How Data Analysts Use a Visualization Grammar in Practice

Xiaoying Pu
University of California, Merced
Merced, CA, USA
xpu@umich.edu

Matthew Kay
Northwestern University
Evanston, IL, USA
mjskay@northwestern.edu

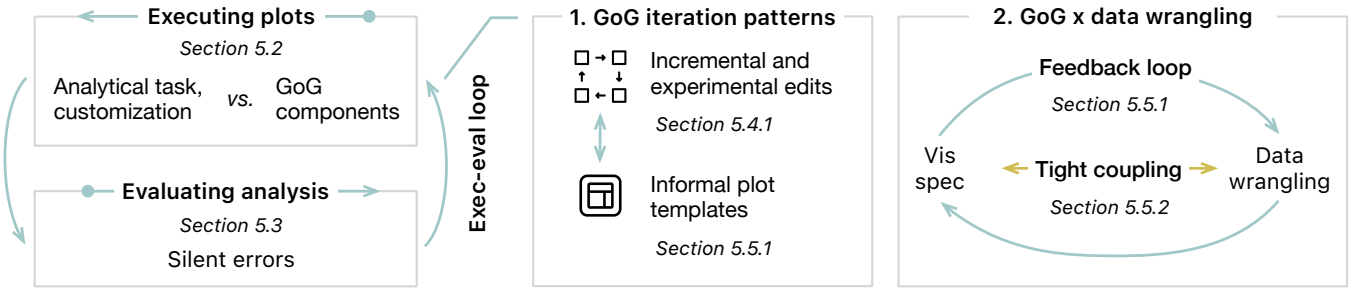


Figure 1: Overview of our study results on how analysts used the Grammar of Graphics (GoG)-based `ggplot2`.

ABSTRACT

Visualization grammars, often based on the Grammar of Graphics (GoG), have much potential for augmenting data analysis in a programming environment. However, we do not know how analysts conceptualize grammar abstractions, or how a visualization grammar works with data analysis in practice. Therefore, we qualitatively analyzed how experienced analysts ($N = 6$) from TidyTuesday, a social data project, wrangled and visualized data using GoG-based ggplot2 without given tasks in R Markdown. Though participants' analysis and customization needs could mismatch with GoG component design, their analysis processes aligned with the goal of GoG to expedite visualization iteration. We also found a feedback loop and tight coupling between visualization and data transformation code, explaining both participants' productivity and their errors. From these results, we discuss how future visualization grammars can become more practical for analysts and how visualization grammar and analysis tools can better integrate within a programming (*i.e.*, computational notebook) environment.

CCS CONCEPTS

• **Human-centered computing** → **Empirical studies in visualization**; *Visualization theory, concepts and paradigms.*

KEYWORDS

Visualization grammar, TidyTuesday, computational notebook

CHI '23, April 23–28, 2023, Hamburg, Germany

ACM ISBN 978-1-4503-9421-5/23/04…$15.00
https://doi.org/10.1145/3544548.3580837

ACM Reference Format:
Xiaoying Pu and Matthew Kay. 2023. How Data Analysts Use a Visualization Grammar in Practice. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23), April 23–28, 2023, Hamburg, Germany.* ACM, New York, NY, USA, 22 pages. https://doi.org/10.1145/3544548.3580837

1 INTRODUCTION

Creating visualizations is a significant part of data analysts' work. Through writing code in computational notebooks, analysts can interleave visualization with analyses to explore data, generate hypotheses, and evaluate modeling results. Given the utility of visualizations, analysts need a way to easily and reliably specify visualizations in their data work. One solution is to use *visualization grammars*, formalisms that create a wide range of visualizations by combinations of grammar components. Building from the original Grammar of Graphics (GoG) [70], visualization grammars have proliferated in the past decade (in literature as reviewed by McNutt [38] and in major scripting languages). Popular visualization grammars include `ggplot2` [66] in the R language and the Vega ecosystem [55–57] in Javascript.

In theory, GoG-based visualization grammars can be beneficial. The Grammar of Graphics is intended to be expressive, using a combination of components to describe a wide range of visualizations with simple and elegant specifications [70]. When evaluated with usability heuristics (*i.e.* the cognitive dimensions of notations [6]), GoG-based grammars have been found to promote iteration and encourage the exploration of visualization designs [48, 57].

Despite the popularity and theoretical benefits of visualization grammars, we know little about whether or how analysts take advantage of these grammars *in practice* [38, 49], especially given the potential tension between GoG design intention and analysts' needs. Understanding how analysts use visualization grammars can be crucial to improving grammar designs. Since GoG is designed for expressiveness, learning from analysts' conceptualizations and usage patterns can make grammars more aligned with analysts'

tasks and analysis less error-prone. Contextualizing grammar use within data analysis can lead to grammar design more integrated into analysis workflows. Thus far in the literature, when assessing a new grammar or system building on a grammar, researchers have asked study participants to recreate visualizations [32, 44], but recreation does not tell us how analysts *explore* and *iterate* on analysis and visualizations. When study participants use a custom interface (*e.g.* Voyager 2 [74]), their exploration can be constrained by what the interface supports, compared to the full array of analyses they would have access to in a programming environment. In this paper, we directly study how real-world analysts use GoG-based `ggplot2` to wrangle and plot data for analyses in computational notebooks. We answer the following research questions:

RQ1. What are the analysts' conceptualization and usage patterns of the Grammar of Graphics (GoG) [65, 70]? We interpret analysts' use of `ggplot2` in terms of the GoG abstractions for generalizability.

RQ2. What is the interplay between a GoG-based visualization grammar and programming-based data wrangling in an analysis environment? Our other focus is on data wrangling, a significant part of the visualization process.

The `#TidyTuesday` R community provided us with an opportunity to answer the above questions. `#TidyTuesday` is a "weekly social data project", where participants *explore*, *wrangle*, and *visualize* weekly datasets in the R language and post their process and results on social media [41, 58]. We collected `#TidyTuesday` recordings from six ($N = 6$) participants with intermediate to advanced `ggplot2` experience, followed up with retrospective interviews, and analyzed this rich data set with reflexive thematic analysis. Summarized in Figure 1, we used an *execution-evaluation loop* [1] to explain participants' use of a visualization grammar in data analysis:

- When creating (*executing*) plots, participants' analysis and customization needs are sometimes mismatched with the GoG design. For example, participants who wanted to apply custom colors, positions, and angles faced difficulties because GoG only facilitates the mapping of data, not customizations, onto visual elements.
- Participants made hard-to-*evaluate* silent errors [37], where their data wrangling and visualization specifications implied different data semantics, producing plausible-looking plots without explicit errors.
- When viewing participants' analysis processes as an execution-evaluation loop, participants iterated and explored visualization alternatives visualizations as GoG was designed for, and they also made informal plot templates that encapsulated their visualization and wrangling code. Between GoG visualization and data wrangling specifications, we identified a feedback loop enabled by the modular design of GoG: plotting outputs inform subsequent data wrangling, and *vice versa*. There is also a tight coupling, where GoG specification and analysis need to be kept consistent to avoid errors.

Our findings can inform future visualization grammar designs: we offer suggestions for making grammars more practical for analysts' needs by supporting plot templates and customizations. We also discuss ways to help analysts evaluate and integrate visualization and analysis specifications in the computational notebook programming environment.

2 RELATED WORK

2.1 The Grammar of Graphics and `ggplot2`

The Grammar of Graphics (GoG) is an influential formalism for specifying statistical graphics [70]. The grammar consists of six types of components, including DATA and ELEMENT, and concise grammar rules, such as one for relating data attributes to visual attributes. GoG is powerful because it describes a wide range of visualizations (*i.e.*, being expressive [36]) through combinations of components. This is in contrast to using visualization templates [72] like in Google Charts and Charts.js,[1] where we might need to start over to change from a scatterplot to a bar chart. GoG inspired many visualization grammars in multiple languages. McNutt provides an in-depth, literature-oriented survey and analysis of 57 visualization grammars (broadly defined as JSON-style DSLs) [38]. In practice, GoG-inspired grammars include `ggplot2` [66] in R, Vega-Lite [55] in Javascript, Seaborn,[2] Altair (Vega-Lite frontend) [62], and plotnine[3] in Python, and Gadfly.jl, Algebra of Graphics in Julia.[4]

We studied how analysts used visualization grammars through `ggplot2` partly because its syntax directly corresponds to the underlying grammar components. `ggplot2` implements the Layered Grammar of Graphics, a re-parametrization of GoG proposed by Wickham [65]. We introduce the syntax of `ggplot2` with a snippet from participant 3 (P3) in our study, see Figure 2. Each `ggplot2` visualization consists of a default layer (`ggplot()`, line 3). The analyst can add (+) layers by instantiating geometries (`geom_point`, line 5) or statistical transformations (*e.g.*, density estimate). Geometries are also known as marks in Vega-Lite [55]. Aesthetics (a.k.a. encoding channels) are visual properties of the geometries that can vary with data variables, such as x-axis position and color. `aes()` (line 4) establish the aesthetic mapping (a.k.a. encoding) from data variables to aesthetics; here a variable about pumpkin size (`ott`, "over-the-top") is mapped onto the x-axis position aesthetic of the point geometry. Note that `alpha` and `size` in `geom_point()` are not part of an aesthetic mapping but hard-coded arguments not in the dataset `pumpkins`. We discuss how our findings may generalize to other GoG-based grammars in Section 6.4.

2.2 Evaluating the benefits of visualization grammars

Visualization grammars are associated with many benefits. Claims about being *expressive* [36] can be demonstrated by enumerating the types of visualizations a grammar can specify, as seen in Vega-Lite [55], ATOM [45], Canis [22], and Nebula [15]. Several studies have used the Cognitive Dimensions of Notations [6], a set of heuristics, to evaluate the usability of a grammar (*e.g.* Vega [57], Nebula [15], the Probabilistic Grammar of Graphics [48]). These

[1] Google charts: https://developers.google.com/chart; Charts.js: https://www.chartjs.org
[2] Seaborn with the "next-generation interface", see https://seaborn.pydata.org/nextgen/
[3] https://plotnine.readthedocs.io/en/stable/
[4] Gadfly.jl: http://gadflyjl.org/. Algebra of Graphics: http://juliaplots.org/AlgebraOfGraphics.jl/stable/

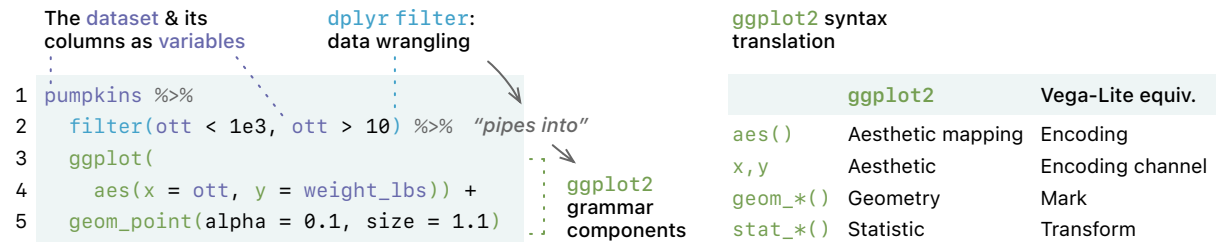


Figure 2: Left: example R code chunk from P3, showing a dataset `pumpkins`, data wrangling using the `filter()` function from the `dplyr` R package [69], and `ggplot2` specification. The results from data wrangling is "piped into" `ggplot2` by the `%>%` operator. Right: equivalences between the `ggplot2` and Vega-Lite [55] syntax for reader's reference.

dimensions imply that if a grammar evaluates favorably, it can promote the iteration of visualization specification and the exploration of the visualization design space, as argued in Pu & Kay [48].

Beyond heuristics, user studies have the potential to further evaluate expressiveness and how grammars promote exploration and iteration. However, existing literature has not studied these benefits directly due to task design and participant expertise. For study task designs, asking participants to recreate pre-specified visualizations with Gemini [32] or Vega-Lite [44] does not capture how analysts would have explored and iterated on visualization designs. Completing given visual analysis tasks in Voyager [74] or answering data questions in `ggplot2` [43] does not necessarily reflect how analysts would have explored data and visualization designs on their own. Another barrier to capturing iteration and exploration in a study can be the participants' expertise: evaluating a grammar with new users can be premature [25] because they need tutorials and may not take full advantage of the benefits of a grammar. In our study, we improve the evaluation of visualization grammars by studying visualization and analysis processes done without prescribed tasks, and by recruiting analysts experienced with `ggplot2` as participants.

2.3 Significance of the analysis context for understanding visualization specification

In descriptive models, data wrangling and analysis have been considered an integral part of the visualization process (not specific to using visualization grammars). For example, the visualization reference model by Card *et al.* includes a step for "raw data transforms" [12, Chapter 1]. As an extension to the reference model, Chi's visualization state model for operators allows multiple analysis pipelines and adopts a state-transition (data-operator) abstraction [17]. Munzner's nested model focuses on the abstraction from domain problem to generic *data operations* [42]. Through a sensemaking lens, Grolemund and Wickham further considered visualization a form of data transformation [26]. These models provide a theoretical motivation for including data analysis as part of the visualization specification process.

In data science practice, visualizations are specified in the context of data analysis. As an analysis medium, computational notebooks are documents where analysts can interleave analysis code, documentation, and visualizations, often used for exploratory data analysis (EDA) [61]. To illustrate, Figure 2 shows a code chunk from P3's R Markdown notebook [78] that contains both data wrangling (`filter`) and visualization specification. Other notebook environments include Jupyter notebook [47] and Observable.[5] As visualization grammars in common scripting languages (*e.g.* Python, R, Julia) get more adoption, we need a better understanding of how visualization specifications (with grammars) integrate with data analysis in notebooks. Wood proposed *litvis*, a notebook environment for integrating writing visualization code with documenting design expositions [75], though the focus is not on analysis. As Battle *et al.* pointed out, there is currently little research on how visualization grammars incorporate into analysts' *implementation workflows* [3]. Chattopadhyay *et al.* identified visualization-related pain points in notebook use, including customizing the plots and interfacing between data exploration and visualization tools [13]; however, they did not analyze specific visualization grammars or libraries. From a sample of Stack Overflow posts, Battle *et al.* tallied the broader toolsets D3 (a visualization "kernel") [7] users employ [3], such as other JavaScript libraries, R/python, and Excel, but this dataset contained little description about analysts' workflow process. In this study, we pinpointed how analysts use a visualization grammar (*i.e.* `ggplot2`) during data analysis in R Markdown. We incorporated the analysis workflow information from participant recordings and formulated answers about how `ggplot2`, or GoG in general, works together with data analysis.

