ggdist: Visualizations of Distributions and Uncertainty in the Grammar of Graphics

Matthew Kay

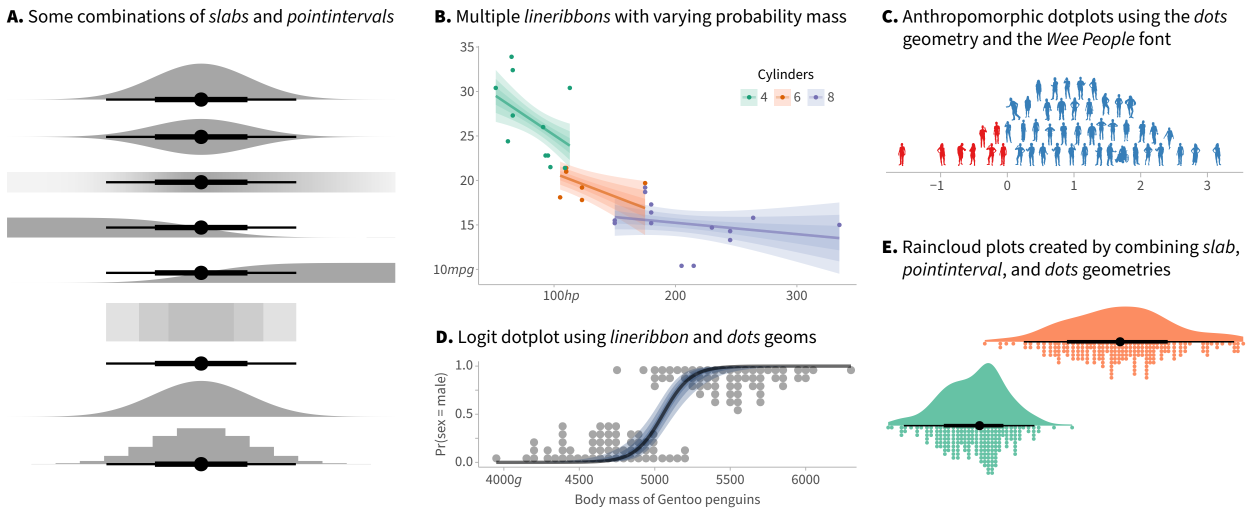


Fig. 1: Examples from the three major classes of geometries in *ggdist*: **(A)** slabintervals, such as density plots, CDFs, intervals, and gradient plots; **(B)** lineribbons, such as uncertainty bands and fan charts; and **(C)** dots, such as dotplots and beeswarm charts. Myriad combinations of these are possible, leading to charts like **(D)** *logit dotplots* and **(E)** *raincloud plots* [1].

Abstract—The grammar of graphics is ubiquitous, providing the foundation for a variety of popular visualization tools and toolkits. Yet support for uncertainty visualization in the grammar graphics—beyond simple variations of error bars, uncertainty bands, and density plots—remains rudimentary. Research in uncertainty visualization has developed a rich variety of improved uncertainty visualizations, most of which are difficult to create in existing grammar of graphics implementations. *ggdist*, an extension to the popular *ggplot2* grammar of graphics toolkit, is an attempt to rectify this situation. *ggdist* unifies a variety of uncertainty visualization types through the lens of distributional visualization, allowing functions of distributions to be mapped to directly to visual channels (aesthetics), making it straightforward to express a variety of (sometimes weird!) uncertainty visualization types. This distributional lens also offers a way to unify Bayesian and frequentist uncertainty visualization by formalizing the latter with the help of confidence distributions. In this paper, I offer a description of this uncertainty visualization paradigm and lessons learned from its development and adoption: *ggdist* has existed in some form for about six years (originally as part of the *tidybayes* R package for post-processing Bayesian models), and it has changed substantially over that time, with several rewrites and API re-organizations as it changed based on user feedback and expanded to cover increasing varieties of uncertainty visualization types. Ultimately, given the huge expressive power of the grammar of graphics and the popularity of tools built on it, I hope a catalog of my experience with *ggdist* will provide a catalyst for further improvements to formalizations and implementations of uncertainty visualization in grammar of graphics ecosystems.
A free copy of this paper and all supplemental materials is available at https://github.com/mjskay/ggdist-paper and is archived on Zenodo at doi:10.5281/zenodo.7770984.

Index Terms—Uncertainty visualization, probability distributions, confidence distributions, grammar of graphics

1 Introduction

The grammar of graphics [53] is ubiquitous, providing the foundation for a variety of visualization toolkits. Yet support for uncertainty visualization in grammar graphics systems, generally speaking, remains rudimentary. Popular implementations like *ggplot2* [48, 49] and *Vega-lite* [44] typically provide versions of error bars (for points), uncertainty bands (for lines), boxplots, and density plots. However, research in uncertainty visualization has developed a rich variety of alternative uncertainty representations, often designed to address shortcomings in those existing visualization types. Examples include, but are not limited to, quantile dotplots [14, 31], gradient error bars [11], gradient plots [6, 25], fan charts [25], and innumerable variations on eye plots [4, 19, 31, 45]. Yet, most of these alternative representations are—simply put—painful to convince existing grammar of graphics implementations to produce.

ggdist [29] is an attempt to rectify this situation. It started under the guise of *tidybayes* [30]—an R package I wrote for post-processing Bayesian models for use with *ggplot2*—in 2016. *tidybayes* slowly gained usage in the Bayesian statistics community, but the package always had two complementary, but not perfectly aligned, uses: post-processing Bayesian model output for visualization, and creating uncertainty visualizations in the grammar of graphics. The latter is bigger than *just* Bayesian statistics: everyone needs to visualize uncertainty! And, contrary to popular opinion—as we'll see later—Bayesian and

• *Matthew Kay is with Northwestern University. E-mail: mjskay@northwestern.edu*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

frequentist uncertainty visualization can be done within the same framework. Recognizing this broader need, in 2020 I spun off the uncertainty visualization components of *tidybayes* into a new package, *ggdist*, and published it to CRAN [21]. Since then it has steadily grown in use inside and outside the Bayesian statistics community in R, and now averages about 14,000 downloads per month (best described as "modest").

ggdist aims to be (1) a coherent extension to the grammar of graphics that makes it easy to create a variety of uncertainty visualizations, and (2) an implementation of a particular formalism for describing uncertainty visualizations: mappings of functions of distributions (e.g. densities, CDFs, quantiles) onto aesthetics (e.g. height, color, opacity). That formalism is the primary focus and contribution of this paper. My ambition is that not only should that formalism be able to express a variety of uncertainty visualizations through a single coherent framework, but that that framework should be complete enough that someone else can wander along and use it to express new uncertainty visualization types I've never thought of before.[1] Naturally this is hard to demonstrate, but I will attempt to do so by giving a tour of *ggdist* through recreations of a variety of uncertainty visualizations in the literature. Along the way, I'll discuss lessons I've learned in how to effectively integrate uncertainty visualization into the grammar of graphics.

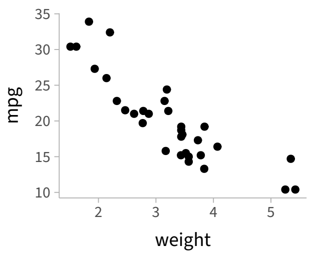
Ultimately, in the spirit of recent retrospectives on visualization system design [43], I hope to distill down some of what I've learned developing a moderately-well-used uncertainty visualization toolkit.[2] Given the huge expressive power of the grammar of graphics and the popularity of tools built on it, I hope a systematic approach to integrating uncertainty into the grammar of graphics might provide a catalyst for improved implementations of uncertainty visualization to flourish in existing grammar of graphics ecosystems, and ultimately for even better formal descriptions of uncertainty visualization to arise.

2 Setting the Stage

2.1 A simplified notation for the grammar of graphics

As I would like to talk more generally than a specific grammar of graphics implementation—that is, I am more concerned with the formalism underlying *ggdist* than with the idiosyncrasies of its API—I'll need a formal way of writing down visualization specifications separated from a particular implementation. I'll adopt here a notation that I've used when teaching *ggplot2*, *Vega-lite*, *Altair*, and *Tableau*. I've found students pick it up handily, which is at least some evidence for its understandability. The core notation describes a visualization in terms of its *data variables*, *aesthetic mappings*, and *geometries*. We create a scatterplot of the infamous `mtcars` dataset [20], for example, by mapping one variable onto the x aesthetic and another onto the y aesthetic:[3]

$$\begin{aligned} weight &\to x \\ mpg &\to y \\ \text{GEOM} &= point \end{aligned}$$



This notation sets up two *aesthetic mappings*:[4] a mapping from the *weight* variable onto the x aesthetic, and a mapping from the *mpg* variable onto the y aesthetic. It then employs a *point* geometry[5] for display. I like this notation because it emphasizes that we are creating *functions* from data space to aesthetic (display) space; in grammar of graphics parlance these are *scale* functions which can themselves be specified (e.g., to use log scales, to pick how colors are assigned, etc). Other notations obscure this key insight into the structure of the grammar of graphics by, e.g., placing the aesthetic first and "assigning" data variables to it. In fact, let's see that now: here is a translation of the above into *ggplot2* and *Vega-lite*[6] code, assuming `cars` is a data frame with `weight` and `mpg` columns:

ggplot2:

```
ggplot(cars) +
  aes(
    x = weight,
    y = mpg
  ) +
  geom_point()
```

Vega-lite:

```
vl.data("path/to/cars.json")
  .encode(
    vl.x().fieldQ("weight"),
    vl.y().fieldQ("mpg")
  )
  .markPoint()
```

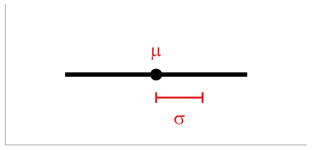
This shows the close correspondence between the abstract notation above and the particulars of code in actual grammar of graphics implementations. Throughout the rest of this paper, I'll stick to the abstract notation and corresponding *ggplot2* + *ggdist* code for some particulars.

