their first digit [87] needed to be modified because Ivy currently lacks a custom calculation feature. As a way to guide this design, we strove to maximize the *concatenation ratio*, or the ratio between the size of a template body and that of the concatenated examples it captures, which is computed for a particular measure of size γ ,

$$\frac{\sum_{x \in \text{covered examples}} \gamma(x)}{\gamma(\text{template})} \quad (10)$$

The resulting templates have an average lines of code concatenation ratio of 1.48x (a proxy for simplification) and an average abstract syntax tree concatenation ratio of 1.80x (a proxy for UI complexity minimization). In conjunction with the average example compression ratio of 3.5x, this suggests that our templatisations are better than merely concatenating the examples.

5.1.2 Google Sheets Chart Chooser. The set of charts found in this chooser consists of 32 distinct chart forms. We focused on Google Sheets as a representative chart chooser because it possesses a similar set of chart options as other choosers, such as Excel and LibreOffice. We reproduced 29 of these charts through 16 templates for a 1.8x compression factor. We skipped 3 examples for the following reasons: “3D Pie” charts require a 3D visualization grammar, “Org. Charts” fall outside of our tabular data model, and “Timelines” require annotations, and therein modifications to the underlying dataset, which is not currently supported in Ivy.

5.2 Approachability Study

We conducted a study in order to evaluate whether the multimodal user interface in Ivy is approachable by real users. During pre-study pilot sessions, we found that users who were more familiar with Vega-Lite learned more quickly how to integrate Ivy’s full capabilities to address tasks than those unfamiliar with Vega-Lite. Therefore, we aimed to recruit users familiar with these paradigms in order to evaluate the approachability of interacting with multiple charting modalities, rather than that of the underlying grammars. The target expertise for our study can be characterized roughly as between that of the hypothetical users from Sec. 3.

5.2.1 Participants. We solicited 5 participants from a recent visualization course in a computational public policy masters program, all of whom are now employed as professional data analysts. Based on their participation in the course, we were confident that these now-graduated students were familiar with Vega-Lite and Tableau. Participants, denoted P1 through P5, self-reported a mean familiarity with Vega/Vega-Lite of $\mu = 2.4$ on 5-point Likert scale, and $\mu = 3.2$ with visual analytics tools (5 indicating high familiarity). Participants were students at the same institution as the experimenters—specifically, in the context of a student-instructor relationship—thus constituting a source of potential bias.

5.2.2 Methods. Following a brief introduction to Ivy through in-application documentation, participants completed a series of 8 tasks that covered data exploration, chart specification, and template construction problems (additional details in the appendix). For instance, one task asked participants to templatize a particular

Vega-Lite-based box plot [85], while another asked them to find the number of countries in a multi-year version of the World Happiness Report [27]. We focused on these tasks because they are similar to tasks supported in related systems and (with the exception of template building) were addressable with any of the modalities individually or in conjunction. Correctness was judged by comparison with a solution set prepared prior to the study. We engaged subjects in an informal think-aloud protocol during the session, which led to a number of the reported observations in the next section. Subjects were then asked to fill out an exit survey on the usability of Ivy and templates. Sessions were held over video conference software and lasted a mean of $\mu = 95$ minutes, although 2 hours were allotted. Participants received a \$50 Amazon gift certificate for their work.

5.2.3 Results. All participants were able to complete all tasks within the allotted time, although all users required some assistance at various stages, typically due to implementation bugs or learnability hurdles. A variety of strategies were used to accomplish the data exploration tasks, some using only shelf building, some only chart choosing, or a mixture of both. All users were able to complete the template construction tasks and then use the templates to address data exploration tasks. We believe this indicates that real users, once acclimated, are able to produce non-trivial templates to accomplish varying goals.

This system was generally seen as usable. Participants mostly agreed with the statement “I think that I would like to use this system frequently” ($\mu = 4.4$ on a 5-point Likert scale), and gave a mean system usability score of $\mu = 68.0$ —describable as being between “OK” and “Good” [3]. More critical than users’ perception of the usability, which may have been positively biased, is the demonstration they were able to navigate the system and use the interlocking modalities to achieve various tasks. While our study did not cover some concepts in our system—such as conditionals and catalog search—it demonstrated that users can approach and utilize the various UI modalities in Ivy. Participants sometimes had to be directed to use certain unfamiliar features, such as the templatization suggestions and fan outs. However, once familiar, users tended to continue to use those features.

Participants were enthusiastic about mixing code and graphical specification. P5 commented that the combination “*feels more useful than just coding*”. P4 noted that their organization had recently switched a major Tableau-based dashboard to a Shiny-based dashboard because of a lack of precise data controls in Tableau. In contrast to the push for visual analytics tools to completely shed ties to text (embracing a “no-code” approach), we believe this suggests that systems that straddle the boundary between code and GUI specification can offer a valuable mixture of affordances that support real use cases.

5.2.4 Limitations. We observed that Ivy held some challenges for participants. Participants sometimes struggled to understand what the templatization suggestions would do, indicating a closeness of mapping [22] failure. P1 and P2 suggested that visual design could be improved to aid in feature discovery. While some participants (P2, P4) agreed that templates are good for creating rapidly reusable charts, P2 noted that templates are “*not very portable*.” This could be addressed by embedding Ivy in tools like Shiny [70] or Jupyter [38].

The learnability of the system and mixed-modality remained a primary difficulty. Users typically struggled to figure out how to bridge the gap between text and GUI at first, however, by the end of the session, all users were competent in both regimes. For instance, most subjects iteratively refined their solutions to the box plot task, modifying both code and GUI values to address developing hypotheses. P3 noted that the system required a non-trivial level of computational and visualization literacy. Fortunately, users could seemingly bootstrap their knowledge to overcome these hurdles. P4 noted that the integration between the “code body and the point-and-clickable GUI is really tight and also good for reinforcing learning” and that “If you know how to do something in one form, you can do it and watch how it changes the other side of the tool.”

The scope of this study was small. We merely sought to demonstrate that real users of similar systems could approach the mixed-modality UI found in Ivy. While these results suggest that this combination is promising, further investigation is required to understand its utility in the context of more developed interfaces. Once our system reaches maturity, we intend to conduct a study comparing it with standard analytics tools, such as Tableau or Excel.

6 DISCUSSION

In this paper, we described how *parameterized declarative templates*—a typed abstraction layer over JSON specifications—can serve as a basis for a multimodal UI to create and explore visualizations. Ivy-style templates may help in the organization and reuse (G4) of existing visualization corpora (per Sec. 5.1). Vega and Vega-Lite have garnered ample popularity, and new declarative visualization grammars are being actively developed [42, 49, 96]. As the availability and use of these grammars continues to proliferate, there is opportunity for shared platforms and tooling between languages, which we explore in our grammar-agnostic templates.