3 STUDY

The `#TidyTuesday` community provides an opportunity to study how analysts wrangle and visualize data in a practical setting. We collected existing and new recordings of participants completing `#TidyTuesday` projects at their own pace, and we followed up with retrospective interviews. Our goal is to answer **RQ1** about the conceptualization and use of GoG and **RQ2** about the interplay between visualization grammar and data analysis. Before recruiting started, the IRB at the University of Michigan determined the study to be exempt (HUM00201007).

3.1 Reasons to recruit from `#TidyTuesday`

`#TidyTuesday` is a community-based, "social data project in R" running since 2018 [41]. It provides a new dataset each week, and

[5] https://observablehq.com

participants wrangle and visualize the data following their interests. As the guidelines[6] of #TidyTuesday encourage, participants often share their visualizations, source code, even videos and animations of their creation process on social media. #TidyTuesday is popular—the hashtag has been tweeted more than 22,000 times as of November, 2022 [41]. We chose to recruit from the #TidyTuesday community for the following reasons:

- Shared goal and convention: according to Shrestha *et al.*, #TidyTuesday is a connected community with the shared goal of improving their analysis and visualization skills [58]. From our observation, #TidyTuesday participants tend to wrangle and visualize data for their weekly datasets as self-contained projects, which helped us avoid giving task instructions. Participants also mostly use ggplot2, a grammar we aimed to study.
- Culture of sharing: #TidyTuesday participants post their visualizations, even recordings of their process, on social media. We collected these recordings as part of our study.
- Variety of expertise: analysts from novices to experts participate in #TidyTuesday albeit in different roles [58]. Compared to recruiting from college classrooms, #TidyTuesday participants who are data science professionals might bring their practical experiences and expertise. We recruited people who successfully completed #TidyTuesday projects—they self-identified as intermediate to expert-level ggplot2 users and had relevant industry experience or academic training, see Table 1.

3.2 Participants

We recruited in two phases between September and November 2021. In **Phase 1**, we contacted Twitter, Youtube, and Twitch users who posted recordings of their data analysis with the #TidyTuesday hashtag. Three out of 13 potential participants joined the study; the inclusion criteria are:

(1) The recording is a self-contained, unedited data analysis session with iterations on > 3 ggplot2 visualizations.
(2) R source code is available or shown in the recording.
(3) For better recall during the interview, the recording is no more than a month old.[7]
(4) The participant does not reside in the EU or UK for GDPR compliance.

After exhausting the first participant pool, we recruited with a survey (**Phase 2**). All Phase 2 participants created new recordings for our study. We distributed the survey in a public Slack channel and on Twitter under #TidyTuesday and #RStat hashtags. Recruiting continued until we gathered enough data to answer our research questions and observe similar patterns in participants' recordings [10]. Three participated among 24 survey respondents.

In total, six ($N = 6$) participants completed the study, their demographics summarized in Table 1. Phase 1 and Phase 2 participants may differ in their proficiency and motivation, but both groups met the inclusion criteria and contributed to the richness of the results. During interviews, participants self-reported their ggplot2 experience level (all intermediate to advanced). Phase 2 participants received $25 for making recordings for this study, and everyone received $25 for their respective interview.[8]

3.3 Recording task

Phase 1 participants (P1, P2, P3) recorded and posted their analysis process online without the knowledge of our study.[9] For Phase 2 participants (P4, P6, P7), we asked them to record a video as if they were creating a new #TidyTuesday submission while thinking out loud. We expected that the survey would reach people who understood what a #TidyTuesday submission entails—wrangling data and creating plots; all participants met our expectation and the inclusion criteria. Applicable to all participants, we did not specify which dataset to work on, what data questions to answer, or what visualizations to create. Prior work has found that task questions can affect participants' process and visualization choices [24], so by holding back specific instructions, we hoped to capture a wide variety of data analysis and visualization processes.

3.4 Retrospective interview

The first author conducted a retrospective, semi-structured interview with each participant via Zoom. We scheduled the interviews within 3.3 weeks on average after each participant's recording date. The time delay was for us to analyze the recordings and write targeted questions. According to Russell and Chi, 3.3 weeks is within the acceptable range of review delay [53]. At the beginning of each interview, we asked the participant to confirm their consent to be recorded (video of screenshare and audio).

There are two parts to each interview: 1) general questions to elicit experiences and opinions about ggplot2 and the Grammar of Graphics, and 2) targeted questions that asked participants to clarify and explain their decisions and analysis patterns. We encouraged participants to use their own words to describe analysis and visualization concepts. For the second part, we followed the *retrospective cued recall* protocol [53]. As we asked participants questions specific to each GIF or video, we showed slides with the visualizations and the corresponding R source code. The visuals served as memory cues to reconstruct the context of the recording quickly. Questions were ordered chronologically, in the order of the recording. For validation, we asked two recall questions per participant, for example, "Could you describe what you did next?". All participants answered correctly, if not immediately. Even if participants had blurred memory, the interviews still revealed what participants would have done in similar situations.

4 ANALYSIS

Combining the participant recordings and interviews, we used reflexive thematic analysis [8, 9] to find **RQ1** participants' conceptualizations and use of the Grammar of Graphics; and **RQ2** the interplay between visualization grammar and data transformation

[6] https://github.com/rfordatascience/tidytuesday#readme
[7] Russell and Chi provide an example review delay of 1-6 weeks [53]. With a maximum six-week delay in mind, we looked for videos recorded at most four weeks ago and factored in two weeks for scheduling the interview.
[8] P5 was interrupted during their recording and chose not to finish. We did not include P5 in the analysis, but they still received $25.
[9] Unlike other participants' videos, P1's recording was a GIF without think-aloud audio or a history of code edits. We reconstructed P1's code by using our ggplot2 knowledge and asking clarifying questions during the interview.

	Education	ggplot2 experience	Industry	Job title	Format	Phase
P1	Master	Intermediate	HigherEd/Gov	Data scientist	GIF	1
P2	PhD	Expert	Tech/Analytics	Data scientist	📹 (55 m.)	1
P3	PhD	Adv. intermediate	Software	Software engineer	📹 (37 m.)	1
P4	Bachelor	Fairly experienced	Financial services	Business analyst	📹 (57 m.)	2
P6	(PhD)	Experienced	HigherEd (biostat)	PhD student	📹 (66 m.)	2
P7	(Bachelor)	Intermediate	HigherEd (HCI)	Undergrad. student	📹 (100 m.)	2

Table 1: Participant demographics and the formats of their recordings. Education level in parentheses is the level each participant was working towards. ggplot2 experience was self-described during the interviews. 📹: a video recording of x minutes.

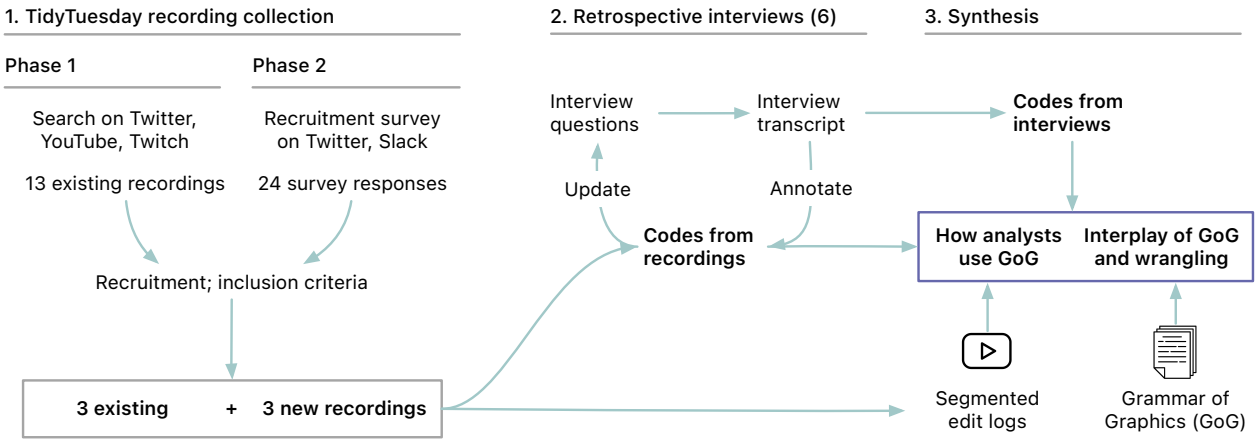


Figure 3: The three steps of our study and thematic analysis. Data, codes, and themes are bolded.

in a data project. Figure 3 summarizes our analysis process in the context of the study design.

4.1 Recording edit logs and segmentation

We derived codes and conceptualized themes from the #TidyTuesday recordings (Figure 3.2). First, we reconstructed the recordings through a log of code edits and other interactions, shown in Figure 5. The types of code edit are: data wrangling (dplyr functions [69]) edits, visualization (ggplot2) edits, console output, errors and interactions such as googling and reading the R documentation. For example, if a participant runs their code twice by adding an aesthetic mapping in their code and then changing its arguments, we count this process as two visualization edits. We reconstructed P1's analysis process in R based on the GIF keyframes and P1's final R script (which included all data wrangling). During the interview, we confirmed our reconstruction with P1. Since all participants did think-aloud during video recordings, we also selectively transcribed quotes when participants explained their edits.

We arranged each participant's edit log into *segments*. Similar to visualization construction cycles in Grammel *et al.* [24], we define each segment to capture how a participant created and refined a visualization or dataframe. A segment starts when a participant started a new analysis objective [4], a new plot (ggplot() call), or switched to a different dataframe or variable. The segment ends when the participant finished iterating on the plot or dataframe. Segmentation helps us structure our qualitative analysis, and it captures how individual plots are created—including the data wrangling that precedes the plot and the iterations on the plot design.

4.2 Thematic analysis

We used reflexive thematic analysis [8] to analyze our data. The first author derived the initial codes, either semantic or latent, from the edit logs and think-aloud quotes from the recordings, as well as the transcribed interviews. In Figure 4, for example, the first author assigned a semantic code to *describe* the filter() data edit P3 made. Latent codes are from when the first author *interpreted* participants' common behavioral patterns, mistakes, or conceptualizations, such as "confusion about how color mapping works: data vs. aesthetic space". We created 219 ($\mu = 36.50, sd = 20.34$ per participant) initial codes from the recordings and 422 ($\mu = 70.33, sd = 19.69$) from the interviews. To incorporate different perspectives on interpreting the data, the first author discussed code assignments with other authors, which included reviewing latent codes in the context of the raw data (*i.e.*, participants' R code and quotes).

Then, we combined codes from the recordings and interviews to collaboratively generate inductive and deductive themes in an affinity diagram. We intended the themes to capture "clusters of meaning" with a coherent narrative, as advised by Braun *et al.* [10]. In particular, inductive themes were bottom-up, from common meanings in the codes, such as "participants liked ggplot2 for the Tidyverse". Deductive themes were informed by theory: 1) the components of the Grammar of Graphics, and 2) the hypothesis

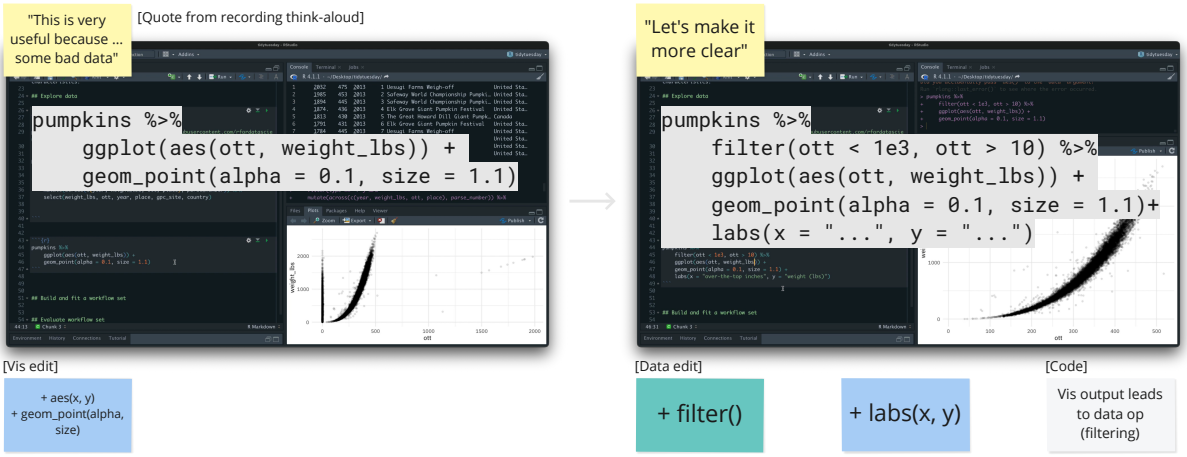


Figure 4: How we coded participant recordings on a virtual whiteboard, showing two consecutive screenshots from P3's RStudio IDE interface. Colored stickies notes contain think-aloud quotes, visualization or data edits, and a thematic analysis code.

that GoG helps iterative visualization designs. After two to three passes, all authors contributed to adding, removing, and finalizing the themes, which make up the titles in Section 5.

5 RESULTS

5.1 Overview of participant recordings

The six participants completed data projects with different foci. P1 made a lollipop chart (Figure 8), "data art" in their own words. P2, P3, and P6 recorded similar exploratory data analysis (EDA) processes where they explored the relationships among several variables through wrangling and plotting. P2 and P3 additionally tuned machine learning models. P4 and P7 each created a communicative visualization that showed one aspect of the data they explored. As an example of participants' analysis process, Figure 5 shows the edit log from P4 divided into segments (defined in Section 4.1). Other edit logs are available in Supplemental Materials.

To roughly assess how expressive participants' data and visualization specifications were, we tallied the average number of code edits across all participants with video recordings. We also contextualized the tally by analyzing a GitHub corpus of all R code files (R, Rmd, Qmd) containing the `#TidyTuesday` library import [41], with $N = 3649$ files from 975 unique contributors. Shown in Figure 6, our participants used roughly the same set of data wrangling functions as the broader code corpus. Our participants also used a variety of geometry (14) and scale (9) functions, while the code corpus contained even more unique geometries, themes, and statistical transformations as expected from the larger sample size. The average edit per file was higher in participants' edit logs; one explanation is that the code corpus contained the final versions of the code and did not capture addition, change, and deletion edits. Judging by the tallies, our participants' analysis code could be typical of the `#TidyTuesday` code corpus.