2.2 Uncertainty visualization in the grammar of graphics, as she is spoke

Speaking of existing implementations of the grammar of graphics, how do they implement uncertainty visualization? Rudimentarily, I think.

One natural approach to uncertainty visualization is to use a Gaussian approximation: to represent all estimates and their uncertainty as a mean and standard deviation. This makes specification easy: instead of mapping a single value onto the x aesthetic, say, we map a point estimate onto x and provide an aesthetic for its standard deviation; call it x_{SD}. Say we had estimated a variable μ and had quantified its standard error (i.e. the standard deviation of its sampling distribution) as σ, we might plot a point with an error bar using a composite *pointinterval* geometry as follows, yielding a 95% interval calculated from a Normal distribution with mean μ and standard deviation σ:

$$\begin{aligned} \mu &\to x \\ \sigma &\to x_{\text{SD}} \\ \text{GEOM} &= pointinterval \end{aligned}$$



This is one approach taken by *Vega-lite*: it provides an `errorbar`[7] mark (analogous the interval portion of *pointinterval*) and an `xError` channel (analogous to x_{SD}). I'll refer to this as the $\{x, x_{\text{SD}}\}$ approach.

The problem, fundamentally, is that not all uncertainty is well-represented by a Gaussian distribution. Consider uncertainty in a proportion (bounded at 0 and 1, thus as estimates approach the boundary the interval becomes asymmetric—yet Gaussian intervals must be symmetric) or uncertainty in a variance parameter (bounded below at 0). Or, consider the ubiquitous Student-t confidence interval: as the degrees of freedom go to ∞, a Student-t distribution is well-approximated by the Normal, but with low degrees of freedom (incidentally common in small-n studies—like at VIS), the tails of the distribution become fatter, and the Normal distribution is a poor approximation. Thus, a more general approach is needed.

The obvious alternative, at least for interval representations, is to simply specify the interval endpoints; e.g. for a 95% Gaussian interval:

[1]Emphasis on *I*—I think a formalism and its corresponding API truly shows its power when people make things with it that its creator did not conceive.

[2]In the spirit of this being a retrospective, I'm keeping the tone informal. I think that's more honest; besides, after two years of a pandemic, I at least need a break from stilted academic writing. I hope you do too! If not, my condolences.

[3]I am aware it is traditional to slap a number on each figure and float it off into a random corner of the page. I violate that norm throughout this paper without remorse.

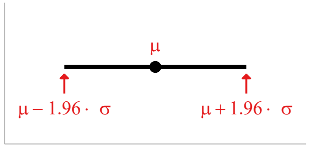
[4]Or in *Vega-lite* parlance, *encodings*

[5]*Vega-lite*-ese: *mark*

[6]I use the Vega-lite API instead of its JSON form, as JSON is a horrifying mess of visual noise that no sane human should want to read or write. The Vega-lite API is a notable improvement, though without R's facility for capturing and re-writing abstract syntax trees, it doesn't quite reach the succinctness of *ggplot2*. This seems to be a fundamental limitation when writing domain-specific languages in JavaScript, though the *Vega-lite* authors have done an excellent job with the language they've been given.

[7]`errorbar` is, in my view, a misnomer, as is `xError`: fundamentally, the mark calculates a Gaussian interval, which might be being used to represent error in an estimate, but might not. *Error* is not the generic notion at play, a standard deviation is; and the generic mark is an interval, not an error bar.

$$\begin{aligned} \mu &\rightarrow x \\ \mu - 1.96 \cdot \sigma &\rightarrow x_{\text{MIN}} \\ \mu + 1.96 \cdot \sigma &\rightarrow x_{\text{MAX}} \\ \text{GEOM} &= \textit{pointinterval} \end{aligned}$$



Here, the magic values -1.96 and $+1.96$ are the $(1-95\%)/2 = 2.5\%$th and $(1+95\%)/2 = 97.5\%$th quantiles of the standard Normal distribution, thus yielding a $97.5\% - 2.5\% = 95\%$ interval. This is generic in the sense that any interval can be represented, but unsatisfying in the sense that we have lost some level of abstraction that was present when we were just thinking in terms of estimates and their variances. This approach also requires that the user knows how to make these calculations. Both *ggplot2* (with `geom_pointrange`) and *Vega-lite* (with the `x` and `x2` channels supplied to `errorbar`) offer a variant of this solution for pre-calculated intervals.

I will offer a different solution: to instead represent intervals as properties of a distribution, allowing us to neatly handle both the simple case of Gaussian error and more complex cases. Centering distributions—not standard deviations or intervals—in the specification of uncertainty will also allow us to build a richer set of uncertainty representations.

3 Uncertainty Visualization as Distributional Visualization

3.1 Intervals

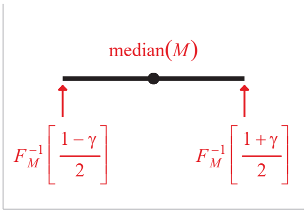
Imagine we represent an uncertain value generically as a distribution, or a random variable, M. Importantly, I do not consider this a *probability* distribution necessarily: it could be a probability distribution, but it could also be a *confidence* distribution, which is a frequentist generalization of sampling distributions and bootstrap distributions [54]. Its defining characteristic will be that it has a cumulative distribution function (CDF), $F_M(x)$, which is:

- For a probability distribution, $F_M(x) = \Pr(M \le x)$, the probability that M is less than or equal to x.
- For a confidence distribution, $F_M(x) = \gamma$ is the confidence γ at which x would be the upper limit of a one-sided $\gamma\%$ confidence interval, $[-\infty, x]$, for M.

For either representation, we may also be interested in other functions of the distribution. These include the derivative of the cumulative distribution function, i.e. the density function (or the mass function, if the distribution is discrete), $f_M(x)$, as well as the inverse of the CDF (also known as the quantile function), $F_M^{-1}(x)$. Given these functions, we can generate a variety of uncertainty representations, including but not limited to density plots and intervals.

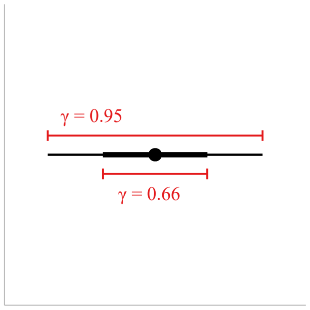
For example, a median and $\gamma\%$ quantile interval could be defined generically on any distribution M as follows:

$$\begin{aligned} \text{median}(M) &\rightarrow x \\ F_M^{-1}\left[\frac{1-\gamma}{2}\right] &\rightarrow x_{\text{MIN}} \\ F_M^{-1}\left[\frac{1+\gamma}{2}\right] &\rightarrow x_{\text{MAX}} \\ \text{GEOM} &= \textit{pointinterval} \end{aligned}$$



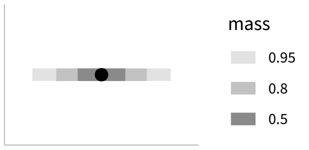
If M is a probability distribution, this is a Bayesian *credible* interval, and if M is a confidence distribution, this is a frequentist *confidence* interval. This lets us abstract over the petty battles between this or that statistical camp and get to the meaningful business of visualizing uncertainty. This also allows us something not present in other attempts so far: to make it easy to specific multiple interval sizes, and to map interval size itself onto an aesthetic. For example, if we[8] wanted to show two intervals, a 95% and a 66%, where the smaller interval is shown as a thicker line, we could write:

$$\begin{aligned} \text{median}(M) &\rightarrow x \\ F_M^{-1}\left[\frac{1-\gamma}{2}\right] &\rightarrow x_{\text{MIN}} \\ F_M^{-1}\left[\frac{1+\gamma}{2}\right] &\rightarrow x_{\text{MAX}} \\ \gamma &\rightarrow \textit{linewidth} \\ \text{GEOM} &= \textit{pointinterval} \\ \gamma &\in \{0.66, 0.95\} \end{aligned}$$



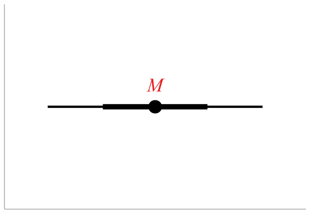
This is a not-uncommon approach that tries to avoid dichotomous thinking by showing multiple intervals of different masses. It also has the nice grammar-of-graphics-ish property of mapping the mass (γ) onto the width of the line, instead of creating two explicit, separate layers, each specifying a different interval—it makes the mass into *data*.[9] This also makes it easy to generalize to other visualizations, e.g. by modifying the previous specification to map mass onto *color* instead of *linewidth*:

$$\begin{aligned} \gamma &\rightarrow \textit{color} \\ \text{GEOM} &= \textit{pointinterval} \\ \gamma &\in \{0.50, 0.80, 0.95\} \end{aligned}$$



On the other hand, these are still a bit low-level: they require the user to know how to calculate interval endpoints from the quantile function. This also limits us specifically to quantile intervals, when other intervals types, such as highest-density intervals [24] or shortest intervals [34], might be preferable. Thus, *ggdist* also supplies a *stat* version of *pointinterval*, which bundles up some statistical calculations and default aesthetic mappings with the *pointinterval* geometry.[10] All *stats* in *ggdist* support the x_{DIST} and y_{DIST} aesthetics, onto which objects that represent distributions can be mapped. They also allow the user to specify the type of point and interval used, and generate the corresponding values and mappings for x, x_{MIN}, x_{MAX}, and *linewidth*. This changes the specification to something like:

$$\begin{aligned} M &\rightarrow x_{\text{DIST}} \\ \text{STAT} &= \textit{pointinterval} \\ \text{POINT} &= \textit{median} \\ \text{INTERVAL} &= \textit{quantile interval} \\ \gamma &\in \{0.66, 0.95\} \end{aligned}$$



The representation of the distribution M could be a sample-based representation, e.g. a bunch of draws from a Bayesian posterior or from a bootstrap distribution, or it could be an object representing a theoretical distribution in terms of its parameters, such as a Normal distribution with a defined mean and standard deviation. Point estimates and interval types can be defined by arbitrary functions of distributions, and predefined functions for mean, median, and mode, and quantile, highest-density, and shortest intervals are provided. This generalizes the $\{x, x_{\text{SD}}\}$ approach used by *Vega-lite* to any distribution type while abstracting over the specifics of how points and intervals are calculated.