The integration of features in our prototype appears to be accessible to users with modest experience in both visual analytics systems and Vega/Vega-Lite (per Sec. 5.2). Users were able to make effective use of affordances for exploration found in our shelf building UI and fan out (G2), and were able to utilize the capability of templates to improve the ease of use (G1) and reuse (G4) of declarative chart specifications while maintaining their flexibility (G3).

We believe that this multimodal approach has value for a variety of use cases. Exposing a connection between GUI and programmatic API may enable analysts to self-serve their chart creation needs. If a particular chart form is not available (but is constructible by one of the supported grammars) then they can create it for themselves, rather than requiring reliance on engineering resources. This connection between text and GUI appears to help users learn and comprehend JSON-based charting grammars, which may be unfamiliar or difficult to understand. The repeatable customization found in templates might also, for example, enable practitioners (e.g. data journalists) to explore designs in a structured manner that does not violate their organization’s visual identity.

6.1 Limitations and Future Work

The version of multimodal visual analytics found in our prototype has its share of limitations. The strength of each modality in Ivy is only as good as its implementation, which can render artificial

barriers between what users expect and what is supported (e.g. P3 expected a pivot table). And while Ivy encompasses chart choosing, shelf building, and textual specification, it does so at the cost of an increased learning curve. However, we believe that this difficulty is not endemic to multimodal systems, and that through attentive design the experience of using the system can be made easier.

In making our template-based approach more viable for practical use, it is easy to imagine a variety of system improvements—such as additional template parameter types (e.g. color schemes or inline data fields), drag-and-drop interactions for refactoring and abstracting specifications (e.g. [29, 47]), as well as enriching the ways in which changes made in one modality are reflected and explained in the others. Beyond these, we highlight below several avenues for future research.

6.1.1 Language Extensibility. Ivy is designed to be extensible: support for each specification language is defined through a standardized interface, which includes a JSON Schema describing the syntax, a JavaScript rendering function for the language, and rewrite rules to help users abstract specifications. Our implementation currently supports a small set of languages (Vega, Vega-Lite, Atom¹, and a simple table language), which, in future work, we would like to increase so as to support a greater variety of tasks.

Our grammar-agnostic template framework provides a standard set of abstraction and data manipulation mechanisms—which may reduce the need for grammar designers to define their own—and novel UI features for exploring candidate templates (catalog search) and encodings (fan outs)—which may facilitate more efficient and consistent exploration. Our system, furthermore, hoists the burden of data transformation out of the rendering grammar (albeit with a currently-limited set of transforms), which would otherwise require each grammar to implement its own data manipulation logic. As users move between templates (specified in possibly different grammars), their settings (including filters) are mapped from one template to the next via role and order-based heuristics. Future work could enable translation between supported grammars, which could yield opportunities for education and portability.

Despite the benefits of language-independent functionality, there are also benefits to taking domain-specific knowledge into account. Language-specific rewrite rules—part of the extension interface, described above—are one such example. Language-specific knowledge could further be used for recommendation, as well as data manipulation and presentation concerns. For instance, when a data field is dragged to a drop zone in Lyra [74] the appropriate type of scale is automatically inferred [75]. Lyra is able to offer this functionality because it has a model of the grammar being manipulated, a functionality which our approach currently lacks.

A lack of context and content-aware automated guidance is a key limitation of our design. Yet, it should be possible to identify a richer extensibility API, while still allowing each language to benefit from the abstraction and UI concerns shared by all. Such an API would enable us to combine domain-specific chart recommendation (Sec. 2.2.3) with Ivy’s domain-independent type-based exploration (Sec. 4.4), as well as embrace new interaction modalities.

¹While implementing support for Atom we extracted the language in the original application into a standalone library: <https://www.npmjs.com/package/unit-vis>

6.1.2 Validation in Visual Analytics. An important question for system designers is how to help users conduct safe visual analysis [101]. Analysts can deceive themselves with statistical traps [65], visualization hallucinations [43], or false graphical inferences [93].

We believe that our type-based template search will dovetail with a metamorphic testing [58] based validation approach: by varying the parameters of a template and comparing the resulting images, a validation system could automatically identify errors at the intersection of data and encoding. Similarly, we suggest that templates likely offer an opportune medium for applying visualization linting [33, 57] to a visual analytics context, as the types expose the specific arguments over which analysis could be conducted. Furthermore, the fan out interaction could be extended to allow not only juxtaposition of variants, but also their layering [51], enabling visual sanity checks to the robustness of parameter selections [10].

6.1.3 Integrated Visualization Editors. A primary focus in this paper has been supporting data exploration tasks, but there are a constellation of other tasks that users perform in visualization tools. Users must carry out tasks “before” exploration (such as model building and data cleaning) and “after” exploration (such as annotation and presentation). Although the designs of many visual analytics systems assume that data has been cleaned and processed prior to analysis [75], in practice these tasks are often interleaved and iterated.

Integrating these tasks within a unified system may thus reap potential benefits. Data manipulation tasks, for instance, might be better facilitated by the incorporation of ideas from dataflow programming and spreadsheets, as well as *programming-by-example* techniques to help with data wrangling [16, 37, 39]. To more fully support presentation tasks, it would be fruitful to extend our combination of modalities to include *visual builders* [20]—which offer a variety of direct manipulation features [71, 79] for creating charts—and *visualization by demonstration* [72, 73, 91, 102].

For each of these tasks, developing rich graphical interactions—while maintaining a “bidirectional” connection to the underlying textual representation—is an exciting research challenge: to integrate what is typically a vast divide between text and GUI based analytics systems. The approach pursued in this paper provides a step in this longer-term direction.

7 ACKNOWLEDGMENTS

We thank our anonymous reviewers for their helpful comments, as well as our study participants for their participation and invaluable insights. We also thank Will Brackenbury, Michael Correll, Galen Harrison, Brian Hempel, Gordon Kindlmann, and Katy Koenig for their commentary and support.