5.2 Execution: conceptualization of GoG components

Even though participants successfully created (executed) visualizations with `ggplot2`, their tasks and needs did not always directly correspond to how Grammar of Graphics components are designed.

5.2.1 Data space vs. aesthetic space. When participants customized their plots, they had difficulties specifying the *data component* in the Layered Grammar of Graphics [65], either making mistakes or finding customization tedious. Customizations included colors (P1), relative sizes (P1, P4), and locations of visual elements (P7).

To interpret participants' difficulties, we use the distinction of *data space* and *aesthetic space* [66, Chapter 15]. The *data space* contains input data with domain meanings, for example, "types of bee colony stressors". In contrast, the *aesthetic space* contains values describing aesthetics, such as the color hex value `#a62d3b`. Participants specified customizations in the data and aesthetic spaces in these ways:

- Hard-coding values in the *data space* (Figure 7.2). Participants hard-coded values to adjust the relative sizing of visual elements (P1), position annotations (P7), and shrink the size of dots to avoid overlaps (P4). With no support from the grammar, participants needed to guess what data space values could achieve their desired output. P1 expressed frustration with this "back-and-forth" process.
- (Mis)-using aesthetic space values in the *data space*. P1 wished to apply a custom color palette. When they first assigned hex values including `"#a62d3b"` to the color aesthetic, the color did not change because `"#a62d3b"` was treated as a value in the data space,[10] not a color in the aesthetic space, see Figure 7.3.
- Specifying *values* in the *aesthetic space* via a custom `scale` function. To fix the color palette problem above, P1 changed the mapping to `aes(color = year)` and introduced the color

[10] In `ggplot2`, categorical values in the data space all use the same default colors palette [66, Chapter 11.3]

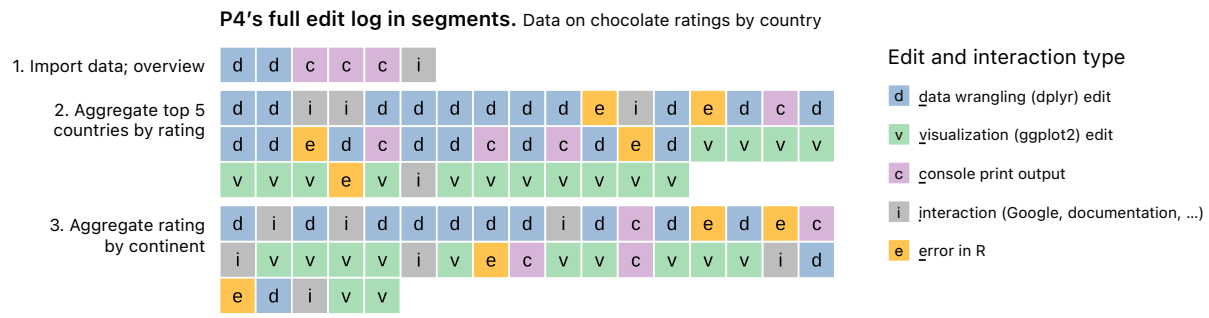


Figure 5: P4's full edit log from their `#TidyTuesday` video recording. One square is one edit or interaction. Squares are arranged from left to right in chronological order, grouped by segments.

Participant edit log

Edit type	Edit	Mean #/file
Data wrangling (dplyr)	group_by	6.8
	mutate	5.6
	filter	4.2
	select	2.8
	count	2.2
Top 6	summarize	2.0

#TidyTuesday corpus (N = 3649)

dplyr fn	Mean #/file
mutate	3.30
filter	3.30
group_by	1.57
select	1.14
count	0.96
arrange	0.71

Edit type	Edit	Mean #/file	# unique
Vis (ggplot2)	geom	8.0	14
	aes	4.6	-
	scale	2.2	9
	facet	1.8	3
	theme	1.8	4
	stat	1.6	2

ggplot fn	Mean #/file	# unique
geom	4.49	105
aes	4.44	-
theme	1.44	104
facet	0.65	13
scale	0.04	9
stat	0.04	23

Figure 6: Top data and visualization edit types by average counts per participant with video recordings. Variants of a grammar component, such as `geom_bar` and `geom_col`, are merged into the base function, in this case `geom`. We also provide average counts from a `#TidyTuesday` code corpus (N = 3649), which is static and does not capture code edits such as changing parameters.

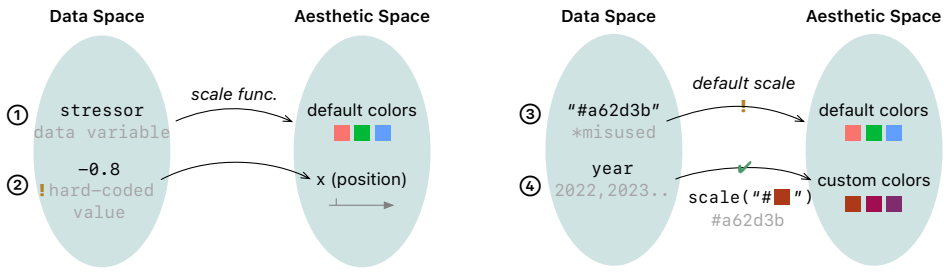


Figure 7: Data space and aesthetic space. 1: mapping a data variable `stressor` in data space to the color aesthetic; all participants specified such mapping without incident. 2: hard-coding values in the data space to place a text annotation (P7). 3: misusing aesthetic space value (hex code) in the data space (P1). 4: P1's fix to use a custom palette—the `year` variable is mapped onto the color aesthetic with a custom scale function, the *range* of which is the custom color palette.

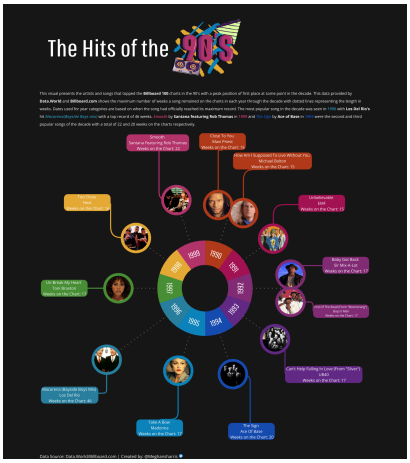


Figure 8: P1's final visualization as a lollipop chart.

values (in the aesthetic space) through a `scale()` function[11] (Figure 7.4). The `scale` function notes the custom mapping from color hex values to *year*, and transforms the *year* data into the corresponding hex values when the plot builds.

Compared to customizing plots, all participants specified within the *data space* by data column/variable names without incident. For example, P6 used `x = stressor` to map the stressor column name onto the *x*-axis as part of a bar chart, see Figure 7.1. We hypothesize that customization was challenging because it fell outside the GoG norm of mapping *names* in *data space* onto aesthetics. For customizations, participants' first intuition (*i.e.* using aesthetic space color value in an aesthetic mapping) might not comply with GoG rules. Since GoG is not designed for generating data to achieve custom plot appearance, participants needed to do extra work (*i.e.* using a custom `scale` function and "back-and-forth" hard-coding) to introduce customization data into the rest of the GoG abstraction. We discuss how a visualization grammar may better support customization in Section 6.

5.2.2 Aesthetic vs. *faceting.* Aesthetics (also called visual channels [55]) are visual properties of geometries that can vary with data, such as the x-axis position or color of a point. Participants chose aesthetics based on their tasks, which included comparing discrete categories and assessing correlations in the dataset. However, participants' tasks and approaches did not always align with the abstractions of GoG. With the task of comparisons, P3 considered color and faceting to be alternatives. When asked how they would explore more variables with the same plot, P3 thought both the color aesthetic and faceting[12] were options, calling them "effects" that can "highlight differences in that relationship, or relationship versus no relationship". What P3 described was the task of making comparisons. In GoG, it is natural to consider color *vs.* alpha as alternatives—they are both aesthetics while color *vs.* faceting are not. However, in terms of comparative visualization design,

[11] P1 used `scale_color_manual()`, see https://ggplot2.tidyverse.org/reference/scale_manual.html

[12] Faceting in `ggplot2` splits a plot into a small multiples by a variable [66].

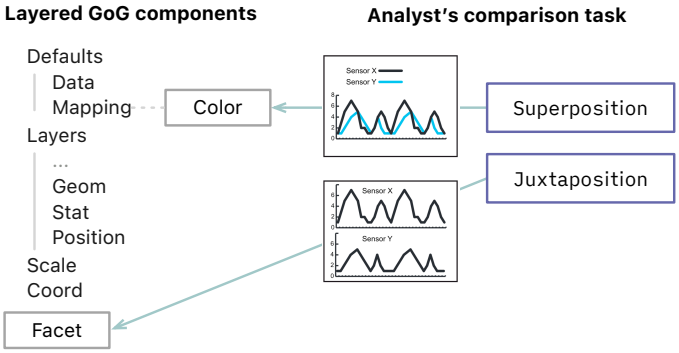


Figure 9: Correspondence between P3's comparison task and GoG components. Color (a type of aesthetic) and facet are not the same type of component in the Layered Grammar of Graphics [65]. Line charts are reproduced from Gleicher [23].

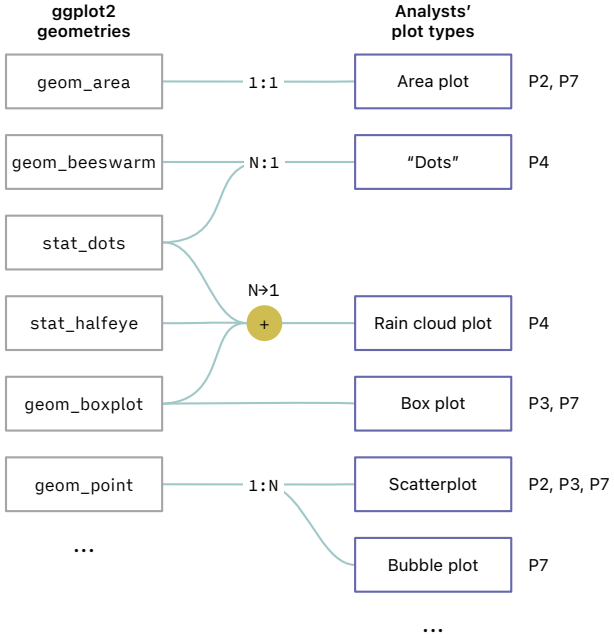


Figure 10: A subset of the mappings between geometries and the plot types described by participants.

faceting corresponds to "juxtaposition" and color aesthetic "superposition" [23]. Figure 9 shows our interpretation—P3's conceptualization matched the *task of comparison* instead of the organization of GoG components.

The mismatch between GoG abstractions and analysts' tasks does not necessarily mean that GoG components are ill-equipped for analysts in practice. However, making comparisons is one task for which the grammar design does not communicate task-appropriate alternatives; therefore, analysts need to translate their task to the choice of GoG components.

5.2.3 Geometry through plot types. Geometries are geometric shapes in visualizations, such as points and lines. Participants used geometries in code but sometimes talked about and made choices based on *plot types*, like box plots and line charts. Across participants, they used 18 different geometries in the `ggplot2` grammar[13] and mentioned nine plot types; the correspondence between plot types and geometries is often one-to-one (1:1) but not always, see Figure 10.

Since participants had both geometries and plot types in their vocabulary, we may ask which concept is closer to the participants' needs. Section 5.4 will discuss how geometries worked well as part of participants' iteration process, but when seeking help to find and choose components (geometries), participants preferred *plot types* with *visual* examples. For instance, P4 searched for "histogram" and "rain cloud plot", P6 looked up "United States map", and P7 visited gallery websites that listed by plot types. They all browsed pages with example plots. Reflecting on their search process, P6 found it helpful to "look for an example visualization that's similar to what I have an idea of", suggesting that they were motivated by the visual outcome of the plot. P4, P6, and P7 did not look up specific geometries, which could be due to a *discoverability* challenge—`ggplot2` grammar contains many geometries, and reading documentation without visuals can be "a steep learning curve" (P7).

Plot type as a concept is excluded from GoG because it can only express as many plots as there are type names [70, Chapter 1.1]. The GoG design is about combining different components like geometries and aesthetics to achieve expressiveness. Almost to the contrary, participants were motivated by achieving visual output—they expanded their use of geometries and increased expressiveness by looking up plot types. Bako *et al.* [2] also echo the significance of plot types in D3, a visualization toolkit without explicit notion of "graphical marks" (geometries) [7]. In Bako *et al.*, "standard" plot types such as bar and line charts made up the majority (80%) of their online code corpus, and users implemented a given plot type with similar specifications. It is hard to quantify whether using plot types limited our participants' expressiveness. Here we highlight the tension among the GoG design, participants' visually-oriented goals, and the discoverability of the grammar components.

5.3 Evaluation: silent errors

As participants wrangled and visualized data, they *evaluated* whether the analyses achieved what they intended. Most participants encountered R error messages that explicitly pointed out problems. In contrast, we also observed harder-to-evaluate *silent errors* [37], where problematic wrangling and visualization specifications successfully produced plausible outputs without explicit errors. Silent errors can be common in visualization specifications—Battle *et al.* found that D3 users also struggled with "unexpected behaviors rather than explicit errors" [3]. McNutt *et al.* named "silent and significant" errors *visualization mirages*, visualizations that can mislead inattentive readers about the data [37].

[13]This count includes `stat_dots` and `stat_halfeye`. Statistical transformation in `ggplot2` transforms the data (usually summarization). In `ggplot2` implementation, `stat_*` creates layers with the namesake statistical transformation and default geometries. We don't report `stat` as a separate category since participants didn't explicitly specify custom `stat` within their `geom` calls, and `stat_dots` and `stat_halfeye` were used in the same way as geometries.

We observed two types of silent errors. For the first type, participants *noticed* the error because the visual output did not meet their expectations. With different datasets, P6 and P7 wrangled and plotted to show the top-*n* categories (*e.g.* top-5) in their data over time. They caught the silent errors because the plots showed an excess number of categories, see Figure 11.6. We will explain their mismatched data wrangling and plot specifications in Section 5.5.2. To fix the errors, P6 aggregated data and P7 manually filtered out categories to reduce the number of categories shown. Their fixes were workarounds, presumably because identifying the root cause in the specification code was challenging; we discuss potential solutions in Section 6.2.