In implementation, *ggdist* allows distributions to be represented

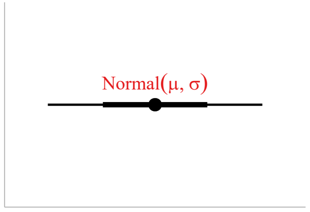
[8] Yes yes, I am using both "I" and "we" in this paper. "I" is me, and "we" is the conspiratorial "we": I'd like to hope you'll come along with me on this exciting journey of trying to sort out reasonable ways to visualize uncertainty.

[9] I learned at least two useful things from a relational databases class in undergrad: (1) it's always better to put data into rows than into column names of tables—an insight that stems from database *normal forms* [10] (distinctions between which I have long forgotten) or what some statisticians call *tidy data* [17]; and (2) you are rarely at Google scale, so you're probably better off with a relational database with proper transactions than some dumb old key–value store. The latter lesson each of my students refuses to learn until they build a webapp to collect data from 300 people using some newfangled database they aren't the target users for, and end up with garbage. Kids these days, etc.

[10] See Sec. 5, or Wickham [48], for more on *stats* and *geoms*.

by numeric vectors (a sample-based representation), objects from the *distributional* R package [39] (which supports theoretical distributions), and rvar objects from the *posterior* R package [7] (a sample-based representation that mimics numeric arrays in R). For example, if we re-create the $\{x, x_{\text{SD}}\}$ representation abstractly thus:

$$\begin{aligned}\text{Normal}(\mu,\sigma) &\rightarrow x_{\text{DIST}}\\ \text{STAT} &= \textit{pointinterval}\\ \text{POINT} &= \textit{median}\\ \text{INTERVAL} &= \textit{quantile interval}\\ \gamma &\in \{0.66, 0.95\}\end{aligned}$$

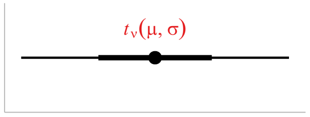


In *ggdist*, using `distributional::dist_normal`, the specification is quite similar:

```
ggplot(data) +
  aes(xdist = dist_normal(mu, sigma)) +
  stat_pointinterval(
    point_interval = median_qi,
    .width = c(.66, .95)
  )
```

These are the default values for `point_interval` and `.width` (γ),[11] so just `stat_pointinterval()` also works here. To demonstrate generalizing this approach, consider the common need to place uncertainty intervals on the results of a t-test, which can be derived from a $t_\nu(\mu,\sigma)$ distribution with ν degrees of freedom, location μ (e.g. an estimated mean), and scale σ (e.g. a standard error). Given these three values in a data frame, a visualization specification might be:

$$\begin{aligned}t_\nu(\mu,\sigma) &\rightarrow x_{\text{DIST}}\\ \text{STAT} &= \textit{pointinterval}\end{aligned}$$



In *ggdist* code, the aesthetic specification closely matches this abstract notation: `aes(xdist = dist_student_t(nu, mu, sigma))`.

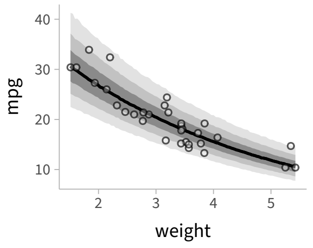
3.2 Ribbons

Once we have *pointinterval* representations, it is straightforward to develop uncertainty band representations by generalizing points to lines and intervals to ribbons—thus, *lineribbon*. Imagine a regression that models car miles per gallon based on weight (the details of the function g are not important):

$$\log(mpg) \sim \text{Normal}\left(g(weight), \sigma\right)$$

Such a model could provide a predictive distribution for a car's miles per gallon conditional on its weight: $p(mpg \mid weight)$, which we might want to plot alongside the raw data. If a *lineribbon* is a geometry combining a line with an arbitrary number of uncertainty bands around it, abstractly, we want something like this:

$$\begin{aligned}weight &\rightarrow x\\ p(mpg \mid weight) &\rightarrow y_{\text{DIST}}\\ \gamma &\rightarrow \textit{fill}\\ \text{STAT} &= \textit{lineribbon}\\ \gamma &\in \{0.50, 0.80, 0.95\}\end{aligned}$$



I also show the raw data as a separate *point* layer, for comparison. Assuming `fit` is a Bayesian version of this model fit using the *brms* modeling package, and `preds` is a data frame of *weight* values to predict on, we can add a column to `preds` that contains a random variable representation of $p(mpg \mid weight)$ using `brms::posterior_predict` [8] and the `posterior::rvar` data type [7]. The latter is a data type I created specifically to wrap large samples that represent distributions into objects that mimic R vectors and arrays, and which can be added to data frames:

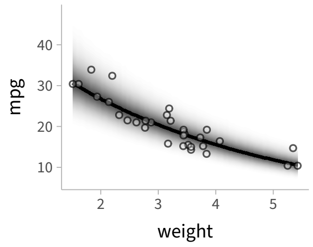
```
preds$mpg_given_weight = rvar(posterior_predict(fit, preds))
```

This creates a new column, `mpg_given_weight`, containing `rvar` objects representing $p(mpg \mid weight)$ for the `weight` on each row of the `preds` data frame. Given this data frame, the code equivalent of the abstract specification above is:

```
ggplot(preds) +
  aes(x = weight, ydist = mpg_given_weight) +
  stat_lineribbon()
```

`stat_lineribbon` defaults to `.width = c(.5, .8, .95)` and maps the resulting `.width` onto the `fill` aesthetic, so we do not need to specify `.width` or `fill` mapping. Once we have a multiple-ribbon geometry, it is easy to create other visualization types, like gradient fan charts [6, 25]. For example, we could use a large number of intervals, say $k = 50$ or 100, with masses between 0 and 1 (exclusive):

$$\begin{aligned}\gamma &\rightarrow \textit{fill}\\ \text{STAT} &= \textit{lineribbon}\\ \gamma &\in \left\{\frac{i-0.5}{k} \;\middle|\; i \in 1\ldots k\right\}\\ k &= 100\end{aligned}$$



This set of k γ values is the same sequence generated by the `ppoints(k)` function in R, so we can pass `.width = ppoints(100)` to `stat_lineribbon` to get a gradient fan chart with 100 intervals. This stems directly from the choice to make γ into data that can be mapped onto aesthetics, and is one example of support for a chart type that was a happy accident of *ggdist*'s design.[12]

3.3 Slabs

Speaking of gradients, the obvious other direction to go for uncertainty—if we are to move beyond intervals—is density plots. Most grammar of graphics implementations support density plots, but these are typically designed only for sample-based representations: they calculate a kernel density estimate (KDE) from a sample and allow this to be visualized. Given *ggdist*'s core abstraction of distributions, we can take this a step further, visualizing both sample-based representations and theoretical distributions.[13]

However, stopping at *just* densities seems the wrong level of abstraction:[14] many useful uncertainty visualizations can be created through the whole suite of distributional functions: CDFs, densities, and quantiles. Thus, *ggdist* instead has a notion of a *slab* geometry, which has a *thickness* onto which arbitrary functions of the distribution can be mapped. For example, imagine two *groups*, *a* and *b*, each with uncertainty in its mean represented by a random variable $M \mid group$. We could specify a density plot of these distributions as:

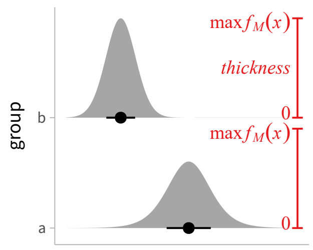
[11] For historical reasons that have to do with a combination of a very long discussion with a bunch of people on the Stan probabilistic programming language forums [28] and naming conventions in some R APIs for fixed arguments to functions with variable argument lists [47], γ in *ggdist* is spelled `.width`. Reflecting on my past mistakes, a better name would be `mass`.

[12] This happy accident was discovered in my response to a user's question in *tidybayes* issue #103.

[13] You may be tempted to say: but Matthew, of course you could easily pre-calculate densities from a theoretical density function and plot them, e.g. using `ggplot2::stat_function`. Unfortunately, once you add non-linear axis transformation into the mix, this is a recipe for silent errors caused by failing to adjust the density by the derivative of the transformation—an error *ggdist* prevents. See Sec. 6.3.