REFERENCES

- [1] Mallika Agarwal, Arjun Srinivasan, and John T. Stasko. 2019. VisWall: Visual Data Exploration Using Direct Combination on Large Touch Displays. In *30th IEEE Visualization Conference, IEEE VIS 2019 - Short Papers, Vancouver, BC, Canada, October 20–25, 2019*. IEEE, 26–30. <https://doi.org/10.1109/VISUAL.2019.8933673>
- [2] Christopher Ahlberg. 1996. Spotfire: an Information Exploration Environment. *ACM SIGMOD Record* 25, 4 (1996), 25–29.
- [3] Aaron Bangor, Philip Kortum, and James Miller. 2009. Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies* 4, 3 (2009), 114–123.
- [4] Alex Bigelow, Steven Drucker, Danyel Fisher, and Miriah Meyer. 2016. Iterating Between Tools to Create and Edit Visualizations. *IEEE Transactions on Visualization and Computer Graphics* (2016). <https://doi.org/10.1109/TVCG.2016.2598609>
- [5] Mike Bostock. 2018. A Better Way to Code. <https://medium.com/@mbostock/a-better-way-to-code-2b1d2876a3a0>
- [6] Michael Bostock and Jeffrey Heer. 2009. Protovis: A Graphical Toolkit for Visualization. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (2009), 1121–1128. <https://doi.org/10.1109/TVCG.2009.174>
- [7] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (2011), 2301–2309. <https://doi.org/10.1109/TVCG.2011.185>
- [8] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. 2009. Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1589–1598. <https://doi.org/10.1145/1518701.1518944>
- [9] Seth J. Chandler. 2018. Executions in the United States. https://datarepository.wolframcloud.com/resources/Seth-J.-Chandler_Executions-in-the-United-States Wolfram Data Repository.
- [10] Michael Correll, Mingwei Li, Gordon Kindlmann, and Carlos Scheidegger. 2018. Looks Good to Me: Visualizations as Sanity Checks. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2018), 830–839. <https://doi.org/10.1109/TVCG.2018.2864907>
- [11] datavized. 2018. Morph. <https://morph.graphics/>
- [12] data.world. 2020. Chart Builder. <https://github.com/datadotworld/chart-builder>
- [13] Datawrapper. 2020. Datawrapper. <https://www.datawrapper.de/index.html>
- [14] 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 *CHI '20: CHI Conference on Human Factors in Computing Systems*. ACM, 1–12. <https://doi.org/10.1145/3313831.3376442>
- [15] ECMA. 2017. *The JSON Data Interchange Syntax, ECMA-404* (2nd ed. ed.). <https://www.json.org/json-en.html>
- [16] Sara Evensen, Chang Ge, and Catagay Demiralp. 2020. Ruler: Data Programming by Demonstration for Document Labeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*. 1996–2005.
- [17] Flourish. 2020. Flourish | Data Visualization & Storytelling. <https://flourish.studio/>
- [18] Jon Gold. 2016. Declarative Design Tools. <https://jon.gold/2016/06/declarative-design-tools/> Accessed December 16, 2020.
- [19] Google. 2020. Types of charts & graphs in Google Sheets. <https://support.google.com/docs/answer/190718> Accessed August 19, 2020.
- [20] Lars Grammel, Chris Bennett, Melanie Tory, and Margaret-Anne D. Storey. 2013. A Survey of Visualization Construction User Interfaces. In *EuroVis (Short Papers)*. <https://doi.org/10.2312/PE.EuroVisShort.EuroVisShort2013.019-023>
- [21] Lars Grammel, Melanie Tory, and Margaret-Anne Storey. 2010. How Information Visualization Novices Construct Visualizations. *IEEE Transactions on Visualization and Computer Graphics* (2010). <https://doi.org/10.1109/TVCG.2010.164>
- [22] Thomas RG Green. 1989. Cognitive dimensions of notations. *People and computers V* (1989), 443–460.
- [23] Li Guozheng, Min Tian, Qinmei Xu, Michael McGuffin, and Xiaoru Yuan. 2020. GoTree: A Grammar of Tree Visualizations. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (2020). <https://doi.org/10.1145/3313831.3376297>
- [24] Pat Hanrahan. 2006. VizQL: A Language for Query, Analysis and Visualization. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, 721–721.
- [25] Jonathan Harper and Maneesh 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>
- [26] Björn Hartmann, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R Klemmer. 2008. Design as Exploration: Creating Interface Alternatives Through Parallel Authoring and Runtime Tuning. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*. 91–100. <https://doi.org/10.1145/1449715.1449732>
- [27] John F Helliwell, Richard Layard, and Jeffrey Sachs. 2012. World happiness report. (2012).
- [28] Brian Hempel, Justin Lubin, and Ravi Chugh. 2019. Sketch-n-Sketch: Output-Directed Programming for SVG. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 281–292. <https://doi.org/10.1145/3332165.3347925>
- [29] Brian Hempel, Justin Lubin, Grace Lu, and Ravi Chugh. 2018. Deuce: A Lightweight User Interface for Structured Editing. In *International Conference on Software Engineering (ICSE)*. <https://doi.org/10.1145/3180155.3180165>
- [30] Jane Hoffswell, Wilmot Li, and Zhicheng Liu. 2020. Techniques for Flexible Responsive Visualization Design. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (2020). <https://doi.org/10.1145/3313831>