With the second type of silent errors, participants *did not notice* their plots were problematic at all. P6 and P7 created *visualization mirages* [37]— Table 2 shows how P6 and P7's final plots may give the readers the wrong impression of the underlying data, such as missing data or a perfectly linear trend. In these silent errors, the visualization grammar in part enabled visualization mirages. Some participants might have relied on "visual hints" (P7) to evaluate their analysis. Of a silent error that they did notice, P7 said:

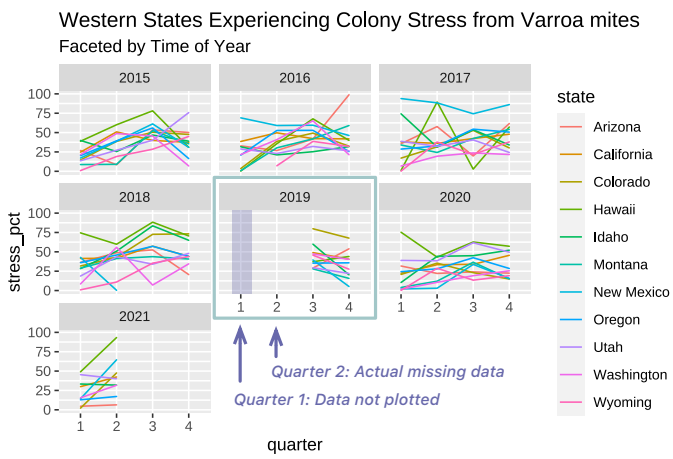
> [T]he only time I realize [the error] is when it shows up in this way because otherwise, I would not have realized what was wrong with the [data] processing...

Since the visualization grammar successfully assembled plots, the plots might have validated the analyses. Beyond evaluating analysis with the visualization outputs, participants had few strategies to notice data errors if the plots appeared plausible—P6 said that their collaborators could judge based on prior knowledge. Without such priors, which can happen during data exploration, analysts risk overlooking the silent errors when using visualization grammars.

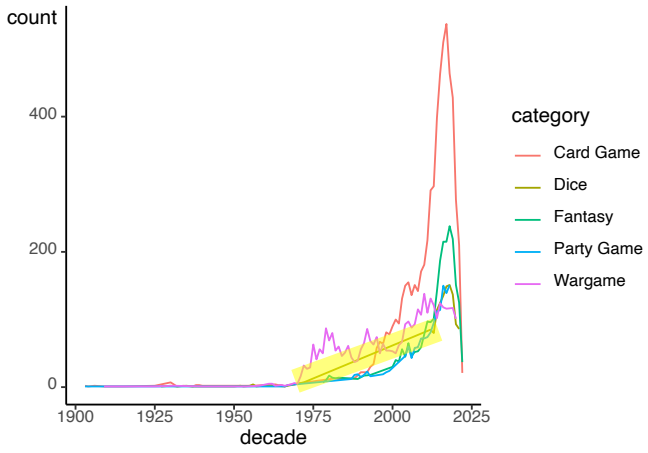
5.4 Execution-evaluation: Iterating with the Grammar of Graphics

To answer **RQ1** about how analysts conceptualize and use GoG, we consider not only stand-alone edits on GoG components (Section 5.2) but also a cycle of execution and evaluation—how and why participants wrangled data, specified visualizations, and evaluated their analyses as a process. According to Wickham, using a GoG-based library should enable users to "iteratively update a plot, changing a single feature at a time" [65]. We identified **incremental** and **experimental** iteration patterns, which were consistent with the design intention of GoG. On the other hand, participants also reused GoG specifications as **plot templates**, showing how participants circumvented specifying and iterating GoG components directly and still achieved their analytical tasks.

5.4.1 Incremental and experimental edits during iteration. We observed an **incremental** pattern in editing aesthetics, geometries, and plot types. For aesthetic edits, P2 and P3 followed a similar editing pattern: they created scatterplots with x and y aesthetics first and then added color or alpha aesthetic for clarity. After each edit, P2 and P3 rendered the resulting plot to evaluate. For geometry edits, P2 and P3 layered more geometries on their scatterplots for visual inference (P2 added a linear fit line, P3 splines). P4 had an analogy for the layering of geometries: "I feel like they're Photoshop layers". To change from one plot type to the next, P2 made two



P6's final plot implies that there were no data for the first quarter in 2019 (highlighted), but only second quarter data were missing. First quarter data were not plotted because a line needs two endpoints.



P7's final plot without customizations. Highlighted: the "dice" category had count of zero between 1972 and 2012. The straight line erroneously implies that there is an upward trend.

Table 2: P6 and P7's final plots showed visualization mirages.

edits to turn a bar chart into a scatterplot: edits to the y aesthetic mapping and geometry, see Figure 12. In P2's case, they said during the recording that they wanted to see the correlation between two variables with a "scatterplot", in addition to the initial bar chart that only tallied one variable. Their two edits saved them the trouble of writing a scatterplot from scratch. Even though P2 reasoned with plot types ("scatterplot") that are not part of the GoG, they iterated as GoG was designed for.

These edits above were *incremental* in that the edits built up to the final plots. In comparison, some edits are **experimental**—they were part of the *exploration* and not reflected in the final plot. P1 often took some geometries off to focus on one part of the plot, only to put them back soon after. P7 experimented with geometries, swapping five (line, area, point, boxplot, density_ridges) in and out while keeping the aesthetic mappings the same, see Figure 12. They iterated through so many geometries because they wanted to affirm their choice of geometry/plot type. Commenting on their trying out the boxplot, P7 said, "now I know [the boxplot] makes no sense and I need to see that". In addition to editing plot designs, experimental edits can also help understand data. To see how rankings change for top songs over time, P2 started with a line chart for the top-ranked song and then used faceting to display two and then nine top songs. Their use of faceting supported their need to understand the data through a small number of examples and concrete thinking. Even though the experimental edits were eventually overwritten, they helped participants design plots and understand data concretely.

5.4.2 GoG anti-pattern: plot templates. Participants did not always make incremental and experimental edits (Section 5.4.1) enabled by GoG design. We define **plot templates** as code chunks participants reused, which could include both visualization and data wrangling specifications. Participants (P2, P4, P6, P7) copied plot templates and added a few edits to create plots similar to what they had made or found online. Despite the convenience, templates were not foolproof. P2 copied and pasted a template with a bar chart, only to miss one variable when updating the template. They quickly fixed the error because they noticed that the bars were unexpectedly out of order. P2 was not satisfied with using copying and pasting because "it [needing to change multiple things] pulls me farther from the data". As a solution, P2 said that they would encapsulate what we call templates with convenience functions. Explaining why they used templates, P6 said that "it's just easier to use code that I already know what's going to happen". In our interpretation, plot templates encapsulate a unit of analysis and apply **plot types** from Section 5.2.3 to new situations.

5.5 Execution-evaluation: GoG and data wrangling

To answer **RQ2** about how GoG works alongside data wrangling, we expand the execution-evaluation perspective to include data wrangling edits. As participants interleaved their data wrangling and visualization in R Markdown notebooks, we found that visualization specification and data wrangling form a **feedback loop**, a flexible process where one informs the other. There was also a **tight coupling** between visualization and wrangling code, which can explain the silent errors in Section 5.3.

5.5.1 Vis-analysis feedback loop. In a feedback loop between data wrangling and visualization specifications, we found that 1) the results of data wrangling informed participants' decisions on plotting, and 2) the results of plotting helped participants progress in data wrangling.

Participants used their understanding of the data to determine how to visualize them. With data wrangling code written, participants often printed the results to console and gained an understanding of the properties of their data, such as data types, dimensions,

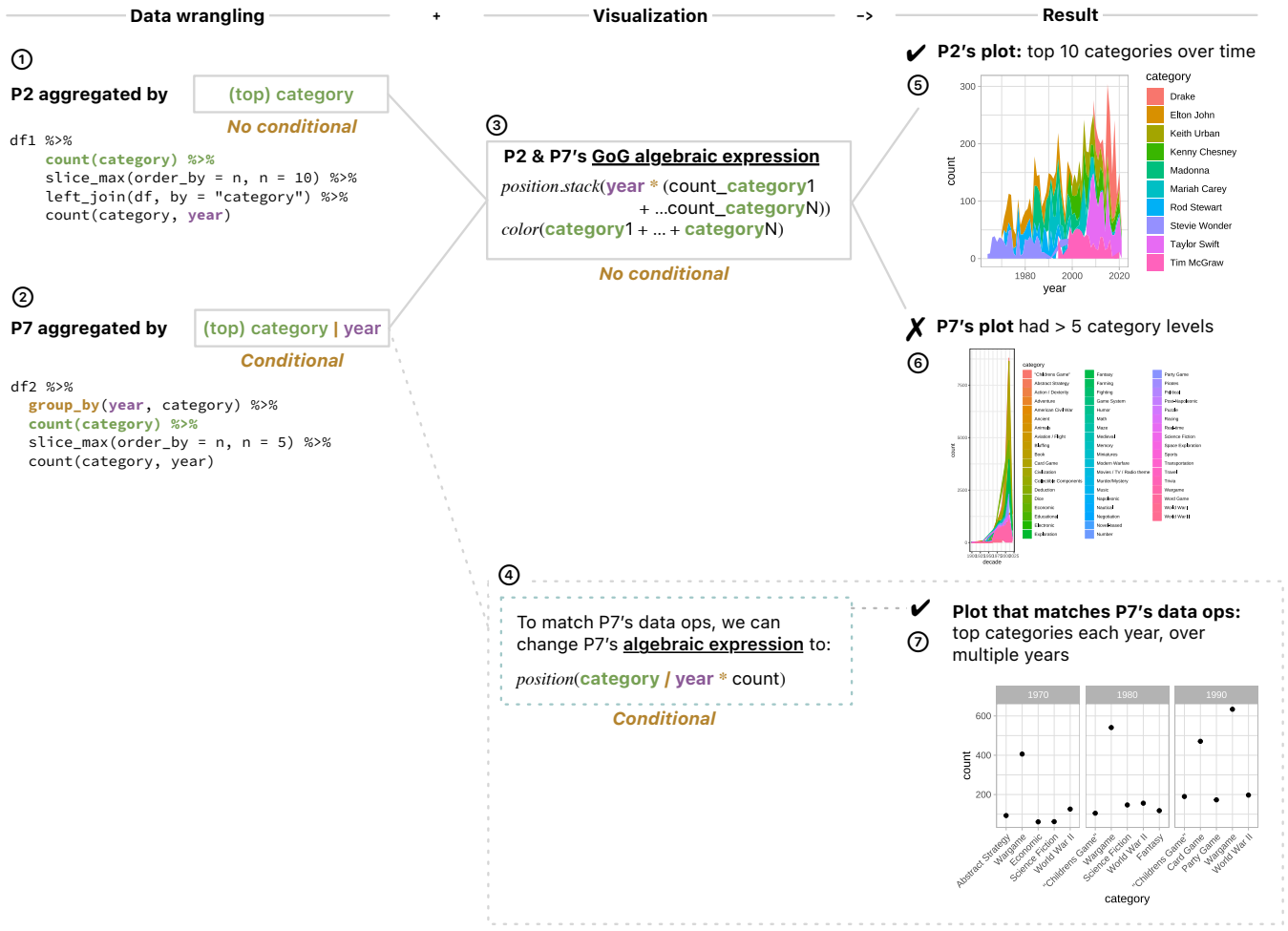


Figure 11: How P2 and P7's data wrangling matched or not matched the visualizations they created. In 7, we only show three years in the facets for simplicity.

and simple aggregates like counts and unique values. As an example of data informing visualization specifications, P2 avoided mapping a variable for song names onto the color aesthetic because P2 "already knew there were hundreds of songs". In another instance, P4 rejected the beeswarm plot they saw on a tutorial because they found that the data variable had "very specific levels" (being discrete) through data aggregation functions. Given their understanding of the data, P4 reasoned that the layout of the beeswarm (jittered points implying continuous values) "would be misleading".

In the other direction, participants decided what to change about the data once they saw a plot output. Participants removed outliers (P3) and filtered to top entries (P2, P3, P4, P6, P7) in response to seeing their plot outputs. P4 reasoned that "nobody would understand" too many levels of categorical data in a plot. Figure 13 shows an example from P2's edit log, where they added `filter()` function based on visualization output.

The feedback loop was *productive* in that participants made data-informed visualizations and visualization-informed wrangling. Reflecting on the role of GoG, we highlight the GoG's capacity to add and edit modular components and still produce a visualization. The data wrangling participants wrote was in the form of analysis pipelines made up of modular `dplyr` data operations such as `filter()` [69]. Though not its express design goal, the modularity of GoG components worked flexibly with data wrangling and thus contributed to the feedback loop.

5.5.2 Tight coupling. In some workflows, data wrangling is done separately before visualization specification [30]. For example, P1 described a process at work, where they exported data from R into Tableau to create plots. In a programming setting with visualization grammar, however, visualization and data wrangling specifications can interleave and express the same meanings such as data aggregation. Therefore, the two parts of the specifications need to be consistent to prevent errors, a constraint we call **tight coupling**.