[14] See earlier footnote[7] about *error bars* versus *intervals*.

$$\begin{aligned} M &\rightarrow x_{\text{DIST}} \\ group &\rightarrow y \\ f_M(x) &\rightarrow thickness \\ \text{STAT} &= slab \end{aligned}$$



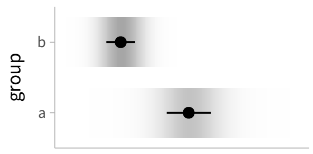
I also include a *pointinterval* with a median and 66% interval above (and elsewhere in this section) for reference. The *slab*s have a subscale for *thickness*,[15] which is an orientation-aware aesthetic (if the distributions were mapped onto y_{DIST}, *thickness* would act as a width instead of height), with a fixed baseline: values of 0 on the thickness scale always correspond to the base of the slab. This makes it appropriate for both probability densities and CDFs, both of which have a natural 0 point. In the above example, the density $f_M(x)$ is mapped onto *thickness*, creating a traditional density plot. Importantly, because the geometries use the same scale, both subscales have the same overall maximum *thickness*, which ensures that the area under both densities is equal.[16]

This is the default output of `stat_slab()` in *ggdist*, which maps densities onto the *thickness* aesthetic. If desired, the mapping $f_M(x) \rightarrow$ *thickness* can be translated to code in one of two ways:

1. `aes(thickness = after_stat(pdf))`: This maps the `pdf` *computed variable* onto *thickness*. Computed variables in *ggplot2* are calculated by *stat*s and made accessible in aesthetic mappings via `after_stat()`. *ggdist* stats provide several computed variables, including `pdf`, `cdf`, and `.width`. See Sec. 5.
2. `aes(thickness = !!p_(x))`: This uses a small domain-specific language for probability expressions I added to *ggdist*, and is intended to more closely mimic a mapping like $p(x) \rightarrow$ *thickness* in code. The `!!` pseudo-operator comes from the *rlang* meta-programming R package, and performs *unquotation* [50]: it inserts the expression returned by `p_(x)`, which in this case is `after_stat(pdf)`, into the `aes` call.[17]

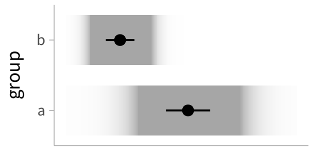
Slab geometries have several useful properties designed to make it easy to create a variety of uncertainty visualizations. One important property is that the *alpha* (opacity), *fill*, and outline *color* aesthetics of the slabs can have data values mapped onto them at a sub-geometry level.[18] This functionality, combined with the ability to map arbitrary distribution functions, means we can easily recreate a bunch of visualizations from the literature. The obvious first example would be a color gradient plot [25], by mapping density onto *alpha*:

$$\begin{aligned} M &\rightarrow x_{\text{DIST}} \\ group &\rightarrow y \\ f_M(x) &\rightarrow alpha \\ \text{STAT} &= slab \end{aligned}$$



This translates naturally to `aes(alpha = !!p_(x))`. Fine fine, but let's get *weird*. Back in 2014, the inimitable Michaels Correll and Gleicher—arguing that error bars should be considered harmful [11]—proposed instead a visualization which has a solid bar inside the 95% interval and gradient tails that fade out beyond the interval, to emphasize the arbitrariness of the 95% confidence level. We can use the CDF to construct a function with these properties, and map it to *alpha*:

$$\begin{aligned} -\max\left[\frac{|2F_M(x)-1|}{0.95}\right] &\rightarrow alpha \\ \text{STAT} &= slab \end{aligned}$$

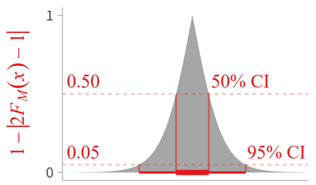


In *ggdist* this function is either:

- `-pmax(abs(2*after_stat(cdf) - 1), .95)` or
- `-pmax(abs(2*!!Pr_(X <= x) - 1), .95)`, using the probability expression mini-DSL.

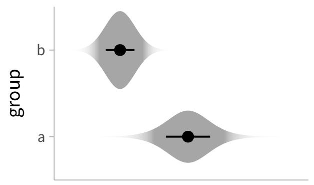
Thus, *ggdist* can create this particularly weird uncertainty visualization without breaking the bonds of its core abstraction. Importantly, the function I've used to construct the tails is not arbitrary; it is meaningful in itself: $1-|2F_M(x)-1|$ is the *consonance curve* [2], which (in the frequentist interpretation) is the two-sided p-value for the null hypothesis $M = x$. Horizontal slices through this curve at a height of α are $(1-\alpha)\%$ quantile intervals:

$$\begin{aligned} 1-|2F_M(x)-1| &\rightarrow thickness \\ \text{STAT} &= slab \end{aligned}$$



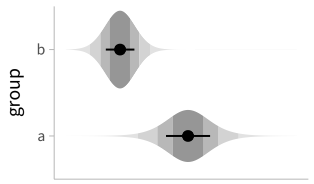
Fair enough, but let's get *weirder*: in 2021, Helske et al [19], inspired by Correll and Gleicher, proposed combining their gradient tail with a violin plot. We can do that by adding back in the density-to-thickness mapping, and use the SIDE parameter, which specifies if the slab should be drawn on the *top* side (default), *bottom* side, or *both*. I'll also adjust the tails to fade outside the 85% interval, since otherwise the fading is hard to see in the skinny tails of the violin:

$$\begin{aligned} f_M(x) &\rightarrow thickness \\ -\max\left[\frac{|2F_M(x)-1|}{0.85}\right] &\rightarrow alpha \\ \text{STAT} &= slab \\ \text{SIDE} &= both \end{aligned}$$



Okay fine, but let's get *even weirder*: Helske et al [19] further suggested using discrete intervals for the colors instead of a gradient, to aid discriminability. We can do that too! A key feature of the *slab* stat is that it *also* computes intervals, and for each point along the slab, retains a column indicating the mass (γ) of the smallest requested interval containing that point.[19] This means that we can integrate intervals directly into the slab by mapping γ onto *alpha* or *fill*:

$$\begin{aligned} f_M(x) &\rightarrow thickness \\ \gamma &\rightarrow fill \\ \gamma &\in \{0.5, 0.8, 0.95, 1\} \\ \text{STAT} &= slab \\ \text{SIDE} &= both \end{aligned}$$



Helske describes these violin-interval plots as "more challenging to create"; *ggdist* supports them naturally through a combination of its core

[15]Grammar of graphics aficionados might ask: why not use faceting instead of creating a new positional subscale? Experience has shown many situations where faceting doesn't cut it: for example, a *thickness* subscale easily handles complex plot layouts with multiple estimates as grouped or even possibly-overlapping densities (even combined with layers showing raw data)—combinations difficult or impossible to create with facets.

[16]This default is, most of the time, what you want, as it ensures each distribution integrates to 1. For exceptions to this rule, *ggdist* provides a `normalize` option and a `scale_thickness_shared` function which allow finer control over how thickness scales are shared across groups, panels, and geometries.

[17] I added this syntax when *ggplot2* deprecated `stat()`, which was a synonym for `after_stat()`. This (1) made expressions with `after_stat` needlessly verbose and (2) replaced a declarative verb, `stat`—which indicates the type of expression but not when it is computed—with a procedural verb, `after_stat`. I feel that this change violates the declarative foundations of *ggplot2*, and the mini-DSL for probabilistic expressions is my small protest against it.

[18]Just as an implementation note, this happens to be an incredible pain in the ass. See the description of `geom_slab` rendering in Sec. 5.

[19]This can also be done by discretizing the CDF, and that was the recommended approach in earlier versions of *ggdist*; see the `cut_cdf_qi` function. However, that does not generalize to other kinds of intervals, like highest-density intervals—hence the approach described here.

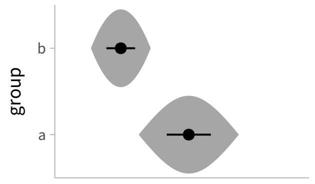
features: making interval mass (γ) into data and allowing distribution functions and intervals to be mapped to aesthetics within a single slab.

Another variation on violin plots is the *raindrop plot* of Barrowman and Myers [4], which maps log-density instead of density onto *thickness* inside a desired interval, say 95%:

$$\left.\begin{array}{ll} \log f_M(x) & \text{if } \gamma \leq .95 \\ \varnothing & \text{otherwise} \end{array}\right\} \rightarrow \textit{thickness}$$

$$\text{STAT} = \textit{slab}$$

$$\text{SIDE} = \textit{both}$$



The rationale here is that some distributional features, such as fat tails (kurtosis), can be easier to see in log-density than density [4]. In *ggdist*, the above spec is written:

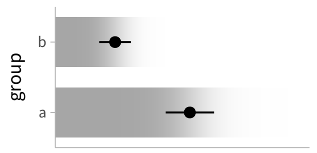
```
aes(thickness = ifelse(.width <= .95, log(!!p_(x)), NA)) +
stat_slab(side = "both", normalize = "groups")
```

Where `normalize = "groups"` is needed to tell the slab to normalize thickness on a per-group basis, as log-density does not have a natural zero point. Without this, the endpoints of the arcs defining the "raindrops" may not reach the *thickness* baseline.

Another interesting[20][21] uncertainty visualization is Haber and Wilkinson's [16] *fuzzygram*, i.e. *fuzzy histogram*. We can generalize it to a *fuzzy bar chart*. This chart type has what I would call a compelling *generative story*: an explanation for how the uncertainty encoding in the chart arises based on a generative process. Imagine we want to add uncertainty to a bar chart, and we have a distribution for the uncertainty in the value of each bar. Now say we sample a large number of semi-transparent bar charts from these distributions, and plot them all on top of each other. As the number of charts approaches ∞ and the opacity of any given chart goes to 0, the stack of overlapping bars for each value will begin to resemble the complementary CDF, $1 - F(x)$, of that distribution.[22] That insight leads to the following encoding:

$$1 - F_M(x) \rightarrow \textit{alpha}$$

$$\text{STAT} = \textit{slab}$$



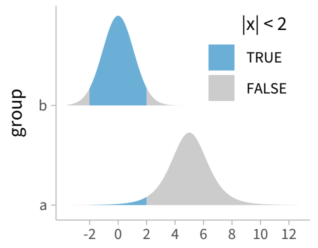
I would describe this chart type as theoretically interesting but interpretationally problematic. Anecdotally, without the overlaid reference point and interval, folks I show this to often do not correctly identify the mean, believing the darker parts of the bar are more likely (a not-unreasonable misinterpretation).[23] On the other hand, I do like the idea of *generative stories* in uncertainty visualization; e.g., I've used Plinko boards to depict uncertainty in election forecasts, which try to capitalize on a physical process—one that might intuitively[24] feel random to people—to depict uncertainty through a form of *sedimentation* [23].