3376777

- [31] Jane Hoffswell, Arvind Satyanarayan, and Jeffrey Heer. 2018. Augmenting Code with In Situ Visualizations to Aid Program Understanding. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM. <https://doi.org/10.1145/3173574.3174106>
- [32] Marius Högreför and Hans-Jörg Schulz. 2019. *ReVize: A Library for Visualization Toolchaining with Vega-Lite*. Eurographics. <https://doi.org/handle/10.2312/stag20191375> ISSN: 2617-4855.
- [33] Aspen K Hopkins, Michael Correll, and Arvind Satyanarayan. 2020. VisualLint: Sketchy In Situ Annotations of Chart Construction Errors. In *Computer Graphics Forum*. <https://doi.org/10.1111/cgf.13975>
- [34] Kevin Hu, Diana Orghian, and César Hidalgo. 2018. DIVE: A Mixed-Initiative System Supporting Integrated Data Exploration Workflows. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. ACM, 1–7. <https://doi.org/10.1145/3209900.3209910>
- [35] John D. Hunter. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- [36] MongoDB Inc. 2020. MongoDB. <https://www.mongodb.com>
- [37] Zhongjun Jin, Michael R. Anderson, Michael Cafarella, and H. V. Jagadish. 2017. Foofah: Transforming Data By Example. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 683–698. <https://doi.org/10.1145/3035918.3064034>
- [38] jupyter. [n.d.]. Interactive Widgets. <https://jupyter.org/widgets> Accessed December 16, 2020.
- [39] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 3363–3372. <https://doi.org/10.1145/1978942.1979444>
- [40] 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. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. ACM.
- [41] Nam Wook Kim, Eston Schweickart, Zhicheng Liu, Mira Dontcheva, Wilnot Li, Jovan Popovic, and Hanspeter Pfister. 2016. Data-Driven Guides: Supporting Expressive Design for Information Graphics. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2016), 491–500. <https://doi.org/10.1109/TVCG.2016.2598620>
- [42] Younghoon Kim and Jeffrey Heer. 2021. Gemini: A Grammar and Recommender System for Animated Transitions in Statistical Graphics. *IEEE Transactions on Visualization and Computer Graphics* (2021).
- [43] Gordon Kindlmann and Carlos Scheidegger. 2014. An Algebraic Process for Visualization Design. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2181–2190. <https://doi.org/10.1109/TVCG.2014.2346325>
- [44] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E. Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B. Hamrick, Jason Grout, and Sylvain Corlay. 2016. Jupyter Notebooks—a publishing format for reproducible computational workflows. In *ELPUB*. 87–90. <https://doi.org/10.3233/978-1-61499-649-1-87>
- [45] Kari Lavikka, Jaana Oikonen, Rainer Lehtonen, Johanna Hynninen, Sakari Hietanen, and Sampsa Hautaniemi. 2020. GenomeSpy: Grammar-Based Interactive Genome Visualization. (2020).
- [46] Doris Jung-Lin Lee. 2020. Insight Machines: The Past, Present, and Future of Visualization Recommendation. <https://medium.com/multiple-views-visualization-research-explained/insight-machines-the-past-present-and-future-of-visualization-recommendation-2185c33a09aa> Multiple Views Visualization Research Explained.
- [47] Yun Young Lee, Nicholas Chen, and Ralph E. Johnson. 2013. Drag-and-Drop Refactoring: Intuitive and Efficient Program Transformation. In *International Conference on Software Engineering (ICSE)*. <https://doi.org/10.1109/ICSE.2013.6606548>
- [48] Germán Leiva, Nolwenn Maudet, Wendy Mackay, and Michel Beaudouin-Lafon. 2019. Enact: Reducing Designer-Developer Breakdowns When Prototyping Custom Interactions. *ACM Transactions on Computer-Human Interaction (TOCHI)* 26, 3, Article 19 (May 2019), 48 pages. <https://doi.org/10.1145/3310276>
- [49] Jianping Kelvin Li and Kwan-Liu Ma. 2021. P6: A Declarative Language for Integrating Machine Learning in Visual Analytics. *IEEE Transactions on Visualization and Computer Graphics* (2021).
- [50] 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*. ACM, 1–13. <https://doi.org/10.1145/3173574.3173697>
- [51] Aran Lunzer, Amelia McNamara, and Robert Krahn. 2014. LivelyR: Making R charts livelier. In *useR! Conference*.
- [52] Kwan-Liu Ma. 2000. Visualizing visualizations. User interfaces for managing and exploring scientific visualization data. *IEEE Computer Graphics and Applications* 20, 5 (2000), 16–19.
- [53] Jock Mackinlay. 1986. Automating the Design of Graphical Presentations of Relational Information. *ACM Trans. Graph.* 5, 2 (1986), 110–141. <https://doi.org/10.1145/22949.22950>
- [54] Jock Mackinlay, Pat Hanrahan, and Chris Stolte. 2007. Show Me: Automatic Presentation for Visual Analysis. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1137–1144.
- [55] Joe Marks, Brad Andalman, Paul A Beardsley, William Freeman, Sarah Gibson, Jessica Hodgins, Thomas Kang, Brian Mirtich, Hanspeter Pfister, Wheeler Ruml, et al. 1997. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 389–400.
- [56] 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*. ACM. <https://doi.org/10.1145/3125571.3125585>
- [57] Andrew McNutt and Gordon Kindlmann. 2018. Linting for Visualization: Towards a Practical Automated Visualization Guidance System. In *VisGuides: 2nd Workshop on the Creation, Curation, Critique and Conditioning of Principles and Guidelines in Visualization*.
- [58] Andrew McNutt, Gordon Kindlmann, and Michael Correll. 2020. Surfacing Visualization Mirages. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (2020). <https://doi.org/10.1145/3313831.3376420>
- [59] 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. <https://doi.org/10.1016/j.jvlc.2017.10.001>
- [60] interact. 2020. data-explorer. <https://github.com/interact/data-explorer>
- [61] interact. 2020. papermill. <https://github.com/interact/papermill>
- [62] Deokgun Park, Steven M. Drucker, Roland Fernandez, and Niklas Elmqvist. 2018. Atom: A Grammar for Unit Visualizations. *IEEE Transactions on Visualization and Computer Graphics* 24, 12 (2018), 3032–3043. <https://doi.org/10.1109/TVCG.2017.2785807>
- [63] Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. 2016. Foundations of JSON Schema. In *Proceedings of the 25th International Conference on World Wide Web*. <https://doi.org/10.1145/2872427.2883029>
- [64] Ate Poorthuis, Lucas van der Zee, Grace Guo, Jo Hsi Keong, and Bianchi Dy. 2020. Florence: a Web-based Grammar of Graphics for Making Maps and Learning Cartography. *Cartographic Perspectives* (2020).
- [65] Xiaoying Pu and Matthew Kay. 2018. The Garden of Forking Paths in Visualization: A Design Space for Reliable Exploratory Visual Analytics. In *2018 IEEE Evaluation and Beyond-Methodological Approaches for Visualization (BELIV)*. IEEE, 37–45. <https://doi.org/10.1109/BELIV.2018.8634103>
- [66] Xiaoying Pu and Matthew Kay. 2020. A Probabilistic Grammar of Graphics. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. <https://doi.org/10.1145/3313831.3376466>
- [67] Donghao Ren, Bongshin Lee, and Matthew Brehmer. 2018. Chartulator: Interactive construction of bespoke chart layouts. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 789–799.
- [68] Hugo Romat, Nathalie Henry Riche, Ken Hinckley, Bongshin Lee, Caroline Appert, Emmanuel Pietriga, and Christopher Collins. 2019. ActiveLink: (Th) Inking with Data. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–13. <https://doi.org/10.1145/3290605.3300272>
- [69] Hans Rosling and Zhongxing Zhang. 2011. Health advocacy with Gapminder animated statistics. *Journal of Epidemiology and Global Health* 1, 1 (2011), 11–14. <https://doi.org/10.1016/j.jegh.2011.07.001>
- [70] RStudio. [n.d.]. Shiny. <https://shiny.rstudio.com/> Accessed December 16, 2020.
- [71] Bahador Saket, Samuel Huron, Charles Perin, and Alex Endert. 2019. Investigating Direct Manipulation of Graphical Encodings as a Method for User Interaction. *IEEE Transactions on Visualization and Computer Graphics* (2019). <https://doi.org/10.1109/TVCG.2019.2934534>
- [72] Bahador Saket, Lei Jiang, Charles Perin, and Alex Endert. 2019. Liger: Combining Interaction Paradigms for Visual Analysis. *arXiv* (July 2019), arXiv:1907.08345. <https://ui.adsabs.harvard.edu/abs/2019arXiv190708345S/abstract>
- [73] Bahador Saket, Hannah Kim, Eli T. Brown, and Alex Endert. 2016. Visualization by Demonstration: An Interaction Paradigm for Visual Data Exploration. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2016), 331–340. <https://doi.org/10.1109/TVCG.2016.259883>
- [74] Arvind Satyanarayan and Jeffrey Heer. 2014. Lyra: An Interactive Visualization Design Environment. In *Eurographics Conference on Visualization*, Vol. 33. 10. <https://doi.org/10.1111/cgf.12391>
- [75] Arvind Satyanarayan, Bongshin Lee, Donghao Ren, Jeffrey Heer, John Stasko, John Thompson, Matthew Brehmer, and Zhicheng Liu. 2020. Critical Reflections on Visualization Authoring Systems. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2020), 461–471. <https://doi.org/10.1109/TVCG.2019.2934281>
- [76] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics* (2016). <https://doi.org/10.1109/TVCG.2016.2598620>

