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

Matthew Kay

Fig. 1: In the Clouds: Vancouver from Cypress Mountain. Note that the teaser may not be wider than the abstract block.

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 scarce. 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 implementation, 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) of geometries, 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 a retrospective 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 experiences with *ggdist* might provide a catalyst for further improvements to formalizations and implementations of uncertainty visualization across grammar of graphics ecosystems.
A free copy of this paper and all supplemental materials are available at https://OSF.IO/2NBSG.

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

1 Introduction

The grammar of graphics [26] has become ubiquitous, providing the foundation for a variety of visualization toolkits. Yet support for uncertainty visualization in grammar graphics implementations, generally speaking, remains rudimentary. Popular implementations like *ggplot2* [22, 23] and *Vega-lite* [19] 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 [6, 13], gradient error bars [4], gradient plots [3, 10], fan charts [10], and innumerable variations on eye plots [2, 8, 13, 20]. Yet, most of these alternative representations are—simply put—painful to convince existing grammar of graphics implementations to produce.

ggdist is an attempt to rectify this situation. It started under the guise of *tidybayes*—an R package I wrote for post-processing Bayesian models for use with *ggplot2*—in about 2016. I published *tidybayes* to CRAN (the Comprehensive R Archive Network) in 2018, and it slowly gained some use in the Bayesian statistics community. However, the package had two complementary, but not always perfectly aligned, use cases: post-processing Bayesian model output for visualization, and uncertainty visualization in the grammar of graphics. The latter is bigger than *just* Bayesian statistics: everyone needs to visualize uncertainty! And, contrary to popular opinion—as I'll demonstrate later—Bayesian and frequentist uncertainty visualization can in fact be done within the same framework. Recognizing this broader need, in 2020 I spun off the uncertainty visualization components of *tidybayes*

• *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

into a new package, *ggdist*, and published it to CRAN. 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" for an R package).

Over the course of its continuing development, I have learned a lot about how to integrate uncertainty visualization into the grammar of graphics, and developed a formal way of describing that integration (more on that later). In the spirit of recent retrospectives on visualization system design **CITE**, I'd like to distill down some of what I've learned here.[1] Ultimately, what would be nice to have—and what I hope to demonstrate *ggdist* to be—is a coherent extension to the grammar of graphics that makes it easy to create the various proposed alternative uncertainty visualizations from the literature, *and more*. By "and more" I mean: not only should we be able to express a variety of uncertainty visualizations through a single coherent framework, but that framework should be complete enough that someone else can wander along and express new uncertainty visualization types I've never thought of before (emphasis on *I*—I think a formalism and its corresponding API truly shows its power when people make things with it that its creator has never thought of before). I'll do this by grounding the design of *ggdist* in a formal representation of uncertainty through mappings of functions of distributions onto visual channels or *aesthetics*. As time is a straight arrow, I can't provide direct evidence that there are visualization types I've never seen before that can be created with *ggdist*, but I will describe instances in which I have encountered uncertainty visualizations in the literature and subsequently realized they were straightforward to express in *ggdist*.

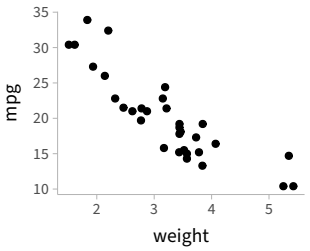
Ultimately, given the huge expressive power of the grammar of graphics and the popularity of tools built on it, I hope a catalog of my experiences here might provide a catalyst for improved implementations of uncertainty visualization to flourish in existing grammar of graphics ecosystems. Let's give it a try.

2 Setting the Stage

2.1 A simplified notation for the grammar of graphics

To be able to talk more generally than a specific grammar of graphics implementation, we'll at least need a formal way of writing down visualization specifications separated from a particular implementation. I'll adopt here a notation that I've found works pretty well when I teach the grammar of graphics to undergraduates. I've used a variation on this notation when teaching *ggplot2*, *Vega-lite*, *Altair*, and *Tableau* and found students pick it up easily, which is at least some evidence it might be easy for folks to understand. The core notation describes a visualization in terms of its *data variables*, *aesthetic mappings*, and *geometries*. We create a scatterplot, for example, by mapping one variable onto the x aesthetic and another onto the y aesthetic:[2]

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



This notation sets up two *aesthetic mappings*:[3] a mapping from the *weight* data variable onto the x aesthetic, and a mapping from the *mpg* data variable onto the y aesthetic. It then employs a *point* geometry[4] 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 set up log scales, to pick how colors are assigned to data values, 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, which gives an incorrect intuition in my view. In fact, let's see that now: here is a translation of the above into *ggplot2* code, assuming `cars` is a data frame with `weight` and `mpg` columns:

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

Or in *Vega-lite*:[5]

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

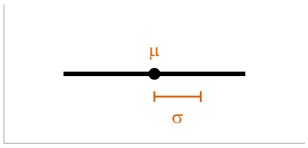
This shows the close correspondence between the abstract notation above and the particulars of code in actual grammar of graphics implementations. Throughout the rest of this paper, I'll stick to the abstract notation and corresponding *ggplot2* + *ggdist* code for the particulars.

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 assume a Gaussian approximation: to represent all estimates and their uncertainty as a mean and standard deviation. This makes the specification problem 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 *pointinterval* geometry as follows, yielding a 95% interval calculated from a Normal distribution with mean μ and standard deviation σ:

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



This is one approach taken by *Vega-lite*: it provides an `errorbar`[6] mark (analogous to *pointinterval*) and an `xError` channel (analogous to x_{SD}). I'll refer to this notation 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

[1] 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.