The constraints from tight coupling are in contrast to the flexibility and productivity of the feedback loop in Section 5.5.1. Since GoG offers options to express data transformation *within* the GoG specification, participants needed to keep visualization and data

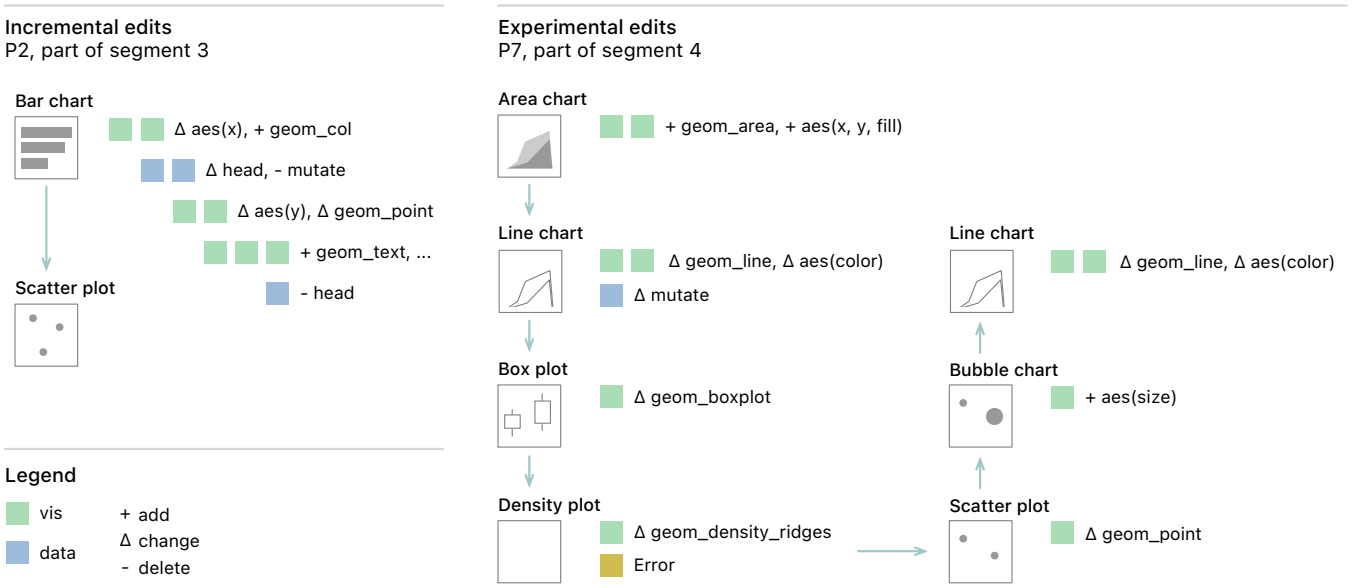


Figure 12: Two excerpts from the edit logs to demonstrate incremental and experimental edits.

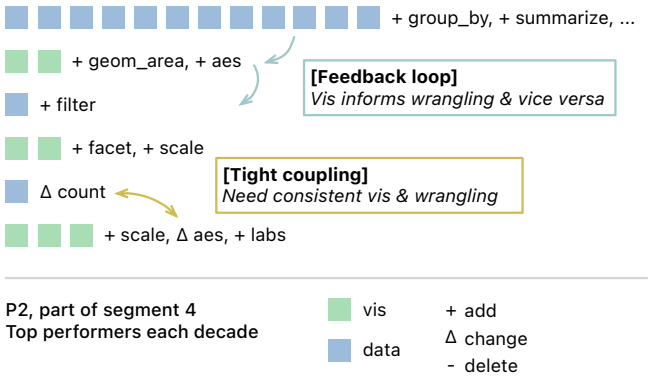


Figure 13: Excerpt from P2's edits, demonstrating the feedback loop and tight coupling between visualization and data wrangling. The full edit log is in Supplementary Materials.

wrangling specifications consistent. As an example (Figure 13), P2 intended to smooth out a count over time, changing data aggregation (`count()`) from "by year" to "by every five years". Executing their code, P2 produced an area plot showing counts inflated about five times. To keep the visualization specification consistent with the updated data wrangling, P2 needed to change the data variable n to $n/5$ in the aesthetic mapping so that the plot still showed annual counts after smoothing.

Breaking the coupling and having inconsistent specifications can lead to the silent errors described in Section 5.3. To explain, we introduce the fact that GoG [70, Chapter 5] and the data wrangling tool (`dplyr` [69]) share the same roots in relational algebra. Since analysts can edit GoG and data wrangling separately (feedback loop), the underlying algebra representation can go out of sync. We compare P2 and P7's code and plots in Figure 11.[14] `ggplot2` specifications were translated into GoG algebra expressions [70, Chapter 5]. Figure 11-1 shows P2's code, which found the top categories by count, and counted the (top) categories for each year. In comparison, P7 `group_by()` the *year* variable first, making category count *conditional* on *year* (Figure 11-2). There can be different top categories for each year. We argue that P2's visualization matched their wrangling but P7's did not. Figure 11-3 shows that the visualization specification contains a cross operator ($*$) in the GoG algebra, implying *year* and *catgory* being independent, which was the case with their wrangling code. P7's visualization algebra expression is the same, but their *category* is conditional on *year*, and therefore P7's visualization did not match their analysis. In other words, P7's visualization did not convey the *conditional* from the analysis. To synchronize P7's visualization with analysis, Figure 11-4 shows one potential visualization specification, using a nesting operator (/) that expresses the meaning of conditional. In this case, the nesting operator translates to faceting by *year*, and each facet only contains the top categories for that year.

Participants needed to negotiate and disentangle the tight coupling, since the visualization and data wrangling specifications cannot communicate with each other and maintain consistency, a problem Wu *et al.* call the *semantic gap* [77]. The semantic gap, in our context, means that the wrangling code does not know how the data is plotted, and the visualization specification has no access to the wrangling history of the data. But successfully maintained or not, tight coupling is not immediately visible to participants, therefore leading to silent errors. We discuss how to bring more

[14] For ease of reading, we translated P2 an P7's `fct_lump()` calls into data operation functions (verbs) that are closer to SQL syntax. `fct_lump()` "lumps all [factor/categorical variable] levels"; documentation: https://forcats.tidyverse.org/reference/fct_lump.html. SQL translations determined by the `dbplyr` package, see https://dbplyr.tidyverse.org/articles/sql-translation.html

consistency and transparency to coordinate visualization and data wrangling specifications in Section 6.2.2.

5.5.3 Participants liked ggplot2 for the Tidyverse. Since not all participants would know visualization grammars (the framing of our analysis), we directly asked them about preferences for the `ggplot2` library. Participants' answers were as much about `ggplot2`'s integration with data analysis as about the quality of visualizations it creates. Since the participants were likely biased, they found `ggplot2` "intuitive" (P2), "makes so much sense" (P3), customizable (P1), layer-able (P4), and nice-looking (P6). Despite the praise for `ggplot2` as a visualization library, we were struck by how much participants (P1, P2, P6, P7) talked about the synergy between ggplot and their data analysis, mostly enabled by the `dplyr` [69] and Tidyverse [68] packages.[15] P2 told the story of a "big realization", where they realized the `plyr` package (predecessor of `dplyr`) and `ggplot2` were linked by the tidy data format [68]. The convenience of transforming data and plotting made P2 switch to using `ggplot2`. Other participants gave similar reasons: `ggplot2` fit into P6's job with "a lot of data cleaning and preparation", and it "integrates well" with P6's projects. P1 thought `ggplot2` gave them more control of the data. Participants' subjective preferences for `ggplot2` corroborate our findings of the vis-analysis feedback loop in Section 5.5.1.

6 DISCUSSION AND DESIGN IMPLICATIONS

6.1 Practical visualization grammars for data analysts

6.1.1 Supporting plot templates. At its core, `ggplot2` is an implementation of the layered GoG without explicit components for plot types [65]. However, there can be a disconnect between the motivation for the grammar design and how it is used. Our participants talked about visualizations in terms of plot types (Section 5.2.3), and they were also able to code-switch and edit the GoG abstractions to achieve their goals such as copying a plot or changing plot types. In effect, participants encapsulated their both visualization and data wrangling specifications into plot templates. Our findings suggest that if plot templates are based on a visualization grammar, data analysts can take advantage of the plot templates as well as the grammar itself (Section 5.4.2).

Interests exist in creating plot templates for analysts based on GoG-style grammars. The `ggplot2` API already implements convenience geometries, such as `geom_jitter`, a shortcut for `geom_point` with jittered positions. Outside of `ggplot2`, Ivy and Encodable are two recent template-based contributions. Ivy is a visualization editor that can create a plot template from any visualization grammars based on JSON [39]. However, the Ivy interface is Polestar/Tableau-like [60, 73] and analysts cannot encapsulate data wrangling in Ivy templates, except data transformations within Vega-Lite. Encodable is a configurable grammar for plot "components" (*i.e.* templates) [71] with user-defined "component-specific channels" (*i.e.* aesthetics). Compared to Encodable templates, the plot templates our participants used are more informal, with no error checks, and they are rendered only with `ggplot2`. We propose a design wishlist for supporting plot templates in a data analysis environment, covering definition, presentation, and use:

- We need more research on how to define or generate a plot template in a programming/computational notebook environment. Our participants only made informal plot templates by copying and pasting code chunks. Bako *et al.* suggests that to help D3 users, templates could be extracted from users' implementation patterns from an existing corpus [2]. Lee *et al.* considered boilerplate template code cumbersome, motivating them to build an always-on visualization recommender in Jupyter notebooks [33]. Regardless of how plot templates should be extracted or introduced, they should integrate well with the rest of the analysis workflow, which Battle *et al.* recommend in the context of D3 implementation challenges [3].
- Based on how our participants sought templates online, plot templates should be visual, meaning the analysts can preview a concrete instance of a template. Battle *et al.* also outlined the importance of "meaningful code components" (in D3)—users may otherwise be unable to relate part of a template to the visual output [3].
- Plot templates should be editable—analysts may benefit from code-switching and moving between editing the abstractions of GoG and using plot templates. Templates should also reduce the opportunities for slips and mistakes during edits.

6.1.2 Supporting customizations in the aesthetic space. Participants customized their plots by creating new data, building special plot types, applying custom color palettes, and adding annotations. Beyond our study, literature and anecdotal evidence also support the need for customizations. In a study about Tableau, Heer *et al.* found that "formatting" (changing sizes and styling) was a common category of actions "performed in succession" in interaction logs [27]. Hadley Wickham, the creator of `ggplot2`, stated that "theming is unimportant early on [in the development of ggplot2] but critical in the long run" [49].

To systematically describe customization in GoG, we used the concepts of data space *vs.* aesthetic space. Grammar of Graphics uses *scale functions* to map from the data space (domain) to the aesthetic space (range). Therefore, changes in the data space, like filtering or binning, will be updated to the aesthetic space.

The GoG design leads to two types of difficulties around customization we observed. First, analysts can make conceptual slips, confusing aesthetic space specification as one in the data space (*e.g.* P1 with the custom color palette). Granted, to customize colors and other aesthetics, analysts can use the identity scale function[16] or a scale function with a custom color range as P1 did (see Figure 16 in the Appendix for code in `ggplot2` and Vega-Lite). We speculate that the slips are possible because analysts might expect the aesthetic space values (like color hex values) to just work like data space values. Another reason might be that the scale functions are usually not explicitly specified—many visualization libraries, `ggplot2` and Vega-Lite included, provide good defaults and therefore analysts might not be conceptually familiar with data *vs.* aesthetic spaces

[15] `dplyr` is part of the Tidyverse collection of packages that "share an underlying design philosophy, grammar, and data structures", see http://www.tidyverse.org/.

[16] Available in `ggplot2` as `I()` and in Vega-Lite as `.scale(null)`

and scale functions. Addressing such difficulty with customization may entail more transparency into visualization library internals—if analysts can peak into the data being plotted before and after the scale computation, they may have a better mental model to identify the scale function they need.

Second, since there are no inverse scale functions to map aesthetic space values back to the data space, participants needed to make "back-and-forth" edits in the data space to achieve the desired appearance of a plot with the abstractions of GoG alone. For example, if an analyst wants to add a text label in the bottom left corner of a plot, they need to translate "bottom left" into values in the data space, *e.g.*, `year = 1900`, see Figure 15 in the Appendix. Worst still, if the analyst changes the data space values (like by filtering), the analyst needs to manually update label positions, *e.g.*, `year = 1950` to keep the label in the same relative position. Without an automatic two-way mapping *between* the data space and aesthetic space, it can be tedious to update in the data space and keep a plot looking the way the analyst wants.

Outside GoG, there are escape hatches in visualization libraries that allow analysts to specify in the aesthetic space directly. In the R language, the low-level graphics library `grid` exposes physical positions like inches and "native units", which are relative positions between 0 and 1. For example, $(0, 0)$ is the coordinate for the "bottom left" corner of a viewport [46, Chapter 4.5]. However, analysts cannot access native units in `ggplot2` unless they program custom components. Similarly, Vega-Lite supports specification by pixel positions[17] and relative height/width (*e.g.*, `0.1 * width`), but these are arguments passed to marks, not part of the GoG abstractions.

We need systematic and practical solutions to help data analysts customize grammar-based plots. Analogous to the aesthetic mappings (mapping data variables to aesthetics), we can design similar APIs to support translating aesthetic space definitions to data space values. Such mapping should alleviate analysts' burden of trial and error, *e.g.* P1's back-and-forth edits. One step towards this goal, `ggrepel` is a `ggplot2` extension that detects collision of text labels and repel ones that overlap.[18] With `ggrepel`, the aesthetic space instruction is effectively "no overlap". Outside of `ggplot2`, we may borrow ideas from authoring systems for custom charts, where chart designers could start from visual marks or hand-drawn shapes in the aesthetic space. Then, the data space information can be bound to those marks in stand-alone systems such as Lyra [54], Data Illustrator [34] and Charticulator [51]. Hempel *et al.* used direct manipulation of graphical output to augment text-based programming and specify a recursive fractal pattern [28]. In a programming environment such as a computational notebook, direct manipulation of marks might not be feasible, but it is worth investigating how analysts can specify in the aesthetic space alongside the GoG abstractions while keeping the aesthetic space and data space in sync.

6.2 Helping analysts evaluate data wrangling

Visualization can be viewed as a type of data transformation [26]. Participants made silent errors and failed to identify the problems in their analyses because of the *tight coupling* between data wrangling and visualization specifications (Section 5.5.2). Here we propose two areas of research that may help analysts *evaluate* the results of data wrangling:

6.2.1 Maintaining the consistency between visualization and data wrangling specifications. In our study, silent errors could have been avoided if analysts maintained the consistency between the data wrangling and plot specifications (Section 5.5.2). Instead of solely relying on analysts, we suggest that visualization grammars validate the consistency as well. Grammar of Graphics exposes two ways to wrangle data within a specification: the *data* and *statistics* components. Depending on the implementation, visualization grammars support data wrangling with various degrees of expressiveness: `ggplot2` allows arbitrary R functions, while Vega-Lite supports a predefined list that covers common wrangling operations,[19] such as pivots and joins (see Table 3 in the Appendix). With Vega-Lite's pre-defined list of operations, for example, it is possible to have a *linter* in a grammar implementation to check for consistency. With examples in Vega-Lite, McNutt *et al.* have proposed a "metamorphic visualization linter"; this linter verifies that data changes reflects proportionally in visualization output [37]. By varying the input data, such a linter may notice that the highlighted line from P7 in Table 2 does not change as expected. Alternatively, a linter can leverage and verify using the shared algebraic representation between data transformation and visualization as seen in recent works in databases and visualization [16, 35, 76, 77]. Ideally, a consistency-aware implementation can identify and explain the mismatch to analysts; VizLinter, a Vega-Lite linter that checks for problems in encodings and marks, implements similar explanations [14].