- 2016.2599030
- [77] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. 2016. Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (2016), 659–668. <https://doi.org/10.1109/TVCG.2015.2467091>
 - [78] Arvind Satyanarayan, Kanit Wongsuphasawat, and Jeffrey Heer. 2014. Declarative Interaction Design for Data Visualization. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, 669–678. <https://doi.org/10.1145/2642918.2647360>
 - [79] Ben Shneiderman. 1983. Direct Manipulation: A Step Beyond Programming Languages. *Proceedings of the Joint Conference on Easier and More Productive Use of Computer Systems.(Part-II): Human Interface and the User Interface-Volume 1981* (1983).
 - [80] 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.
 - [81] Wenbo Tao, Xinli Hou, Adam Sah, Leilani Battle, Remco Chang, and Michael Stonebraker. 2020. Kyrix-S: Authoring Scalable Scatterplot Visualizations of Big Data. *IEEE Transactions on Visualization and Computer Graphics* (2020).
 - [82] TIBCO. 2020. Spotfire. <https://www.tibco.com/products/tibco-spotfire>
 - [83] Jacob VanderPlas, Brian E. Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. 2018. Altair: Interactive Statistical Visualizations for Python. *Journal of Open Source Software* 3, 32 (2018), 1057. <https://doi.org/10.21105/joss.01057>
 - [84] Vega. 2020. Aggregate Bar Chart. https://vega.github.io/vega-lite/examples/bar_aggregate.html
 - [85] Vega. 2020. Box Plot with Min/Max Whiskers. https://vega.github.io/vega-lite/examples/boxplot_minmax_2D_vertical.html
 - [86] Vega. 2020. Editor/IDE for Vega and Vega-Lite. <https://vega.github.io/editor/>.
 - [87] Vega. 2020. One Dot per Zipcode in the U.S. https://vega.github.io/vega-lite/examples/geo_circle.html
 - [88] Vega. 2020. Vega-Lite Documentation. <https://vega.github.io/vega-lite> Accessed August 19, 2020.
 - [89] Bret Victor. 2013. Drawing Dynamic Visualizations. <https://vimeo.com/66085662>
 - [90] Fernanda B. Viegas, Martin Wattenberg, Frank Van Ham, Jesse Kriss, and Matt McKeon. 2007. Many Eyes: A Site for Visualization at Internet Scale. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1121–1128. <https://doi.org/10.1109/TVCG.2007.70577>
 - [91] Chenglong Wang, Yu Feng, Rastislav Bodik, Alvin Cheung, and Isil Dillig. 2019. Visualization by Example. *Proceedings of the ACM on Programming Languages* 4, POPL (Dec. 2019), 49:1–49:28. <https://doi.org/10.1145/3371117>
 - [92] Hadley Wickham. 2010. A Layered Grammar of Graphics. *Journal of Computational and Graphical Statistics* 19, 1 (2010), 3–28. <https://doi.org/10.1198/jcgs.2009.07098>
 - [93] Hadley Wickham, Dianne Cook, Heike Hofmann, and Andreas Buja. 2010. Graphical Inference for Infovis. *IEEE Transactions on Visualization and Computer Graphics* 16, 6 (2010), 973–979. <https://doi.org/10.1109/TVCG.2010.161>
 - [94] Hadley Wickham and Heike Hofmann. 2011. Product Plots. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (2011), 2223–2230. <https://doi.org/10.1109/TVCG.2011.227>
 - [95] Leland Wilkinson. 2013. *The Grammar of Graphics*. Springer.
 - [96] Krist Wongsuphasawat. 2020. Encodable: Configurable Grammar for Visualization Components. *IEEE Transactions on Visualization and Computer Graphics* (2020).
 - [97] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2015. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (2015), 649–658. <https://doi.org/10.1109/TVCG.2015.2467191>
 - [98] 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*. ACM, 2648–2659. <https://doi.org/10.1145/3025453.3025768>
 - [99] Jo Wood, Alexander Kachkaev, and Jason Dykes. 2019. Design Exposition with Literate Visualization. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2019), 759–768. <https://doi.org/10.1109/TVCG.2018.2864836>
 - [100] Yifan Wu, Joseph M. Hellerstein, and Arvind Satyanarayan. 2020. B2: Bridging Code and Interactive Visualization in Computational Notebooks. In *UIST '20: The 33rd Annual ACM Symposium on User Interface Software and Technology*. ACM, 152–165. <https://doi.org/10.1145/3379337.3415851>
 - [101] Zheguang Zhao, Emanuel Zraggen, Lorenzo De Stefani, Carsten Binnig, Eli Upfal, and Tim Kraska. 2017. Safe Visual Data Exploration. In *Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD '17*. ACM, 1671–1674. <https://doi.org/10.1145/3035918.3058749>
 - [102] Jonathan Zong, Dhiraj Barnwal, Rupayan Neogy, and Arvind Satyanarayan. 2021. Lyra 2: Designing Interactive Visualizations by Demonstration. *IEEE*