Another advantage to being able to map arbitrary data values onto slab aesthetics is that we can map the results of logical conditions onto those aesthetics. A common procedure in Bayesian estimation is to use regions of practical equivalence (ROPEs) [32]. Say you are interested in whether an estimate is "practically" equal to 0. You could define a ROPE of $0\pm$ some small effect size that you consider so close to 0 it is effectively indistinguishable from it. Then you can ask: *what is the probability the value is in the* ROPE*—i.e., practically equivalent to 0?* We can visualize this probability by highlighting the ROPE on a density:

$$f_M(x) \rightarrow \textit{thickness}$$

$$|x| < 2 \rightarrow \textit{fill}$$

$$\text{STAT} = \textit{slab}$$



$\Pr(|x| < 2)$—the probability the value is in a ROPE of ± 2—is directly encoded by the proportion of the slab highlighted in blue. Because logical conditions can be mapped directly onto aesthetics, we can use `aes(fill = after_stat(abs(x) < 2))` to create this chart.

3.4 Dotplots

These days, when every other uncertainty visualization paper is about *frequency framing* or *discrete outcome* displays—spaghetti plots [12, 33] hypothetical outcome plots [22, 27], quantile dotplots [14, 31] and so on—I would be remiss to write a whole uncertainty visualization toolkit that can't make one measly discrete outcome visualization.[25] The natural counterpart to the continuous encoding of the *slab* geometry is *ggdist*'s *dots* geometry, which can create Wilkinson dotplots [52] (for raw data)[26] and quantile dotplots (to show uncertainty).

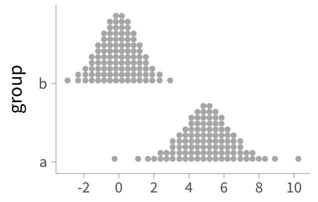
Quantile dotplots were designed for uncertainty communication, and depict quantiles of a continuous distribution using a dotplot, to help the viewer reason about that distribution as a set of discrete possible outcomes [31]. Put another way, they discretize the density function, and allow reasoning about intervals through counting: e.g., in a 100-dot quantile dotplot, $\Pr(X < 2)$ translates to the question, *how many dots out of 100 are less than 2?* Continuing our example from the *slab* geometries, here are two quantile dotplots using 100 dots each:

$$M \rightarrow x_{\text{DIST}}$$

$$\textit{group} \rightarrow y$$

$$\text{STAT} = \textit{dots}$$

$$\text{QUANTILES} = 100$$



It is worth noting that *ggplot2* has a *dotplot* geometry already, but that it is often awkward to use: as it uses a bin width specified by the user, there is no guarantee the dotplot fits inside the available space, often leading to dots running outside the plot region. By contrast, the *dots* geometry in *ggdist* uses numerical optimization to automatically find a shared bin width across all dotplots in the geometry that ensures they fit in the available space (notice I did not specify bin width above).[27]

One property of dotplots is the opportunity to encode additional information in the dots themselves. For example, we could take advantage of Unicode support in the *shape* aesthetic (which determines how points are drawn) to use emoji to encode just how we feel about possible effect sizes in a distribution:[28]

[20]AKA weird.

[21]Yes, there are a lot of footnotes. No, I don't care.

[22]Assuming additive blending and opacities that sum to 1.

[23]I don't know if this is an example of within-the-bar bias [38] or not.

[24]As this is an uncertainty visualization paper, I am professionally obligated to use "intuitive" at least once without defining it.

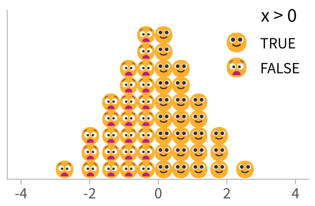
[25]Plus it would be a bit silly for me to develop a whole-ass visualization toolkit that can't at least make quantile dotplots, since I came up with them in the first place [31].

[26]And minor variations thereof, like *beeswarm* plots, which can be created by setting SIDE $=$ *both* and (optionally) adjusting the LAYOUT parameter; see the *dotsinterval* vignette for examples.

[27]It also allows constraints on minimum and maximum bin width, and has various options for what to do if those constraints are exceeded, such as compressing the layout so that dots may overlap. Anecdotally, automatic dotplot layout is one of the most popular features of *ggdist*, as it relieves the intense frustration of the endless tweaking cycle to get the dot size right.

[28]This example was inspired by a tweet from Michael Correll suggesting effect sizes at VIS might be best described with a dotplot of poop emojis. I was delighted to find that *ggdist* could replicate it, but have spared the literature a faithful rendering of the poopmoji plot by opting for smiley faces here.

$$\begin{aligned} M &\rightarrow x_{\text{DIST}} \\ x > 0 &\rightarrow \textit{shape} \\ \text{STAT} &= \textit{dots} \\ \text{QUANTILES} &= 50 \end{aligned}$$



This plot does require us to specify what the *shape* scale function is. Abstractly, if we assume each aesthetic mapping *foo* has a scale function s_{foo}, we might define the scale function for *shape* as:

$$s_{shape}(x) : x \mapsto \begin{cases} 😯 & \text{if } x \\ 😨 & \text{if } \neg x \end{cases}$$

In *ggplot2* this is:

```
scale_shape_manual(values = c("TRUE" = "😯", "FALSE" = "😨"))
```

This may seem manifestly silly, but there's something to the idea of making discrete outcomes more concrete—or at least more memorable—through graphic depictions, whether it be with emoji or other icons [18]. Since arbitrary fonts can be used with *ggplot2* shapes, it is one avenue to creating more expressive dotplots: for example, inspired by a tweet from Gabe Bassett, I once used Alberto Cairo's *Wee People* font [9] with `geom_dots` to create dotplots with anthropomorphic icons (Fig. 1C).

3.5 Further examples

My hope in the preceding sections was to whet your appetite for the expressive power of an uncertainty visualization grammar grounded in distributions, both of probability and of confidence. It was not intended as a tutorial (hence the paucity of code), but to give a taste for the underlying formalism of *ggdist*, how it can help us think about uncertainty visualizations, and how it extends the grammar of graphics in a principled way to support fluid specification of a variety of uncertainty visualizations. For more examples, and specifically for examples with code, the interested reader can check out the vignettes in the *ggdist* documentation, which include a slew of examples not covered here, and the vignettes in the *tidybayes* documentation, which include examples of the use of *ggdist* with Bayesian models.

4 Use in the Wild

I sometimes wonder if the best way to validate visualization ideas is to deploy them into the world and wait to see what happens (as Munzner puts it—to see what users do of their own accord [36]). Yet, given the vagaries of research project timelines, most systems in VIS do not have this luxury—understandably so, lest we wait 10 years to graduate a PhD student. While there are examples of wildly successful systems that came out of visualization research (e.g. *d3* [5], *Vega-lite* [44]), few systems have both the generality and reach of those exemplars. I think *ggdist* falls somewhere in the middle: it has been deployed for several years and enjoys a modest following. This gives me the opportunity to look at some coarse—but naturalistic—data on usage.

ggdist currently sees about 14,000 downloads per month from CRAN [13] (up from just a few thousand on original launch in 2020, piggy-backing on *tidybayes*' already-established community). Per Google Scholar, it has been cited 46 times (and *tidybayes* 236 times).[29] Its Github project has been starred ~620 times, putting it in the company of popular packages like *cowplot* (~640), *ggalt* (~630), and *ggtext* (~600) in the *ggplot2* extension gallery. It has 169 issues in its Github issue tracker, not counting the 305 issues in *tidybayes* (many of which are *ggdist*-related issues from before *ggdist* and *tidybayes* split). Issues especially can be a sign of user engagement, because projects without broader engagement will have issues opened only by the author. To better understand that engagement, I exported all issues from *ggdist* (see Github or the supplement) and read back through them, engaging in a light tagging process based on what I recalled about those issues.

[29]These are all citations of the software itself: neither package has had a paper written about it—yet.

Of the 169 issues on Github, 79 (47%) were opened by someone other than me; there were 45 unique authors, excluding me. A further 16 issues were opened by me in direct response to some user need, either flowing from a comment on another issue or from a conversation on another platform, often Twitter.[30] Thus, just over half of all issues on *ggdist*'s tracker stem from user engagement—and from a variety of users. I roughly categorized issues (with some overlap) as follows:

- 38%: A request that did (or would) result in a new feature.
- 20%: A user asking for help, usually with a particular plot they are trying to create.
- 17%: A bug.
- 16%: Internal issues, such as code refactoring, cleanup, or TODOs.
- 9%: Documentation.

Currently, 34 issues (20%) remain open; only one of these is a bug (recently-reported). I have been fortunate to get a wide variety of engagement from users, which has lead to substantive improvement to *ggdist*. It is hard for me to summarize that engagement, but I'll highlight two instances that had a salient impact on the overall design of *ggdist*, and which also carry some useful lessons for uncertainty visualization in the grammar of graphics.