An alternative approach is to introduce *atomic primitives* for specific tasks. In visualization grammars, such primitives can replace otherwise multi-step, error-prone operations and reduce opportunities for errors. For example, the Probabilistic Grammar of Graphics makes probability expressions, such as $P(A|B)$, primitives [48]. This design avoids the pre-computation of conditional probabilities and circumvents errors in plot specification. Another example is `autoplot()`, a generic function in `ggplot2` used by P2 and P3. When called on a model tuning object, `autoplot()` dispatches the corresponding method to produce a `ggplot2` object. Since the plot is determined from the modeling workflow without analyst input, there is little concern for consistency. Compared to the algebraic approach, however, creating new primitives is less generalizable and hinges on understanding analysts' particular tasks and goals.

6.2.2 Increasing transparency and visibility. Visualization grammars should support the interpretation and critical thinking around data analyses. In our study, making plots is a major way that the participants understood their wrangled data. When our participants did not notice silent errors in their plots or apply their prior knowledge, it was difficult to recognize problems in the data manipulation. McNutt *et al.* has suggested tests to catch silent errors from plot specification and data wrangling [37]. Besides error detection, visualization tools may help analysts interpret analysis pipelines. Recent studies have taken a variety of approaches, such as explaining the analysis pipelines with animations (Datamations [50]), comparing

[17] For example, `vl.markText("x": 5)`, 5 is in pixels. With Vega-Lite Javascript API https://vega.github.io/vega-lite-api/.

[18] https://ggrepel.slowkow.com

[19] https://vega.github.io/vega-lite/docs/transform.html

results of alternative data manipulations (DITL [63]) and showing thumbnails of variable distributions inline with code [29].

Trust is involved in evaluating a visualization output. The fact that a plot rendered without error might give analysts more trust in the output, validating the analysis in general. We do not yet have good evidence for this form of trust (*cf.* [18]), but more transparency and visibility, especially in visualization format, may give analysts more opportunities to spot problematic analysis.

6.3 Degree of vis-analysis integration

We argue that to understand a visualization grammar, we need to look at where it's used—in data analysis that includes not only wrangling but also modeling. We can imagine a continuum of how well a visualization grammar integrates with data analysis, see Figure 14. On it, there are visualization grammars and systems for a specific task, such as CAST for creating animations [21]. They are low on data-analysis integration, assuming cleaned data as input. For Tableau-like[20] systems, usually for exploratory visual analysis (EVA), there are fixed set of data operations (filtering, *etc.*) enabled through interface widgets; analysts are known to copy and paste results from notebooks into Tableau-like interfaces [13], presumably because some analyses are not well-supported within such interfaces. On the far end, there is the R Tidyverse (`ggplot2` included) ecosystem, which our participants used to program their analyses. Similar ecosystems exist in other languages. For example, JavaScript ecosystem contains JSON-based visualization grammars [38], data libraries like Arquero,[21] and Observable notebooks. Granted, the visualization grammars and systems on this continuum serve various purposes, and not every analyst can or need to specify arbitrary analysis through programming.

Our focus is on the right end of the continuum: we need more research into how to support visualization and analysis specifications in computational notebooks, an environment that can afford a high degree of vis-analysis integration. There is a substantial need for the vis-analysis integration [19, 30]—data analysts could find it desirable to rapidly explore, transform, model, and visualize data in the same programming environment. A computational notebook environment (R Markdown) gave our participants full access to the analysis capacities in R and led to our findings on the feedback loop and tight coupling between visualization and wrangling specifications. With notebooks getting wider adoption [52], this environment can give analysts more agency and write expressive code for analysis and plotting. One implication can be that analysts do not have to rely on visualization designers to create task abstractions (*e.g.* [11]) and monolithic systems. In other words, a notebook environment with vis-analysis integration can be an opportunity to introduce visualization research into practical use [5].

On the flip side, analysts who write code can make mistakes, and they can use support. With a relatively small set of notebook-related contributions in visualization (*e.g.* litvis [75]), we can look to the data science literature. For example, there is the idea of a fluid interface, where interacting with a table (mage [31]) or visualization GUI (B2 [77]) can turn into an update in the analysis code.[22] There is the idea of sticky notes, where the analyst can drag a notebook cell onto a dashboard like a sticky note [64]. More recently, the integration between visualization and analysis has been made more explicit. Data analysis provenance can now inform visualization recommendations within the notebook [20, 33], instead of similarity between visualization specifications alone (*e.g.* [79]). We may borrow the different modes of interactions and integration, which consider writing code and interacting with the GUI simultaneously. We are excited about the outlook in this area of research: what can visualization contributions look like in computational notebooks or other environments with this high degree of visualization-analysis integration? And what can we learn and enable analysts to do?

6.4 Limitations and generalizability

Our study participants can be biased in favor of `ggplot2` because of the recruitment protocol—they were within the `#TidyTuesday` community. However, the positive bias is necessary for participants to be proficient enough to complete data wrangling and plotting tasks, because we were explicitly interested in experienced analysts, not novices. Another issue is whether participants' use of `ggplot2` was realistic. It was realistic in that participants were data professionals or students and used `ggplot2` in context of their `#TidyTuesday` projects. When asked, participants stated that their goals for participating `#TidyTuesday` were to hone skills or help others, corroborating the findings in Shrestha *et al.* [58]. On the other hand, participants were motivated by their interests instead of specific domain question when analyzing the `#TidyTuesday` data. Their approaches might differ with problems they had more prior knowledge in. We mitigated this concern by asking about participants' general preferences and approaches during interviews.

6.4.1 Generalizing to Vega-Lite. Since all our participants created visualizations with `ggplot2`, we assess the generalizability of our findings in the context another GoG-based grammar, Vega-Lite [55]. Vega-Lite is similarly popular in practice (1.1M monthly downloads *vs.* `ggplot2`'s 2.9M[23]), and there is a vibrant research ecosystem around Vega-Lite [38, Figure 8].

We can assume generalizability from the close correspondence between `ggplot2` and Vega-Lite syntax and Grammar of Graphics concepts, shown in Figure 2. We frame our results in Section 5 around the shared GoG concepts, and what our participants did in `ggplot2` can be closely replicated in Vega-Lite (code examples in the Appendix). However, replicability does not necessarily translate to an identical usage pattern.

We speculate that how a visualization grammar handles data transformation can affect how analysts use it. Though `ggplot2` and Vega-Lite enable data transformation through the same set of GoG components (Table 3 in the Appendix), Vega-Lite supports pre-defined data operations, more limited compared to `ggplot2`.

[20] The Tableau commercial software can pull results from R and Python scripts through client-server connections, but this setup still separates the scripting and visualization https://www.tableau.com/about/blog/2013/10/tableau-81-and-r-25327

[21] A relation algebra-based data querying and transformation library, see uwdata.github.io/arquero

[22] Commercial notebook tools such as Hex and Deep Note have supported specifying Vega/Vega-Lite with no-code interfaces; though users can export or update Vega/Vega-Lite specifications as code, chart cells do not synchronize with analyses outside of visualization specifications. See https://learn.hex.tech/docs/logic-cell-types/display-cells/chart-cells and https://deepnote.com/docs/chart-blocks

[23] Vega-Lite download count on Node Package Manager (npm): https://npm-stat.com/charts.html?package=vega-lite&from=2022-10-28&to=2022-11-28; `ggplot2` download count on the Comprehensive R Archive Network (CRAN): https://cranlogs.r-pkg.org. As of November 2022.

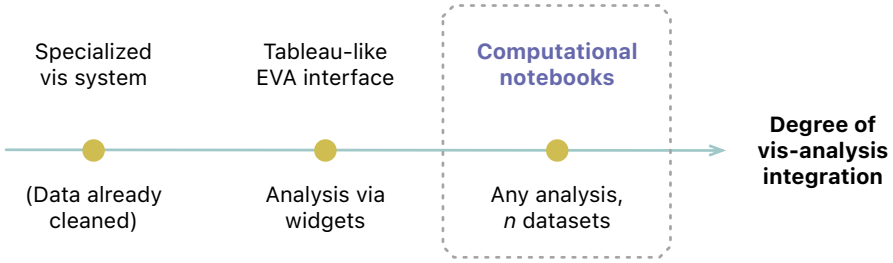


Figure 14: Continuum of vis-analysis integration for visualization tools. Towards the right end, computational notebooks, with visualization grammars and data wrangling libraries, offer a high degree of integration.

Vega-Lite's transformations may be sufficient for many use cases, or analysts may be more inclined to wrangle data with other tools before specifying plots, changing the tight coupling described in Section 5.5.2. In addition, interactions, such as selecting and brushing on a visualization, can be considered data transformations [77], but we did not observe such interactions with our participants. Vega-Lite supports a wide range of interactions *natively* [55], while `ggplot2` visualizations can *become* interactive with Shiny apps or Plotly [59]. With Vega-Lite, analysts might be more likely to replace a part of their data analysis code with interactions such as brushing and selecting, adding new dimensions to our findings.

Beyond the design choices of visualization grammars, we should consider the grammar ecosystems. First, the *extensibility* of a visualization grammar can determine how *expressive* analysts can be with plotting. Our participants (P4 and P7) used `ggplot2` components from community contributors,[24] not part of the core library. `ggplot2` is extensible because it uses an prototype-based system [67]; user-created components (custom geometries, scales, *etc.*) are plain functions, compatible with the rest of `ggplot2`. In comparison, Vega-Lite has no similar user-facing extension mechanism. Extending Vega, what Vega-Lite synthesizes to [55], can lead to specifications not reusable by other people.[25] If our participants had used Vega-Lite, their visualizations could have been different. Second, factors other than language features can influence how analysts use visualization grammars. A programming languages survey has suggested that "domain specialization" and "developer experience" are significant factors in language adoption [40]. Analysts often use `ggplot2` in tandem with statistical modeling packages in R, while Vega-Lite often works with Javascript or Python (via the Altair API [62]). A future study with Vega-Lite/Altair may help answer whether analysts have different analytical tasks or norms beyond the R ecosystem.

7 CONCLUSION

We conducted a qualitative study to understand how data analysts conceptualize and use a GoG-based visualization grammar (*i.e.* `ggplot2`), and to characterize how the grammar works alongside data wrangling specifications. Our participants were intermediate to advanced analysts who recorded how they completed `#TidyTuesday` data projects involving data wrangling and `ggplot2` visualizations in R Markdown, without prescribed tasks. We found that when participants created (*executed*) visualizations, their needs for analytical tasks and customization directed their use of GoG components. Despite specifying valid plots, participants sometimes made hard-to-*evaluate* silent errors. Viewing the analysis process as an execution-evaluation loop, we identified incremental and experimental visualization iteration patterns consistent with GoG design intentions. Between visualization specification and data wrangling, we found a feedback loop that *informs* iteration, while the tight coupling between visualization and wrangling *constrains* it. Based on our findings, we discuss design implications for future visualization grammars used in computational notebooks, a programming environment we believe can facilitate vis-analysis integration. Our recommendations focus on making the grammar more practical for analysts by incorporating plot types and other customizations, in addition to helping analysts maintain consistency between visualization and data wrangling specifications.

[24]The components were `geom_density_ridges` for ridgeline plots and `stat_dots` for dotplots. A gallery of `ggplot2` extensions: https://exts.ggplot2.tidyverse.org/gallery/
[25]Explained in Vega Documentation: https://vega.github.io/vega-lite/ecosystem.html

ACKNOWLEDGMENTS

We thank Eytan Adar, Mark Guzdial, Cyrus Omar, Priti Shah, Arvind Satyanarayan, and the organizers of VIS 2021 Doctoral Colloquium for general feedback, and Gabi Marcu for advice on qualitative methods. This project is funded by the National Science Foundation, Award Number 1910431.

REFERENCES

[1] D Abowd. 1991. Formal Aspects of Human-Computer Interaction. (June 1991), 237.
[2] Hannah Bako, Alisha Varma, Anuoluwapo Faboro, Mahreen Haider, Favour Nerrise, Bissaka Kenah, and Leilani Battle. 2022. Streamlining Visualization Authoring in D3 Through User-Driven Templates. , 16–20 pages.
[3] Leilani Battle, Danni Feng, and Kelli Webber. 2022. Exploring D3 Implementation Challenges on Stack Overflow. In *IEEE VIS 2022*. Oklahoma City, OK.
[4] Leilani Battle and Alvitta Ottley. 2022. A Programmatic Definition of Visualization Tasks, Insights, and Objectives. https://doi.org/10.48550/arXiv.2206.04767 arXiv:2206.04767 [cs]
[5] Enrico Bertini. 2022. Building (Easy-To-Adopt) Software While Doing Visualization Research. https://filwd.substack.com/p/building-easy-to-adopt-software-while
[6] A. F. Blackwell, C. Britton, A. Cox, T. R. G. Green, C. Gurr, G. Kadoda, M. S. Kutar, M. Loomes, C. L. Nehaniv, M. Petre, C. Roast, C. Roe, A. Wong, and R. M. Young. 2001. Cognitive Dimensions of Notations: Design Tools for Cognitive Technology. In *Cognitive Technology: Instruments of Mind*, Meurig Beynon, Chrystopher L. Nehaniv, and Kerstin Dautenhahn (Eds.). Springer Berlin Heidelberg, 325–341.
[7] M. Bostock, V. Ogievetsky, and J. Heer. 2011. D^3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (Dec. 2011), 2301–2309. https://doi.org/10.1109/TVCG.2011.185
[8] Virginia Braun and Victoria Clarke. 2006. Using Thematic Analysis in Psychology. *Qualitative Research in Psychology* 3, 2 (Jan. 2006), 77–101. https://doi.org/10.