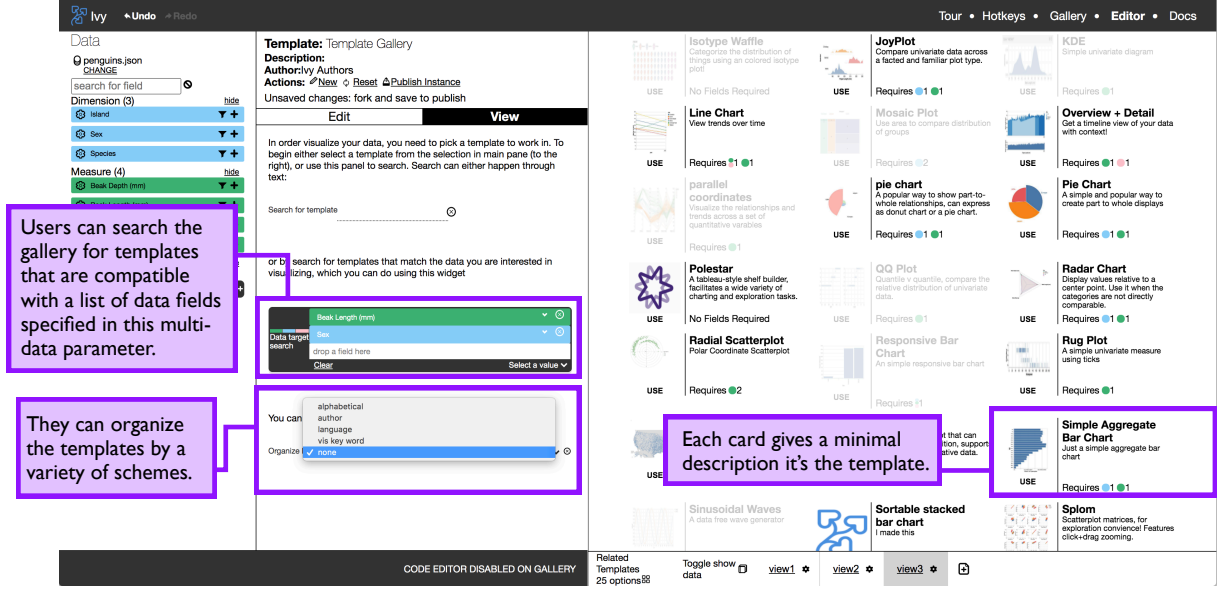


Figure 9: Users can search for templates via text or catalog search in the gallery, which is a set of user created templates (hosted on a communal server) and system templates.

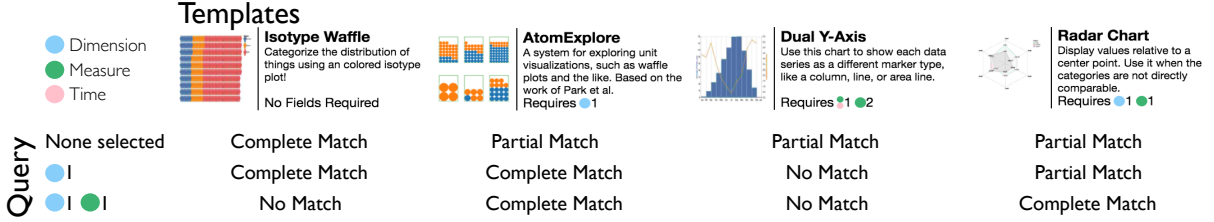


Figure 10: Catalog search (Sec. 4.4.1) compares templates to a set of data fields, which can yield a partial match (no conflict but unrenderable), a complete match (renderable), or no match (conflict).

A APPENDIX

In this appendix we expand upon several elements of the design, implementation, and evaluation of Ivy discussed in the main paper.

A.1 Catalog Search Matching Heuristic

Fig. 9 shows the gallery of Ivy templates in our current implementation. Here we provide a more precise description of the heuristic used to perform the template matching in our catalog search.

For a template t (among the set of templates T) with a set of data parameters d , (where each parameter has a set of allowed types $d_i\tau$), and a search S consisting of a set of data columns $\{c_i\}$ which each have a single type $c_i\tau$, then S is a *partial match* for t if there is an injective mapping m between them such that

$$(\exists m)(\forall c_i \in S)(\exists d_j \in d)(m : c_i \rightarrow d_j : c_i\tau \in d_j\tau) \quad (11)$$

Correspondingly S is a *full match* for t if

$$(\exists m)(\forall rd_i \in rd)(\exists c_j \in S)(m : c_j \rightarrow rd_i : c_j\tau \in rd_i\tau) \quad (12)$$

where we define the required set of data param in t as $rd \subseteq d$. This check is bounded by the size of S and T , so a search across the templates will take $O(|S||T|)$. We illustrate this in Fig. 10.

A.2 Implementation Details

Ivy is a TypeScript React-Redux application. We were motivated to use React as the basis of our application as it provides an opinionated approach on how to build additional renderers, which is important for our approach to extensibility. Our template server is a cloud-based node.js server backed by PostgreSQL.

As described in Sec. 6.1.1 our system is designed to be extensible: support for each specification language is defined through an extension interface comprising metadata (such as a JSON Schema describing the syntax), a React component [105] that exposes the rendering function of the language, and rewrite rule definitions that help users abstract specifications into templates. While Ivy currently supports a relatively limited class of rewrite rules, future work could extend the approach with more expressive languages for transforming structured data (such as in CDuce [103] and XDuce [107]).

Our current implementation supports four languages—Vega, Vega-Lite, Atom, and a toy data table language—which serves as a limited demonstration of the validity of our extensible approach. As

JSON-mediated visualization grammars continue to gain popularity, additional languages will inevitably emerge to solve problems unaddressed in prior efforts. Future languages could support more complex rendering schemes, focusing on particular domains such as geospatial analytics [106], 3D visual analytics, pivot tables (perhaps simplifying the language of VizQL [24]), or even on the chart recommendation language CompassQL [111]—which would enable task-specific variations of Voyager [98].

A.3 Templates for Vega-Lite Gallery

As described in [Sec. 5.1.1](#), we aimed to factor the Vega-Lite examples into templates in reasonable ways, a heuristic which was guided by the minimization of complexity and the maximization of simplification. We illustrate these metrics in [Fig. 11](#). The metrics in this figure are computed following [Equation 10](#). Most templates exhibit a compression greater than 1, indicating that the template is better than simply concatenating the examples together. Those that do worse tend to have particular affordances to the template usable and also tend to only cover a single example, limiting their compression. See <https://osf.io/cture/> for further details.

A.4 Templates to Reproduce Chart Choosers

We describe here in greater detail our templatzation of the Google Sheets chart chooser (described in [Sec. 5.1.2](#)), as well as an additional chart corpus provided by Russell [109]. The latter fell outside the narrative in the main body of the paper, but we include it here as an example of the chart making culture in one particular organization. [Table 1](#) summarizes the resulting templates described below.