First, issue #83 involves an extensive discussion about how to refactor *ggdist* to merge two classes of *stat*s. Older versions of *ggdist* distinguished between *stat*s designed to summarize distributions represented as vectors of samples (mapped onto x or y) and those designed to represent distribution objects (mapped onto a now-superceded *dist* aesthetic). My experience answering user issues led me to conclude that this distinction was confusing, so I opened this issue as an attempt to find a better solution.

Two expert users joined in the discussion, and together we considered a variety of options, including (1) creating a new *distribution* subtype of x and y scales, similar to the way that continuous values, discrete values, dates, and times are handled by *ggplot2*; or (2) creating new aesthetics, like x_{DIST} and y_{DIST}. I prototyped an implementation of the former, realizing some shortcomings: distributions cannot easily be treated as subtypes of positional scales like continuous or discrete variables are, because distributions themselves have subtypes (like being continuous or discrete). After further discussion, I settled on the x_{DIST} and y_{DIST} design, which also makes it easy to intermix distributional objects with non-distributional objects along the same positional scale. This has proven to be a good choice, as I have received feedback from users that the new x_{DIST} and y_{DIST} aesthetics are much easier to use.

Second, issue #19—to automatically detect discrete theoretical distributions and render them correctly as histograms—had been a longstanding wishlist item. About 9 months after I opened it, prompted by a conversation on Twitter, a *ggdist* user posted some examples of plots they might be able to create if the issue were resolved. After a productive brainstorming about the design of discrete distribution displays, I implemented a feature in *ggdist* whereby histograms of discrete theoretical distributions are treated as a type of density plot: a stepped version of their probability mass function is available through the same mappings used for density functions of continuous distributions. This integrates discrete distributions into the same broader framework used to construct other visualization types in *ggdist*, meaning I was able to resolve the issue without creating new special cases for handling discrete distributions from the users' perspective.

Several other issues and features have stemmed from interactions of this kind, including interactions with other R package authors who make use of *ggdist* (15 other packages, not including *tidybayes*, depend on *ggdist*). Overall, I think *ggdist* has garnered a modest following, and I owe a great debt to community for contributing to its development.

5 How does all this work, anyway?

Interested implementers might ask: *whither the implementation details?* The glib response, of course, is the *ggdist* source code, which is

[30]I spend possibly too much time on `#rstats` Twitter engaging with data scientists around visualization problems.

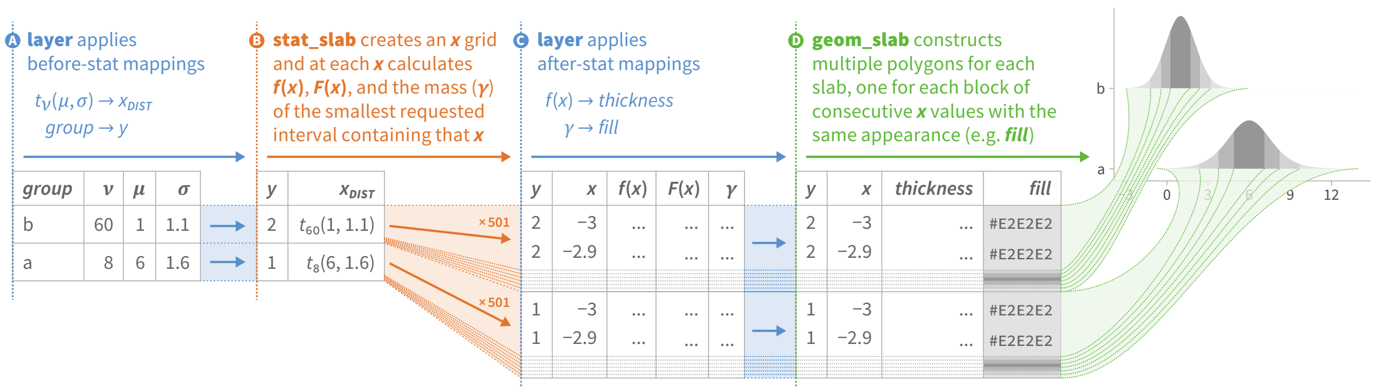

A layer applies before-stat mappings
$t_\nu(\mu,\sigma) \rightarrow x_{DIST}$
group → y
B stat_slab creates an x grid and at each x calculates f(x), F(x), and the mass (γ) of the smallest requested interval containing that x
C layer applies after-stat mappings
f(x) → thickness
γ → fill
D geom_slab constructs multiple polygons for each slab, one for each block of consecutive x values with the same appearance (e.g. fill)
group
ν
μ
σ
b
60
1
1.1
a
8
6
1.6
y
x_{DIST}
2
$t_{60}(1, 1.1)$
1
$t_8(6, 1.6)$
×501
×501
y
x
f(x)
F(x)
γ
−3
−2.9
...
thickness
fill
#E2E2E2
b
a
0
3
6
9
12

Fig. 2: Flow of data through the visualization construction pipeline for Helske [19]-style interval+density plot using `ggdist::stat_slab`. The `layer`, part of *ggplot2*, applies initial aesthetic mappings, translating input data into columns understood by `stat_slab`, which then constructs a larger data frame containing evaluations of distribution functions over an x grid for each distribution (the user can specify the grid size if needed; the default size is 501). Then, `layer` applies a second set of aesthetic mappings, and `geom_slab` uses the final data frame to construct the plot.

well-tested (with near-100% test coverage), available on Github, and archived on Zenodo. The less glib response is: this section of the paper.

Space being artificially scarce,[31] I'll outline the data processing pipeline (Fig. 2) for just the *slab* statistic/geometry; the others follow similar principles. The basic implementation of slabs relies on two entities: `stat_slab` and `geom_slab`. Both of these objects are contained inside a *ggplot2* `layer`, which amongst other things is responsible for applying aesthetic mappings and scale functions. For more details on how *stat*s, *geom*s, and *layer*s work in *ggplot2*, see Wickham [48].

To render a *slab* layer, first, the `layer` applies an initial set of aesthetic mappings (Fig. 2A), defining (at least) positional variables like x/y and distribution mappings like x_{DIST}/y_{DIST}. Then, `stat_slab` (Fig. 2B) evaluates the distributional functions (the density and CDF) and intervals for each distribution in the x_{DIST} (or y_{DIST}) aesthetic. It determines reasonable limits within which to draw each distribution (the exact limits if finite, and the 0.001th or 0.999th quantile by default if infinite) and creates a grid of x (or y) values to evaluate the functions at. Several special cases must be handled, including constant distributions (rendered as a point mass) and discrete distributions (as a histogram; see the end of Sec. 4). It then creates a data frame containing values of x, $f(x)$ (in a column named `pdf`),[32] $F(x)$ (`cdf`), and γ (`.width`)—the mass of the smallest interval containing x. These new columns are called *computed variables* in *ggplot2*.

The `layer` then applies a second set of aesthetic mappings (Fig. 2C): those which depend on computed variables (as indicated by wrapping them in `after_stat()` or by using the probability expression mini-DSL; see Footnote 17). Then, `geom_slab` (Fig. 2D) renders each slab using one of two algorithms, depending on its `fill_type` parameter: `segments` or `gradient`. Prior to R 4.1, the R graphics engine did not have proper gradient support, so the only available algorithm was `segments` (depicted in Fig. 2D), which sub-divides slabs into blocks of consecutive x (or y) values with the same appearance, interpolating x, y, and *thickness* values at cutpoints between blocks. This algorithm remains well-suited for fills with sharp cuts, such as ROPEs or small numbers of intervals. Alternatively, as of R 4.1, with proper gradient support in some R graphics output formats (e.g. SVG and PDF) [37], *ggdist* can output high-quality color gradients. The `gradient` algorithm does this by rendering each slab as a polygon with a linear gradient fill, using the x (or y) positions as control points on the gradient. This algorithm is best-suited to color gradient density plots.

In support of the above pipeline, *ggdist* includes extensive utilities for manipulating distributions (some of which are exposed in its API), including functions for calculating various interval types and for determining various distributional properties (e.g. detecting discrete or constant distributions and determining distribution limits). As the basic structure of the pipeline in *ggplot2* should be similar to other grammar of graphics systems, my hope is that such systems could easily adopt a formalism like the one I've described, allowing them to support a variety of uncertainty visualizations through a distributional approach.

[31] Perhaps one day we'll throw off the shackles of the IEEE.

[32] This is also where corrections to the density must be made, depending on the x (or y) scale function; see Sec. 6.3.

6 Reflections and Lessons Learned

6.1 Distributional notation makes uncertainty visualization much less annoying

The core use of distributional visualization to enable a variety of uncertainty visualization types was inspired by our earlier work developing a *Probabilistic Grammar of Graphics* (PGOG) [40], which tackled the problem of specifying area and unit visualizations of conditional probability distributions by integrating probability notation into the grammar of graphics. PGOG focused mainly on product plots [51], icon arrays [3], and dotplots [52], while *ggdist* expands the expressiveness of a distributional visualization syntax to cover visualization types that are not just functions of density and mass functions, but also functions of CDFs and intervals.[33]

A key insight from working on both PGOG and *ggdist* is that bringing notation for probability distributions into the grammar of graphics is a powerful, expressive way to create visualizations of distributions and uncertainty. Iterating on that syntax through user feedback (Sec. 4) has lead to, I think, an approachable but flexible abstraction for uncertainty visualization. A crucial insight—that frequentist uncertainty visualization can be brought into that same framework under the remit of *confidence distributions* [54]—frees us from multiple tyrannies: (1) endless battles about whether one should be a frequentist or Bayesian when you just want to get on with visualizing your uncertainty; (2) memorizing silly little formulas for this or that test statistic; (3) implementing different, mutually incompatible, and wholly brittle code paths for visualizing uncertainty under different statistical paradigms. A better world is possible!