1191/1478088706qp063oa
[9] Virginia Braun and Victoria Clarke. 2019. Reflecting on Reflexive Thematic Analysis. *Qualitative Research in Sport, Exercise and Health* 11, 4 (Aug. 2019), 589–597. https://doi.org/10.1080/2159676X.2019.1628806
[10] Virginia Braun, Victoria Clarke, Nikki Hayfield, and Gareth Terry. 2019. Thematic Analysis. In *Handbook of Research Methods in Health Social Sciences*, Pranee Liamputtong (Ed.). Springer, Singapore, 843–860. https://doi.org/10.1007/978-981-10-5251-4_103
[11] M. Brehmer and T. Munzner. 2013. A Multi-Level Typology of Abstract Visualization Tasks. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (Dec. 2013), 2376–2385. https://doi.org/10.1109/TVCG.2013.124
[12] Mackinlay Card. 1999. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann.
[13] Souti Chattopadhyay, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. 2020. What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, Honolulu HI USA, 1–12. https://doi.org/10.1145/3313831.3376729
[14] Qing Chen, Fuling Sun, Xinyue Xu, Zui Chen, Jiazhe Wang, and Nan Cao. 2022. VizLinter: A Linter and Fixer Framework for Data Visualization. *IEEE Transactions on Visualization and Computer Graphics* 28, 1 (Jan. 2022), 206–216. https://doi.org/10.1109/TVCG.2021.3114804
[15] Ran Chen, Xinhuan Shu, Jiahui Chen, Di Weng, Junxiu Tang, Siwei Fu, and Yingcai Wu. 2021. Nebula: A Coordinating Grammar of Graphics. *IEEE Transactions on Visualization and Computer Graphics* (2021), 1–1. https://doi.org/10.1109/TVCG.2021.3076222
[16] Yiru Chen and Eugene Wu. 2022. PI2: End-to-end Interactive Visualization Interface Generation from Queries. (2022), 15. https://doi.org/doi.org/10.1145/3514221.3526166
[17] Ed Huai-Hsin Chi and J.T. Riedl. 1998. An Operator Interaction Framework for Visualization Systems. In *Proceedings IEEE Symposium on Information Visualization (Cat. No.98TB100258)*. 63–70. https://doi.org/10.1109/INFVIS.1998.729560
[18] Pierre Dragicevic and Yvonne Jansen. 2018. Blinded with Science or Informed by Charts? A Replication Study. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (Jan. 2018), 781–790. https://doi.org/10.1109/TVCG.2017.2744298
[19] Ian Drosos, Titus Barik, Philip J. Guo, Robert DeLine, and Sumit Gulwani. 2020. Wrex: A Unified Programming-by-Example Interaction for Synthesizing Readable Code for Data Scientists. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, Honolulu HI USA, 1–12. https://doi.org/10.1145/3313831.3376442
[20] Will Epperson, Doris Jung-Lin Lee, Leijie Wang, Kunal Agarwal, Aditya G Parameswaran, Dominik Moritz, and Adam Perer. 2022. Leveraging Analysis History for Improved In Situ Visualization Recommendation. *Computer Graphics Forum* (2022), 11.
[21] Tong Ge, Bongshin Lee, and Yunhai Wang. 2021. CAST: Authoring Data-Driven Chart Animations. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3411764.3445452
[22] T. Ge, Y. Zhao, B. Lee, D. Ren, B. Chen, and Y. Wang. 2020. Canis: A High-Level Language for Data-Driven Chart Animations. *Computer Graphics Forum* 39, 3 (June 2020), 607–617. https://doi.org/10.1111/cgf.14005
[23] M. Gleicher. 2018. Considerations for Visualizing Comparison. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (Jan. 2018), 413–423. https://doi.org/10.1109/TVCG.2017.2744199
[24] L Grammel, M Tory, and M Storey. 2010. How Information Visualization Novices Construct Visualizations. *IEEE Transactions on Visualization and Computer Graphics* 16, 6 (Nov. 2010), 943–952. https://doi.org/10.1109/TVCG.2010.164
[25] Saul Greenberg and Bill Buxton. 2008. Usability Evaluation Considered Harmful (Some of the Time). In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 111–120.
[26] Garrett Grolemund and Hadley Wickham. 2014. A Cognitive Interpretation of Data Analysis. *International Statistical Review* 82, 2 (2014), 184–204. https://doi.org/10.1111/insr.12028
[27] Jeffrey Heer, Jock Mackinlay, Chris Stolte, and Maneesh Agrawala. 2008. Graphical Histories for Visualization: Supporting Analysis, Communication, and Evaluation. *IEEE transactions on visualization and computer graphics* 14, 6 (2008), 1189–1196.
[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 (UIST '19)*. Association for Computing Machinery, New York, NY, USA, 281–292. https://doi.org/10.1145/3332165.3347925
[29] 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, Montreal QC Canada, 1–12. https://doi.org/10.1145/3173574.3174106
[30] Sean Kandel, Jeffrey Heer, Catherine Plaisant, Jessie Kennedy, Frank van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, Dominique Brodbeck, and Paolo Buono. 2011. Research Directions in Data Wrangling: Visualizations and Transformations for Usable and Credible Data. *Information Visualization* 10, 4 (Oct. 2011), 271–288. https://doi.org/10.1177/1473871611415994
[31] Mary Beth Kery, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wongsuphasawat, and Kayur Patel. 2020. Mage: Fluid Moves Between Code and Graphical Work in Computational Notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. ACM, Virtual Event USA, 140–151. https://doi.org/10.1145/3379337.3415842
[32] Younghoon Kim and Jeffrey Heer. 2020. Gemini: A grammar and recommender system for animated transitions in statistical graphics. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2020), 485–494.
[33] Doris Jung-Lin Lee, Dixin Tang, Kunal Agarwal, Thyne Boonmark, Caitlyn Chen, Jake Kang, Ujjaini Mukhopadhyay, Jerry Song, Micah Yong, Marti A. Hearst, and Aditya G. Parameswaran. 2021. Lux: Always-on Visualization Recommendations for Exploratory Dataframe Workflows. *Proceedings of the VLDB Endowment* 15, 3 (Nov. 2021), 727–738. https://doi.org/10.14778/3494124.3494151
[34] Zhicheng Liu, John Thompson, Alan Wilson, Mira Dontcheva, James Delorey, Sam Grigg, Bernard Kerr, and John Stasko. 2018. Data Illustrator: Augmenting Vector Design Tools with Lazy Data Binding for Expressive Visualization Authoring. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3173697
[35] Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. 2021. Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks. In *Proceedings of the 2021 International Conference on Management of Data*. ACM, Virtual Event China, 1235–1247. https://doi.org/10.1145/3448016.3457261
[36] Jock Mackinlay. 1986. Automating the Design of Graphical Presentations of Relational Information. *ACM Transactions on Graphics (TOG)* 5, 2 (April 1986), 110–141. https://doi.org/10.1145/22949.22950
[37] Andrew McNutt, Gordon Kindlmann, and Michael Correll. 2020. Surfacing visualization mirages. (2020), 1–16.
[38] Andrew M McNutt. 1912. No Grammar to Rule Them All: A Survey of JSON-style DSLs for Visualization. *IEEE Transactions on Visualization and Computer Graphics* (1912).
[39] Andrew M McNutt and Ravi Chugh. 2021. Integrated visualization editing via parameterized declarative templates. (2021), 1–14.
[40] Leo A. Meyerovich and Ariel S. Rabkin. 2013. Empirical Analysis of Programming Language Adoption. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA '13)*. Association for Computing Machinery, New York, NY, USA, 1–18. https://doi.org/10.1145/2509136.2509515
[41] Thomas Mock. 2022. Tidy Tuesday: A Weekly Data Project Aimed at the R Ecosystem. https://www.tidytuesday.com
[42] Tamara Munzner. 2009. A Nested Model for Visualization Design and Validation. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (2009), 921–928. https://doi.org/10.1109/TVCG.2009.111
[43] Leslie Myint, Aboozar Hadavand, Leah Jager, and Jeffrey Leek. 2020. Comparison of Beginning R Students' Perceptions of Peer-Made Plots Created in Two Plotting Systems: A Randomized Experiment. *Journal of Statistics Education* 28, 1 (Jan. 2020), 98–108. https://doi.org/10.1080/10691898.2019.1695554
[44] Bahare Naimipour, Mark Guzdial, and Tamara Shreiner. 2020. Engaging Pre-Service Teachers in Front-End Design: Developing Technology for a Social Studies Classroom. In *2020 IEEE Frontiers in Education Conference (FIE)*. IEEE, Uppsala, 1–9. https://doi.org/10.1109/FIE44824.2020.9273908
[45] Deokgun Park, Steven Mark Drucker, Roland Fernandez, and Niklas Elmqvist. 2017. ATOM: A Grammar for Unit Visualizations. *IEEE Transactions on Visualization and Computer Graphics* 2626, c (2017). https://doi.org/10.1109/TVCG.2017.2785807
[46] Roger D. Peng, Sean Kross, and Brooke Anderson. 2020. *Mastering Software Development in R*.
[47] Jeffrey M. Perkel. 2018. Why Jupyter Is Data Scientists' Computational Notebook of Choice. *Nature* 563, 7729 (Oct. 2018), 145–146. https://doi.org/10.1038/d41586-018-07196-1
[48] Xiaoying Pu and Matthew Kay. 2020. A Probabilistic Grammar of Graphics. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, Honolulu HI USA, 1–13. https://doi.org/10.1145/3313831.3376466
[49] Xiaoying Pu, Matthew Kay, Steven M. Drucker, Jeffrey Heer, Dominik Moritz, and Arvind Satyanarayan. 2021. Special Interest Group on Visualization Grammars. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems (CHI EA '21)*. Association for Computing Machinery, New York, NY, USA, 1–3. https://doi.org/10.1145/3411763.3450406
[50] Xiaoying Pu, Sean Kross, Jake M. Hofman, and Daniel G. Goldstein. 2021. Datamations: Animated Explanations of Data Analysis Pipelines. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3411764.3445063

[51] D. Ren, B. Lee, and M. Brehmer. 2019. Charticulator: Interactive Construction of Bespoke Chart Layouts. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (Jan. 2019), 789–799. https://doi.org/10.1109/TVCG.2018.2865158
[52] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. ACM Press, Montreal QC, Canada, 1–12. https://doi.org/10.1145/3173574.3173606
[53] Daniel M Russell and Ed H Chi. 2014. Looking Back: Retrospective Study Methods for HCI. In *Ways of Knowing in HCI*. Springer, 373–393.
[54] Arvind Satyanarayan and Jeffrey Heer. 2014. Lyra: An Interactive Visualization Design Environment. *Computer Graphics Forum* 33, 3 (2014), 351–360. https://doi.org/10.1111/cgf.12391
[55] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 341–350. https://doi.org/10.1109/TVCG.2016.2599030
[56] 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 (Jan. 2016), 659–668. https://doi.org/10.1109/TVCG.2015.2467091
[57] 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 (UIST '14)*. ACM, New York, NY, USA, 669–678. https://doi.org/10.1145/2642918.2647360
[58] Nischal Shrestha, Titus Barik, and Chris Parnin. 2021. Remote, but Connected: How #TidyTuesday Provides an Online Community of Practice for Data Scientists. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1 (April 2021), 1–31. https://doi.org/10.1145/3449126
[59] Carson Sievert. 2020. *Interactive web-based data visualization with R, plotly, and shiny*. CRC Press.
[60] C. Stolte, D. Tang, and P. Hanrahan. 2002. Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases. *IEEE Transactions on Visualization and Computer Graphics* 8, 1 (Jan. 2002), 52–65. https://doi.org/10.1109/2945.981851
[61] John W Tukey. 1977. *Exploratory Data Analysis*. Addison-Wesley Pub. Co., Reading, Mass.
[62] Jacob VanderPlas, Brian 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 (Dec. 2018), 1057. https://doi.org/10.21105/joss.01057
[63] April Yi Wang, Will Epperson, Robert A DeLine, and Steven M. Drucker. 2022. Diff in the Loop: Supporting Data Comparison in Exploratory Data Analysis. In *CHI Conference on Human Factors in Computing Systems*. ACM, New Orleans LA USA, 1–10. https://doi.org/10.1145/3491102.3502123
[64] Zijie J Wang, Katie Dai, and W Keith Edwards. 2022. StickyLand: Breaking the Linear Presentation of Computational Notebooks. (2022), 1–7.
[65] 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
[66] Hadley Wickham. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. springer.
[67] Hadley Wickham. 2019. *Advanced R* (second edition ed.). CRC Press/Taylor and Francis Group, Boca Raton.
[68] Hadley Wickham, Mara Averick, Jennifer Bryan, Winston Chang, Lucy McGowan, Romain François, Garrett Grolemund, Alex Hayes, Lionel Henry, Jim Hester, Max Kuhn, Thomas Pedersen, Evan Miller, Stephan Bache, Kirill Müller, Jeroen Ooms, David Robinson, Dana Seidel, Vitalie Spinu, Kohske Takahashi, Davis Vaughan, Claus Wilke, Kara Woo, and Hiroaki Yutani. 2019. Welcome to the Tidyverse. *Journal of Open Source Software* 4, 43 (Nov. 2019), 1686. https://doi.org/10.21105/joss.01686
[69] Hadley Wickham, Romain François, Lionel Henry, and Kirill Müller. 2020. Dplyr: A Grammar of Data Manipulation. RStudio.
[70] Leland Wilkinson. 2005. *The Grammar of Graphics*. Springer-Verlag, New York.
[71] Krist Wongsuphasawat. 2020. Encodable: Configurable grammar for visualization components. In *2020 IEEE Visualization Conference (VIS)*. IEEE, 131–135.
[72] Krist Wongsuphasawat. 2020. Navigating the Wide World of Data Visualization Libraries. https://medium.com/nightingale/navigating-the-wide-world-of-web-based-data-visualization-libraries-798ea9f536e7
[73] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2016. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (Jan. 2016), 649–658. https://doi.org/10.1109/TVCG.2015.2467191
[74] 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 (CHI '17)*. Association for Computing Machinery, New York, NY, USA, 2648–2659. https://doi.org/10.1145/3025453.3025768
[75] Jo Wood, Alexander Kachkaev, and Jason Dykes. 2018. Design Exposition with Literate Visualization. *IEEE Transactions on Visualization and Computer Graphics* (2018), 1–1. https://doi.org/10.1109/TVCG.2018.2864836
[76] Aoyu Wu, Wai Tong, Haotian Li, Dominik Moritz, Yong Wang, and Huamin Qu. 2022. ComputableViz: Mathematical Operators as a Formalism for Visualisation Processing and Analysis. In *CHI Conference on Human Factors in Computing Systems (CHI '22)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3491102.3517618
[77] Yifan Wu, Joseph M. Hellerstein, and Arvind Satyanarayan. 2020. B2: Bridging Code and Interactive Visualization in Computational Notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. ACM, Virtual Event USA, 152–165. https://doi.org/10.1145/3379337.3415851
[78] Yihui Xie, J. J. Allaire, and Garrett Grolemund. 2019. *R Markdown: The Definitive Guide*. CRC Press, Taylor and Francis Group, Boca Raton.
[79] Jian Zhao, Mingming Fan, and Mi Feng. 2020. ChartSeer: Interactive Steering Exploratory Visual Analysis with Machine Intelligence. *IEEE Transactions on Visualization and Computer Graphics* (2020), 1–1. https://doi.org/10.1109/TVCG.2020.3018724

A GENERALIZING GGPLOT2 FINDINGS TO VEGA-LITE

We demonstrate on how generalizable our findings might be to another GoG-based visualization library, Vega-Lite. Table 3 shows the same two options to transform data in `ggplot2` and Vega-Lite. In Figures 15, 16 and 17, we show that our participants' `ggplot2` specification and analyses can be directly rewritten in Vega-Lite with similar GoG components and syntaxes.