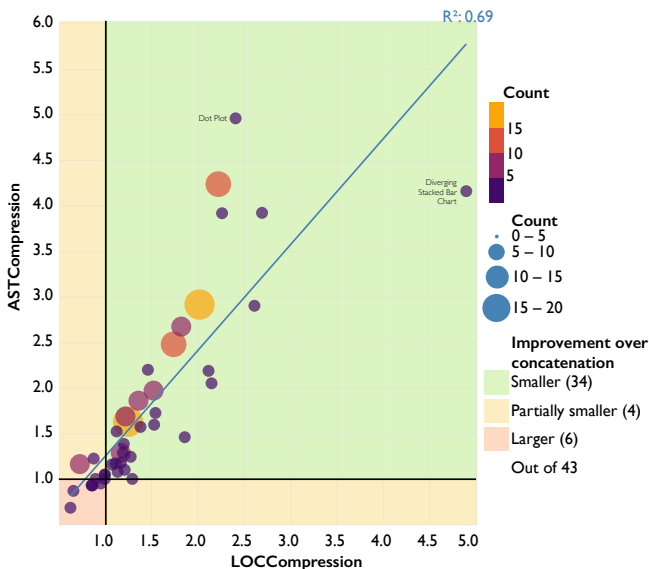


Figure 11: The concatenation ratios across our reproduction of the Vega-Lite gallery between the compression of the Abstract Syntax Tree (a stand-in for complexity minimization) and Line of Code Compression (a stand-in simplification). Higher is better in both cases.

A.4.1 Google Sheets. The Sheets chart chooser consists of 32 options. We reproduce 29 of these through 16 templates, as summarized in Fig. 12. We skipped “3D Pie” charts because there is not yet a dominant grammar for browser-based non-VR 3D visual analytics, although several recent works have put forward interesting approaches [104, 110]. We skipped “Org. Charts” because they fall outside of our tabular data model, requiring a hierarchical one. Finally, we skipped “Timelines” because we do not currently support the data manipulations required to support textual annotations as required by this chart form. Each of these deficiencies could be addressed in future work, such as by extending the range of languages supported.

A.4.2 Russell Survey. This survey [109] of internal presentations at Google included approximately 1,300 charts, which were grouped into 15 distinct visual forms. We reproduce 10 of these through 11 templates. There are more templates than charts because, following Sheets, we split Russell’s “Map” into two templates, “Country Choropleth” and “Scatter Map”. Of the 5 charts from this survey we skipped, 3 involve a non-tabular data model, 1 requires domain-specific data (Lam et al.’s SessionViewer [108]), and 1 uses annotations. Among the 16 plus 11 templates described, 18 are distinct.

While informative, this selection covers one particular analytic culture and one family of tool’s designs. For instance, Russell’s review found SessionViewer [108], a system for understanding web search usage behaviors, made up 2.1% of the review corpus. A review of a different corpus would likely yield a different selection of charts. Furthermore, this selection of charts is also a symptom of software availability. To wit: unit visualizations tend to be uncommon because few systems tend to support them [62].

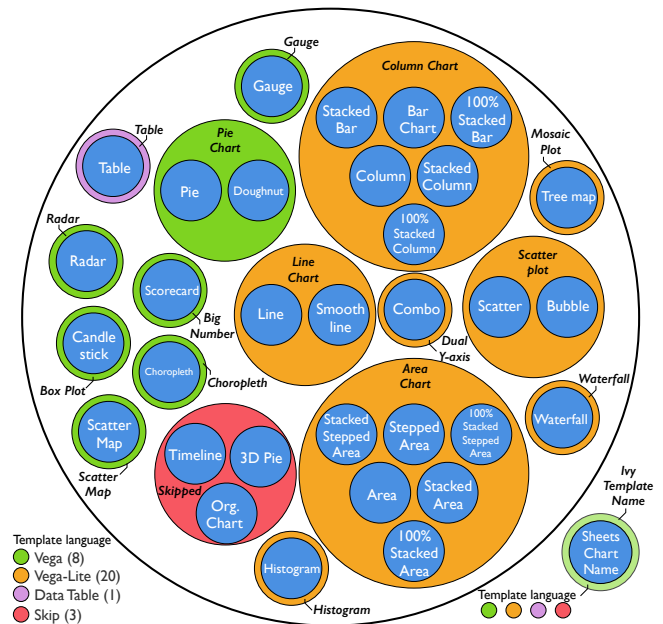


Figure 12: We created 16 Ivy templates that reconstruct the functionality of 29 of the 32 charts in the Google Sheets chart chooser, a 1.8x compression.

Table 1: We now give additional detail regarding the overlap between our coverage of the Google Sheets Chart Chooser Options and the charts described in Russell’s survey. Options that are unsupported under the current set of grammars are marked as Skip . Each template was created by starting from a blank template, from using Polestar to approximate the right behavior, or by abstracting an example found online.

Russel Chart	Sheets Chart	Ivy Template	Russell	Sheets	Language	Creation Method
line graph	Smooth Line Chart	Line Chart	✓	✓	Vega-lite	Polestar
line graph	Line Chart	Line Chart	✓	✓	Vega-lite	Polestar
histogram	Histogram chart	Histogram	✓	✓	Vega-lite	Polestar
table	Table chart	Table	✓	✓	Data table	Blank
pie	Pie chart	Pie chart	✓	✓	Vega	Example
pie	Doughnut chart	Pie chart	✓	✓	Vega	Example
stack histogram	Column chart	Column Chart	✓	✓	Vega-lite	Polestar
stack histogram	Stacked column chart	Column Chart	✓	✓	Vega-lite	Polestar
stack histogram	100% stacked column chart	Column Chart	✓	✓	Vega-lite	Polestar
stack histogram	Bar chart	Column Chart	✓	✓	Vega-lite	Polestar
stack histogram	Stacked bar chart	Column Chart	✓	✓	Vega-lite	Polestar
stack histogram	100% stacked bar chart	Column Chart	✓	✓	Vega-lite	Polestar
box plot	Candlestick chart	Candle stick	✓	✓	Vega	Example
scatterplot	Scatter chart	Scatterplot	✓	✓	Vega-lite	Blank
scatterplot	Bubble chart	Scatterplot	✓	✓	Vega-lite	Blank
map	Geo chart	Country Choropleth	✓	✓	Vega	Example
map	Geo chart with markers	Scatter Map	✓	✓	Vega	Blank
timeline	Timeline chart	Skip	✓	✓	N/A	N/A
pie	3D pie chart	Skip	✓	✓	N/A	N/A
heatmap	N/A	Heatmap	✓	✗	Vega-lite	Example
sunburst	N/A	Sunburst	✓	✗	Vega	Example
arc/node graph	N/A	Skip	✓	✗	N/A	N/A
SessionView	N/A	Skip	✓	✗	N/A	N/A
Sankey	N/A	Skip	✓	✗	N/A	N/A
force vector	N/A	Skip	✓	✗	N/A	N/A
N/A	Organizational chart	Skip	✗	✓	N/A	N/A
N/A	Combo chart	Dual Y-axis	✗	✓	Vega-lite	Example
N/A	Waterfall chart	Waterfall	✗	✓	Vega-lite	Example
N/A	Radar chart	Radar	✗	✓	Vega	Example
N/A	Gauge chart	Gauge	✗	✓	Vega	Example
N/A	Scorecard chart	BigNumber	✗	✓	Vega	Blank
N/A	Tree map chart	Mosaic Plot	✗	✓	Vega-lite	Example
N/A	Area chart	Area chart	✗	✓	Vega-lite	Polestar
N/A	Stacked area chart	Area chart	✗	✓	Vega-lite	Polestar
N/A	100% stacked area chart	Area chart	✗	✓	Vega-lite	Polestar
N/A	Stepped area chart	Area chart	✗	✓	Vega-lite	Polestar
N/A	Stacked stepped area chart	Area chart	✗	✓	Vega-lite	Polestar
N/A	100% stacked stepped area chart	Area chart	✗	✓	Vega-lite	Polestar

A.5 User Study Prompts

Here we provide the text of the tasks involved in the user study. The full study instrument can be found in the supplementary materials. These questions were divided into two sections, *Tutorial*, which involved substantial guidance, and *Independent*, which were more freeform. These tasks were selected because they were similar to tasks that one might address in similar systems.