6.2 Balancing abstraction and learnability

A continual source of tension in the design of *ggdist* has been the balance between abstraction and learnability. At its base, *ggdist* actually only has three geometries, all of which are composite *meta*-geometries: (1) a *slabinterval* geometry, consisting of a *slab*, *point*, and *interval*; (2) a *lineribbon* geometry, consisting of a *line* and *ribbon*; and (3) a *dotsinterval* geometry, consisting of *dots*, a *point*, and an *interval*. All other geometries in *ggdist* are *shortcut* geometries, constructed using some combination of default parameters and/or aesthetic mappings applied to one of those three *meta*-geometries.

In principle, *ggdist* could have only those three geometries; in practice, this would not be as usable. First, certain combinations of options

[33] On the other hand, *ggdist* still does not implement the conditional probability syntax we had in PGOG—though the mini-DSL for probability expressions was inspired by it—and it does not support product plots or stacked densities. In a way, *ggdist* has to be more conservative at integrating crazier ideas from research since it has actual users, and also, while the conditional syntax in PGOG is elegant for specifying a general class of probabilistic visualizations, its use cases for uncertainty specifically are a little less clear to me. So we'll see.

are often used together to create useful geometries; shortcuts for these save time. Second, many users will not delve into the depths of *ggdist*'s options to learn how to create each combination they need, so shortcuts aid learnability and discoverability. *ggdist*'s *shortcut* geometries include ones for intervals, eye plots (mirrored slabs with a *pointinterval*), complementary CDF plots, and gradient plots (See Fig. 1A). I have found users appreciate these shortcuts, and often ask for more: a raincloud plot shortcut (Fig. 1E) is a common request, for example.[34]

I have found it productive to hold off on implementing every requested shortcut. I usually wait until I can figure out how solving a particular problem (e.g., creating Helske-esque violin-interval plots) can be done within the distributional framework in a way that expands the expressiveness of *ggdist* (e.g., allowing interval masses to be mapped onto slab aesthetics). Meanwhile, extensive documentation and examples tempers the need for quite so many *explicit* shortcuts; an easy-to-adapt example in the documentation acts as a sort of shortcut itself. *ggdist* has extensive long-form *vignettes* that showcase myriad plot types with simple examples. This tension between abstraction and learnability is common to grammar of graphics toolkits: if the "purest" form of the grammar of graphics has no chart types at all (per Wilkinson [53]), but designers often think in terms of chart types [41], how do we best meet them in the middle?

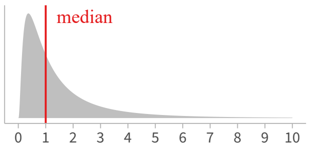
6.3 Uncertainty visualization is tightly coupled with grammar of graphics scales

As an earlier footnote alluded,[13] we must be very careful when visualizing density functions. A naïve implementation of `stat_slab` might simply pass x values through the density function of the distribution mapped to x_{DIST}. However, if a non-linear scale transformation is applied, this will result in incorrect densities. This is because, for a random variable $Y = g(X)$, the density function f_Y is:

$$f_Y(y) = f_X\left[g^{-1}(y)\right] \cdot \left|g^{-1\prime}(y)\right|$$

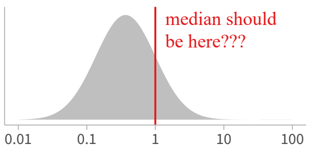
In other words, if we have a random variable X that we transform via the function g to get Y, we must adjust its density values by the factor $|g^{-1\prime}(y)|$, the absolute value of the derivative of the inverse of g. For example, if X is drawn from a log-Normal$(0,1)$ distribution, we could plot this distribution in base *ggplot2* by using `ggplot2::stat_function` combined with R's built-in log-Normal density function, `dlnorm`:

```
ggplot() +
  stat_function(
    fun = \(x) dlnorm(x, 0, 1)
  )
```



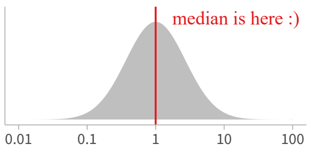
If we plot this on a log scale, i.e. plot $Y = \log(X)$, we should hope to see a Gaussian density with a median of $10^0 = 1$ (since medians are preserved under transformation). However, because *ggplot2* has no way to know this function is a density (nor if it did, does it know the derivative of the scale function), the resulting density will be incorrect:

```
ggplot() +
  stat_function(
    fun = \(x) dlnorm(x, 0, 1)
  ) +
  scale_x_log10()
```



The line at 1 does not divide the area into two regions with equal mass, as it would if it were the median. I have no doubt this has led to errors amongst *ggplot2* users.[35] Fortunately, *ggdist* does know how to correctly transform densities. It uses a combination of symbolic and (as a fallback) numerical methods to calculate derivatives of *ggplot2* scale transformations to adjust densities.[36] As a result, we can visualize a log-transformed log-Normal density and get the correct result:

```
ggplot() +
  aes(xdist = dist_lognormal(0,1))
  stat_slab() +
  scale_x_log10()
```



This emphasizes the need for uncertainty visualization systems to be *scale-aware*: it is not possible to comprehensively implement uncertainty visualization purely as a data pre-processing step.[37] This is one example of what Xiaoying Pu and I termed a *tight coupling* in a study of *ggplot2* users [41]: data pre-processing (calculating the density) is tightly coupled with the visualization specification (the scale transformation), and these must be kept in sync. Moving the calculation of densities into the visualization specification itself is one way to ensure this, eliminating a whole class of potential errors.

6.4 Where to go from here

I would hardly deign to pretend *ggdist* has solved all of uncertainty visualization. In truth, it's stuck to a well-defined corner of it: largely univariate uncertainty visualization, although *lineribbon* supports some multivariate chart types: besides being able to visualize many conditional distributions at once as ribbons, it can visualize joint uncertainty bands in the style of functional boxplots [26, 35, 46] by using it with the `curve_interval` function.[38] Obvious extensions include two-dimensional densities, for which support exists in *ggplot2* but which is not built around the same framework of distributional functions and objects that *ggdist* is. Thus, ripe for integration and extension.

Thinking further afield, there are other types of uncertainty visualizations not well-supported in *ggdist*, or which *ggdist*'s abstractions are not relevant to. One obvious example is spaghetti plots [12, 33]—though, if you already have a joint sample from a distribution of paths in a long-format data frame, *ggplot2* makes it trivial to visualize this. Similarly, animated hypothetical outcome plots (HOPs) [22, 27] are straightforward to construct using *ggplot2* with the *gganimate* package. A more interesting question might be: if one designed a new uncertainty visualization grammar from the ground up to support all of the visualizations in *ggdist* and PGOG, plus static sample-based visualizations like spaghetti plots and animated sample-based HOPs, what would it look like? Can a coherent framework bring all of these ideas together, and suggest new ideas too? I am hopeful it can.

7 Conclusion

ggdist has been a six-year journey in implementing a distributional, petty-statistics-camp-agnostic approach to uncertainty visualization in the grammar of graphics. While there remain many interesting future challenges to integrating further classes of uncertainty visualizations under one umbrella, the flexibility and expressiveness of *ggdist* thus far demonstrates the power of its underlying abstractions. Taking a step back, it also shows the value of continuing to push more aspects of visualization specification into the formal description of the visualization itself, both by enabling a wider range of visualization types to be easily created and by reducing the potential for certain classes of errors.[39]

[34]A request I am unlikely to fulfill, as the complexity of specifying options and aesthetics for such a bloated composite geometry—with *point*, *interval*, *slab*, and *dots* sub-geometries!—means the shortcut is unlikely to be much easier to use than just specifying a *slabinterval* alongside a *dots* geometry.

[35]For example, *ggplot2* issue #4783 is written by a user asking why a theoretical density and samples from a distribution do not line up under transformation—a less attentive user may never have noticed.

[36]Specifically, *ggdist* applies R's built-in `D` [42] function to get the symbolic derivative of the expression defining the scale function and, if that fails, uses `numDeriv::jacobian` [15]. In the future, if my pull request #341 to the *scales* package is accepted, derivatives of scale functions will be offloaded into the guts of *ggplot2* and simultaneously made more reliable.

[37]It also suggests that grammar of graphics systems should implement derivatives as part of their scale transformation functions.

[38]See examples at the end of the lineribbon vignette.

[39]Potential we also saw in reducing errors with incorrect normalization of conditional probability distributions in PGOG [40] or in keeping data transformation and visualization specification code in sync more generally in *ggplot2* [41].

Supplemental Materials

Supplemental materials, released under a CC-BY 4.0 license, are available on GitHub (github.com/mjskay/ggdist-paper) and archived on Zenodo (doi:10.5281/zenodo.7770984). Materials include: (1) an RMarkdown file that generates the figures in the paper; (2) generated figure images; (3) a tagged archive of *ggdist* Github issues; (4) an RMarkdown file calculating some descriptive statistics of *ggdist* Github isses; and (5) the full source of this paper.

The full source of *ggdist*, released under a GPL 3.0 license, is available on GitHub (github.com/mjskay/ggdist) and archived on Zenodo (doi:10.5281/zenodo.3879620). *ggdist* may also be installed from CRAN [21] via the R command `install.packages("ggdist")`.

Figure Credits

All figures were created by Matthew Kay and are released under a CC-BY 4.0 license.