GoG component related to data transformation	**ggplot2**, see book [66, Chpt. 5 & 21]	**Vega-Lite**, see documentation: https://vega.github.io/vega-lite/docs/transform.html
Statistics/transformation	Example: `stat_density()` Out-of-the-box statistical transformations can execute count, density, and other common wrangling operations. Analysts can write custom stats, but doing so requires knowledge of `ggplot2` internals and object-oriented programming.	`.transform(vl.density("density"))` Also called "view-level" transform. There is a pre-defined list of supported transformations (see documentation), including many common operators such as pivot and join.
Data, inside aesthetic mapping/encoding	Example: `aes(x = sort(var))` Analysts can apply arbitrary function to the data variable/column specified in aesthetic mapping, such as taking the absolute value, or casting strings to a discrete variable type (factor in R).	`.encode(vl.x().field("var").sort())` Also called "field-level" or "inline" transform, including aggregate, bin, sort, stack, and timeUnit operators. The data grouping is inferred from unaggregated fields, if applicable.

Table 3: Comparing options of data transformation in `ggplot2` and Vega-Lite. Both `ggplot2` and Vega-Lite allow data transformation through the GoG data and statistics components.

Customization with text annotation position

Code example from P7, mentioned in Sections 5.2.1 and 6.1.2

1. Problematic specification: P7 pasted an annotation from another project

ggplot2

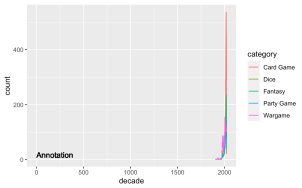
Specification

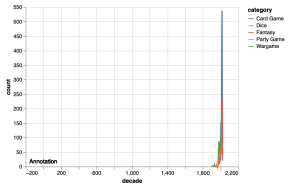
```
ggplot(categories) +
  geom_line(
    aes(
      x = decade,
      y = count,
    color = category)) +
  geom_text(
    aes(
      x = -0.8,
      y = 17,
      label = "Annotation"),
    hjust = "left")
```

Vega-Lite

```
vl.data(categories)
  .layer(
    vl.markLine()
      .encode(
       vl.x().fieldQ("decade"),
       vl.y().fieldQ("count"),
       vl.color().field("category")
      ),
    vl.markText({align: "left"})
      .encode(
        vl.x().datum(-0.8),
        vl.y().datum(17),
        vl.text().datum("Annotation")
      )
  )
.render()
```

Output





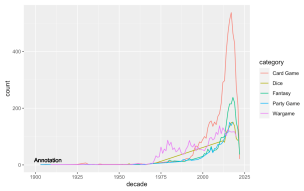
2. Fix we provide: "nudging" annotation in the data space (possible in ggplot2 and Vega-Lite), or use pixel offsets (Vega-Lite)

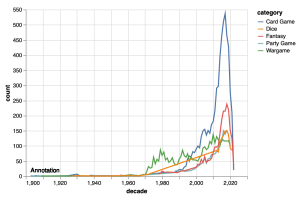
Specification

```
ggplot(categories) +
  geom_line(
    aes(
      x = decade,
      y = count,
      color = category)) +
  geom_text(
    aes(
      x = -0.8,
      y = 17,
      label = "Annotation"),
    position = position_nudge(x = 1900),
    hjust = "left")
```

```
vl.data(categories)
  .layer(
    vl.markLine()
      .encode(
       vl.x().fieldQ("decade"),
       vl.y().fieldQ("count"),
       vl.color().field("category")
      ),
    vl.markText({
      dx: -165,
      dy: 140
    })
      .encode(
        vl.text().datum("Annotation")
      )
  )
.render()
```

Output





Legend (ggplot2/Vega-Lite)

```
data
aesthetic mapping/encoding
geometry/mark
scale
```

```
// Hard-coded data values
```

```
// Text pushed the lines to the side
```

```
// Vega-Lite: dx, dy offsets in pixels;
no explicit x, y encodings needed

// ggplot2: offset specified in data
space (x = 1900)
```

```
// For correspondance between ggplot2
and Vega-Lite, we write Vega-Lite
specifications with its Javascript API
instead of the JSON form.
```

Figure 15: P7's code example about placing a text annotation, reproduced in Vega-Lite syntax. The two versions of the code show similar syntax, suggesting that our findings might generalize to users of Vega-Lite, another GoG-based visualization library. Vega-Lite allows for offsets in pixel units, which are arguments only available for the text mark, not part of the GoG encodings.

Customization with colors

Code example from P1, mentioned in Sections 5.2.1 and 6.1.2

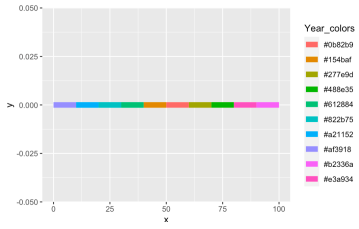
1. Problematic specification: using aesthetic space values (hex codes) as data space values

ggplot2

Specification

```
ggplot(inferred_year_segment) +
  geom_segment(
    aes(
      x = x, xend = x + 10,
      y = 0, yend = 0,
      color = Year_colors),
    size = 3)
```

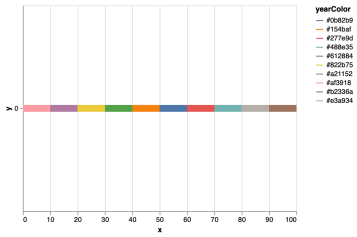
Output



Vega-Lite

```
vl.data(inferred_year_segment)
  .layer(
    vl.markLine({ "strokeWidth": 10 })
      .encode(
        vl.x().fieldQ("x"),
        vl.y().fieldQ("y"),
        vl.color().field("yearColor")
      )
  )
  .render()
```

Output

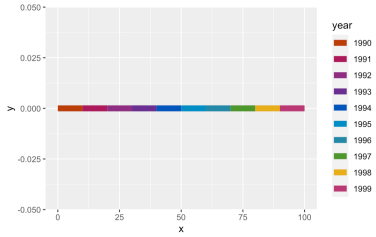


2. P1's corrected specification: using custom scale function

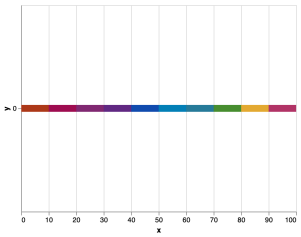
Specification

```
ggplot(inferred_year_segment) +
  geom_segment(
    aes(
      x = x, xend = x + 10,
      y = 0, yend = 0,
      color = year),
    size = 3) +
  scale_color_manual(
      values = Year_colors)
```

Output




```
vl.data(inferred_year_segment)
  .layer(
    vl.markLine({ "strokeWidth": 10 })
      .encode(
        vl.x().fieldQ("x"),
        vl.y().fieldQ("y"),
        vl.color()
          .fieldN("year")
          .scale({range: yearColor})
      )
  )
  .render()
```

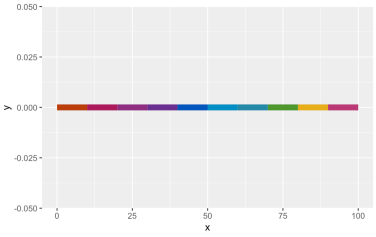


3. Alternative specification we provide: using the identity (null) scale function

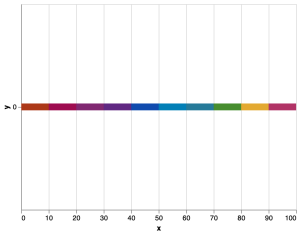
Specification

```
ggplot(inferred_year_segment) +
  geom_segment(
    aes(
      x = x, xend = x + 10,
      y = 0, yend = 0,
      color = I(Year_colors)),
    size = 3)
```

Output




```
vl.data(inferred_year_segment)
  .layer(
    vl.markLine({ "strokeWidth": 10 })
      .encode(
        vl.x().fieldQ("x"),
        vl.y().fieldQ("y"),
        vl.color()
          .field("yearColor")
          .scale(null)
      )
  )
  .render()
```



Legend (ggplot2/Vega-Lite)

```
data
aesthetic mapping/encoding
geometry/mark
scale
```

```
// Vega-Lite does not support
arithmetics (x + 10) or array of
colors (Year_colors) within encode();
otherwise this is a direct
translation.
```

```
// ggplot2 and Vega-Lite outputs still
have the default color palette; color
hex codes are treated as a
categorical/discrete variable.
```

```
// In ggplot2 and Vega-Lite (GoG in
general), a scale function maps data
space values (domain) into the
aesthetic space (range). Here the
range of the color scale is changed
from the defaults to a custom array
yearColor
```

```
// The domain and range are the same
for the identity/null scale function,
both in the aesthetic space.
```

```
// For correspondance between ggplot2
and Vega-Lite, we write Vega-Lite
specifications with its Javascript API
instead of the JSON form.
```

Figure 16: P1's code example about applying a custom color palette, reproduced in Vega-Lite syntax. The two versions of the code use the same GoG components and have similar syntax, suggesting that our findings might generalize to users of Vega-Lite, another GoG-based visualization library.

Silent error example

Code example mentioned in Sections 5.3

1. Problematic: data transformation implies top categories conditional on decade, while visualization spec does not.

Specification

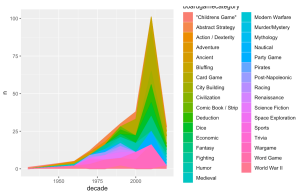
ggplot2 + dplyr

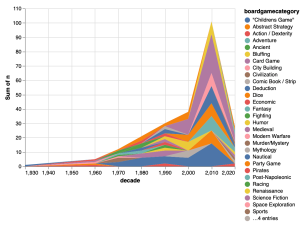
```
categories %>%
  group_by(decade, boardgamecategory) %>%
  count(name = "n") %>%
  group_by(decade) %>%
  mutate(rank = min_rank(-n)) %>%
  filter(rank <= 5) %>%
  ungroup() %>%
  mutate(boardgamecategory =
        fct_drop(boardgamecategory)) %>%
  complete(decade, boardgamecategory,
        fill = list(n = 0)) %>%
  ggplot() +
  geom_area(
    aes(
      x = decade,
      y = n,
      fill = boardgamecategory)
  )
```

Vega-Lite

```
vl.markArea()
  .data(categories)
  .transform(
    vl.groupby(
        ["decade", "boardgamecategory"])
      .aggregate(
        vl.count().as("n")),
    vl.groupby("decade")
      .window(
        vl.rank().as("rank"))
        .sort(vl.field("n")
        .order("descending")),
    vl.filter('datum.rank <= 5')
  )
  .encode(
    vl.x().fieldQ("decade"),
    vl.y().fieldQ("n").aggregate("sum"),
    vl.color().field("boardgamecategory")
  )
  .render()
```

Output





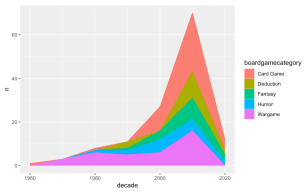
2. Fix we provide: get top 5 boardgame category without conditional on "decade"

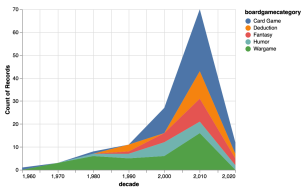
Specification

```
categories %>%
  group_by(boardgamecategory) %>%
  count(name="n") %>%
  ungroup() %>%
  mutate(rank = min_rank(-n))  %>%
  filter(rank <= 5)  %>%
  select(boardgamecategory) %>%
  left_join(
    categories %>% group_by(
      decade, boardgamecategory) %>%
      count(),
    by = "boardgamecategory") %>%
  mutate(boardgamecategory =
          fct_drop(boardgamecategory)) %>%
  complete(decade, boardgamecategory,
          fill = list(n = 0))  %>%
  ggplot() +
  geom_area(
    aes(
      x = decade,
      y = n,
      fill = boardgamecategory)
  )
```

```
vl.markArea()
  .data(categories)
  .transform(
    vl.groupby("boardgamecategory")
      .join([{op: "count", as: "n"}]),
    vl.window(
      vl.dense_rank().as("rank"))
    .sort(
      vl.field("n").order("descending")),
    vl.filter('datum.rank <= 5')
  )
.encode(
    vl.x().fieldQ("decade")
    vl.y().fieldQ("n").aggregate("count"),
    vl.color().field("boardgamecategory")
  )
  .render()
```

Output





Legend (ggplot2/Vega-Lite)

```
data
aesthetic mapping/encoding
geometry/mark
data transformation
```

```
// Too many boardgame categories

// Without grouping by "decade"

// Added the join

// Top 5 categories overall

// For correspondance between ggplot2
and Vega-Lite, we write Vega-Lite
specifications with its Javascript API
instead of the JSON form.
```

Figure 17: P6/P7's code example about silent error, reproduced in Vega-Lite syntax. The two versions of the code share similar syntax and produce the same data operations, suggesting that our findings might generalize to users of Vega-Lite, another GoG-based visualization library. All our participants wrangled data outside `ggplot2` specifications using `dplyr` in R, while Vega-Lite provides its own data transformation functions.