A.5.1 Tutorial Tasks.

- (1) In this task you will make a small multiple log-log scatter plot of happiness vs population for 2015 colored and faceted by region (such as by row or column) where tooltipping shows (among other data) the name of the country. To do so you make use of the Polestar template. Start by finding the Polestar template in the gallery, navigate to it. Now fill in the appropriate fields for X and Y. To make the tooltip reveal useful information, place the Country field onto the detail target. Set the Region to Column as well. Don't forget to filter the appropriate year. What correlation can you see? Give your answer in plain text.
- (2) In this task we will make a SPLOM (scatter plot matrix) for our dataset. Start by creating a new view, and navigate to the Polestar template. Place the row and column cards on the x and y data targets respectively. Next, select 3 measures of interest (you decide!), place each of them in both of the "row" and "column" multi targets which are under the "meta columns section". Now click the lighting bolt next the Color field and select 3 dimensions of interest (you decide!). Just as before it might be helpful to place Country into Detail. What correlations can you find? Give your answer in plain text.
- (3) Next you will make use of a particular template from the gallery. Specifically, you will make a radial scatterplot. This task is a little different in that you will use a different dataset. Open the gallery in a new tab and find the template that will allow you to make a radial scatterplot. Once there select the penguins dataset. What is the most interesting combination of variables you can find? Can you use fanout to effectively move through these options? These plots are a little big. Why don't you try to navigate to the code body and change their height and width to be something a little more reasonable? Copy the code from the output into the below box.
- (4) Next you will try out making a template, specifically let's make a heatmap. Load up the happiness dataset once again. Once again we will start with Polestar. Start by placing the CountryType and GovernmentType variables on to the X and Y targets. Then place the happiness field onto the Color Field. Feel free to adjust the heatmap as you like. You should probably change the mark type to make it more heatmap like! When you are ready, click the "Fork" button and select "Just output". Click the suggestion for CountryType that creates and configures a new field, this will create a new datatarget for this field while keeping the current graphic in place. Select the gear to the right of this new field and give it an informative field name, and select only the appropriate field data type. Do the same for GovernmentType. Next, let's make our heatmap be better able to describe various aggregates. Create a new List widget and add options for each of the aggregation types (the ones that make the most sense are probably count and distinct, but you are welcome to use others,

see <https://vega.github.io/vega-lite/docs/>). Give the widget a descriptive name (no spaces though!) and replace the word "count" in the Body with your new name wrapped in brackets, like so "[YOUR NEW NAME]". You should now be able to switch through various aggregations (or fan across them). Finally, give your template a name and description, and then publish it!

A.5.2 Independent Tasks.

- (1) How many rows are in this dataset? How many countries are represented in this dataset? How many countries are in each region? Please give your answers in plain text. (You can answer the last question using the JSON output.)
- (2) Create a new template by adapting https://vega.github.io/vega-lite/examples/boxplot_minmax_2D_vertical.html to this dataset in a useful manner. The choice of columns is up to you. Try adding height and width sliders. Try enabling switching between types of box plot e.g. https://vega.github.io/vega-lite/examples/boxplot_2D_vertical.html). Copy the body of your template into the space below.
- (3) Please make charts that answer the following questions. What is the global trend in corruption? Do bigger countries tend to be happier? Use code from the output for your answers.
- (4) Please combine the following examples drawn from the Vega-Lite gallery into an Ivy template: 1. https://vega.github.io/vega-lite/examples/bar_aggregate.html 2. https://vega.github.io/vega-lite/examples/bar_aggregate_sort_by_encoding.html. The particular choice of columns and features is up to you. Hint: the big difference is that one is sorted and the other is not! If you are having trouble with this task it may be helpful to check out the documentation of the template language found on the web page. Answer this question by giving the body of the template you've created.

APPENDIX REFERENCES

- [103] Véronique Benzaken, Giuseppe Castagna, and Alain Frisch. 2003. CDuce: An XML-Centric General-Purpose Language. *ACM SIGPLAN Notices* (2003). <https://doi.org/10.1145/944746.944711>
- [104] Peter WS Butcher, Nigel W. John, and Panagiotis D. Ritsos. 2019. VRIA-A Framework for Immersive Analytics on the Web. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–6. <https://doi.org/10.1145/3290607>
- [105] Facebook. 2020. React – A JavaScript library for building user interfaces. <https://reactjs.org/>
- [106] Shan He. 2020. Kepler.gl: Large-scale WebGL-powered Geospatial Data Vis. <http://kepler.gl/>
- [107] Haruo Hosoya and Benjamin C. Pierce. 2003. XDuce: A Statically Typed XML Processing Language. *ACM Transactions on Internet Technology* (2003). <https://doi.org/10.1145/767193.767195>
- [108] Heidi Lam, Daniel Russell, Diane Tang, and Tamara Munzner. 2007. Session Viewer: Visual Exploratory Analysis of Web Session Logs. In *2007 IEEE Symposium on Visual Analytics Science and Technology*. IEEE, 147–154. <https://doi.org/10.1109/VAST.2007.4389008>
- [109] Daniel M. Russell. 2016. Simple is Good: Observations of Visualization Use Amongst the Big Data Digerati. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*. 7–12. <https://doi.org/10.1145/2909132.2933287>
- [110] Ronell Sicut, Jiabao Li, JunYoung Choi, Maxime Cordeil, Won-Ki Jeong, Benjamin Bach, and Hanspeter Pfister. 2018. DXR: A Toolkit for Building Immersive Data Visualizations. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2018), 715–725. <https://doi.org/10.1109/TVCG.2018.2865152>
- [111] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2016. Towards A General-Purpose Query Language for Visualization Recommendation. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. ACM, 4. <https://doi.org/10.1145/2939502.2939506>