Acknowledgments

I would like to thank various folks who have used, given feedback on, or contributed to *ggdist* over the years (this list is neither exhaustive nor in any particular order): Xiaoying Pu, Alex Kale, Abhraneel Sarma, Fumeng Yang, Hyeok Kim, Matti Vuorre, Dominique Makowski, TJ Mahr, Tim Mastny, Aki Vehtari, Brenton M. Wiernik, Arthur Albuquerque, A. Solomon Kurz, Gabe Bassett, Mitchell O'Hara-Wild, Steve Haroz, Jarrett Byrnes, Dylan H. Morris, Dmytro Perepolkin, Cédric Scherer, Isabella Ghement, Teun van den Brand, and Jonas Kristoffer Lindeløv. This work was supported in part by NSF award IIS-1910431.

References

[1] M. Allen, D. Poggiali, K. Whitaker, T. R. Marshall, and R. A. Kievit. Raincloud plots: a multi-platform tool for robust data visualization. *Wellcome open research*, 4, 2019. 1

[2] V. Amrhein and S. Greenland. Discuss practical importance of results based on interval estimates and p-value functions, not only on point estimates and null p-values. *Journal of Information Technology*, 37(3):316–320, 2022. 5

[3] J. S. Ancker, Y. Senathirajah, R. Kukafka, and J. B. Starren. Design features of graphs in health risk communication: a systematic review. *Journal of the American Medical Informatics Association*, 13(6):608–618, 2006. 8

[4] N. J. Barrowman and R. A. Myers. Raindrop plots: a new way to display collections of likelihoods and distributions. *The American Statistician*, 57(4):268–274, 2003. 1, 6

[5] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011. 7

[6] A. W. Bowman et al. Graphics for uncertainty. *JR Stat Soc Ser A Stat Soc*, 182(Pt 2):403–18, 2019. 1, 4

[7] P.-C. Bürkner, J. Gabry, M. Kay, and A. Vehtari. posterior: Tools for working with posterior distributions. *R package version 1.4.1*. https://mc-stan.org/posterior/, 2022. 4

[8] P.-C. Bürkner. brms: An R package for Bayesian multilevel models using Stan. *Journal of Statistical Software*, 80(1):1–28, 2017. doi: 10.18637/jss.v080.i01 4

[9] A. Cairo and S. Klein. Our font is made of people. *OpenNews – Source*. https://source.opennews.org/articles/our-font-made-people/, Feb 2018. 7

[10] E. F. Codd. *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc., 1990. 3

[11] M. Correll and M. Gleicher. Error bars considered harmful: Exploring alternate encodings for mean and error. *IEEE transactions on visualization and computer graphics*, 20(12):2142–2151, 2014. 1, 5

[12] J. Cox, D. House, and M. Lindell. Visualizing uncertainty in predicted hurricane tracks. *International Journal for Uncertainty Quantification*, 3(2), 2013. 6, 9

[13] G. Csárdi. cranlogs: Download logs from the 'rstudio' 'cran' mirror. R package version 2.1.1. https://CRAN.R-project.org/package=cranlogs, 2019. 7

[14] M. Fernandes, L. Walls, S. Munson, J. Hullman, and M. Kay. Uncertainty displays using quantile dotplots or cdfs improve transit decision-making. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pp. 1–12, 2018. 1, 6

[15] P. Gilbert and R. Varadhan. numderiv: Accurate numerical derivatives. R package version 2016.8-1.1. https://CRAN.R-project.org/package=numDeriv, 2019. 9

[16] R. N. Haber and L. Wilkinson. Perceptual components of computer displays. *IEEE Computer Graphics and Applications*, 2(03):23–35, 1982. 6

[17] W. Hadley. Tidy data. *Journal of Statistical Software*, 59(10):1–23, 2014. 3

[18] S. Haroz, R. Kosara, and S. L. Franconeri. Isotype visualization: Working memory, performance, and engagement with pictographs. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pp. 1191–1200, 2015. 7

[19] J. Helske, S. Helske, M. Cooper, A. Ynnerman, and L. Besancon. Can visualization alleviate dichotomous thinking? effects of visual representations on the cliff effect. *IEEE Transactions on Visualization and Computer Graphics*, 27(8):3397–3409, 2021. 1, 5, 8

[20] H. V. Henderson and P. F. Velleman. Building multiple regression models interactively. *Biometrics*, pp. 391–411, 1981. 2

[21] K. Hornik. The comprehensive r archive network. *Wiley interdisciplinary reviews: Computational statistics*, 4(4):394–398, 2012. 2, 10

[22] J. Hullman, P. Resnick, and E. Adar. Hypothetical outcome plots outperform error bars and violin plots for inferences about reliability of variable ordering. *PloS one*, 10(11):e0142444, 2015. 6, 9

[23] S. Huron, R. Vuillemot, and J.-D. Fekete. Visual sedimentation. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2446–2455, 2013. 6

[24] R. J. Hyndman. Computing and graphing highest density regions. *The American Statistician*, 50(2):120–126, 1996. 3

[25] C. H. Jackson. Displaying uncertainty with shading. *The American Statistician*, 62(4):340–347, 2008. 1, 4, 5

[26] J. L. Juul, K. Græsbøll, L. E. Christiansen, and S. Lehmann. Fixed-time descriptive statistics underestimate extremes of epidemic curve ensembles. *Nature physics*, 17(1):5–8, 2021. 9

[27] A. Kale, F. Nguyen, M. Kay, and J. Hullman. Hypothetical outcome plots help untrained observers judge trends in ambiguous data. *IEEE transactions on visualization and computer graphics*, 25(1):892–902, 2018. 6, 9

[28] M. Kay. Unifying names of output columns in bayesplot / tidybayes / etc. *The Stan Forums*. https://discourse.mc-stan.org/t/unifying-names-of-output-columns-in-bayesplot-tidybayes-etc/4577, Jun 2018. 4

[29] M. Kay. *ggdist: Visualizations of Distributions and Uncertainty*, 2023. R package version 3.3.0. https://CRAN.R-project.org/package=ggdist. doi: 10.5281/zenodo.3879620 1

[30] M. Kay. *tidybayes: Tidy Data and Geoms for Bayesian Models*, 2023. R package version 3.0.4. https://CRAN.R-project.org/package=tidybayes. doi: 10.5281/zenodo.1308151 1

[31] M. Kay, T. Kola, J. R. Hullman, and S. A. Munson. When (ish) is my bus? user-centered visualizations of uncertainty in everyday, mobile predictive systems. In *Proceedings of the 2016 chi conference on human factors in computing systems*, pp. 5092–5103, 2016. 1, 6

[32] J. K. Kruschke. Rejecting or accepting parameter values in bayesian estimation. *Advances in methods and practices in psychological science*, 1(2):270–280, 2018. 6

[33] L. Liu, L. Padilla, S. H. Creem-Regehr, and D. H. House. Visualizing uncertain tropical cyclone predictions using representative samples from ensembles of forecast tracks. *IEEE transactions on visualization and computer graphics*, 25(1):882–891, 2018. 6, 9

[34] Y. Liu, A. Gelman, and T. Zheng. Simulation-efficient shortest probability intervals. *Statistics and Computing*, 25:809–819, 2015. 3

[35] M. Mirzargar, R. T. Whitaker, and R. M. Kirby. Curve boxplot: Generalization of boxplot for ensembles of curves. *IEEE transactions on visualization and computer graphics*, 20(12):2654–2663, 2014. 9

[36] T. Munzner. A nested model for visualization design and validation. *IEEE transactions on visualization and computer graphics*, 15(6):921–928, 2009. 7

[37] P. Murrell. Vectorised pattern fills in r graphics. https://www.stat.auckland.ac.nz/ paul/Reports/GraphicsEngine/vecpat/vecpat.html, 2022. doi: 10.17608/k6.auckland.19945787 8

[38] G. E. Newman and B. J. Scholl. Bar graphs depicting averages are perceptually misinterpreted: The within-the-bar bias. *Psychonomic bulletin & review*, 19:601–607, 2012. 6

[39] M. O'Hara-Wild, M. Kay, and A. Hayes. distributional: Vectorised probability distributions. R package version 0.3.1. https://CRAN.R-

project.org/package=distributional, 2022. 4

[40] X. Pu and M. Kay. A probabilistic grammar of graphics. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1–13, 2020. 8, 9

[41] X. Pu and M. Kay. How data analysts use a visualization grammar in practice. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pp. 1–13, 2023. 9

[42] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2022. 9

[43] A. Satyanarayan, B. Lee, D. Ren, J. Heer, J. Stasko, J. Thompson, M. Brehmer, and Z. Liu. Critical reflections on visualization authoring systems. *IEEE transactions on visualization and computer graphics*, 26(1):461–471, 2019. 2

[44] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350, 2016. 1, 7

[45] D. J. Spiegelhalter. Surgical audit: statistical lessons from nightingale and codman. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 162(1):45–58, 1999. 1

[46] Y. Sun and M. G. Genton. Functional boxplots. *Journal of Computational and Graphical Statistics*, 20(2):316–334, 2011. 9

[47] Tidyverse Team. Dot prefix. *Tidyverse Design Guide*. https://design.tidyverse.org/dots-prefix.html, 2020. 4

[48] H. Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010. 1, 3, 8

[49] H. Wickham. ggplot2. *Wiley interdisciplinary reviews: computational statistics*, 3(2):180–185, 2011. 1

[50] H. Wickham. *Advanced R*. CRC press, 2019. 5

[51] H. Wickham and H. Hofmann. Product plots. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2223–2230, 2011. 8

[52] L. Wilkinson. Dot plots. *The American Statistician*, 53(3):276–281, 1999. 6, 8

[53] L. Wilkinson. *The grammar of graphics*. Springer, 2012. 1, 9

[54] M.-g. Xie and K. Singh. Confidence distribution, the frequentist distribution estimator of a parameter: A review. *International Statistical Review*, 81(1):3–39, 2013. 3, 8