[2] 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.

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

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

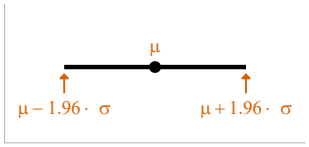
[5] I use the Vega-lite API instead of its JSON form, as JSON is a horrifying mess of visual noise that no sane human should want to read or write. The Vega-lite API is a notable improvement, though without R's facility for capturing and re-writing abstract syntax trees, it doesn't quite reach the succinctness 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.

[6] `errorbar` is, in my view, a misnomer, as is `xError`: fundamentally, the mark calculates a Gaussian interval, which might be for a distribution 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.

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 endpoints of the intervals; e.g. for a 95% Gaussian interval:

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



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 seem to 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

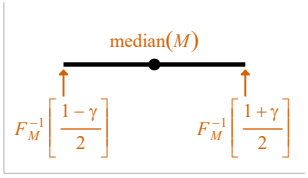
Imagine we represent an uncertain value generically as a distribution, or a random variable, M. Importantly, I do not consider this a *probability* distribution necessarily: it could be a probability distribution, but it could also be a *confidence* distribution, which is a frequentist generalization of sampling distributions and bootstrap distributions **CITE**. 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 γ% confidence interval, $[-\infty, x]$, for M. That sentence is pretty typical of convoluted frequentist definitions, so it might be easier to think in terms of the inverse: $F_M^{-1}(\gamma)$ yields the value x, which is the upper limit of a one-sided γ% confidence interval on M: $[-\infty, x]$. So $[-\infty, F_A^{-1}(0.95)]$ is a one-sided 95% confidence interval on 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.

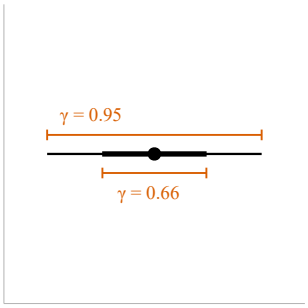
For example, a median and γ% quantile interval could be defined generically on any distribution M as follows:

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



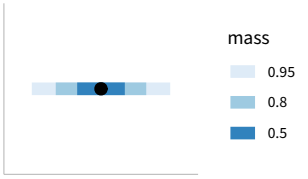
If M is a probability distribution, this is a Bayesian *credible* interval, and if M is a confidence distribution, this is a frequentist *confidence* interval. This lets us abstract over the petty battles between this or that statistical camp and get to the meaningful business of visualizing uncertainty. 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[7] wanted to show two intervals, a 95% and a 66%, where the smaller interval is shown as a thicker line, we could write:

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



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*.[8] 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*:

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



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 **CITE** and shortest intervals **CITE**, 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. All *stat*s 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*.

[7] 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.

[8] 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* **CITE** (distinctions between which I have long forgotten) or what some statisticians call *tidy data* **CITE**; 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.

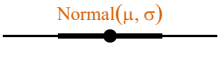
This changes the specification to something like:

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

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 point estimates and intervals are calculated.

In implementation, *ggdist* allows distributions to be represented by numeric vectors (sample-based representation), objects from the *distributional* R package (which supports theoretical distributions), and `rvar` objects from the *posterior* R package (a sample-based representation that mimics numeric arrays in R). For example, if we re-create the $\{x, x_{\text{SD}}\}$ representation abstractly thus:

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

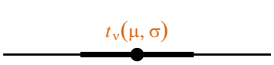


In *ggdist*, using `distributional::dist_normal`, the specification is quite similar:

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

These happen to be the default values for `point_interval` and `.width` (γ),[9] so just `stat_pointinterval()` also works here. To demonstrate generalizing this approach, consider the very common need of placing 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, scale μ (e.g. an estimated mean), and scale σ (e.g. a standard error). Given these three numbers in a data frame, a visualization specification might be:

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



Which in *ggdist* is:

```
ggplot(data) +
  aes(xdist = dist_student_t(nu, mu, sigma)) +
  stat_pointinterval()
```

Thus the code matches us closely to the abstract notation.

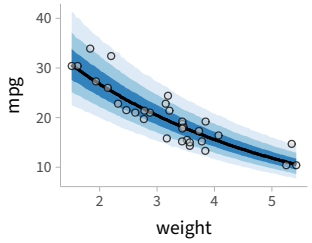
3.2 Ribbons

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):

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

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:

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



I include the raw data as a separate *point* geometry layer as well, for comparison. Assuming `fit` is a Bayesian version of such a model fit using the *brms* modeling package in R, and `preds` is a data frame of desired *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` **CITE** and the `posterior::rvar` data type **CITE**. The latter is a data type I created[10] 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 = preds |> mutate(
  mpg_given_weight = rvar(posterior_predict(fit, preds))
)
```

Given this data frame, the equivalent of the abstract specification above is:

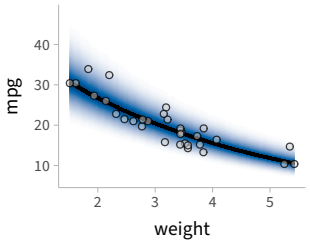
```
ggplot(preds) +
  aes(x = weight, ydist = mpg_given_weight) +
  stat_lineribbon()
```

`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 the `fill` mapping for this example. Once we have a multiple-ribbon geometry, it is easy to create other uncertainty visualization types, like gradient fan charts **CITE**. For example, we could use a large number of intervals, say $k = 50$ or 100, with masses between 0 and 1 (exclusive):

[9] For historical reasons that have to do with a combination of a very long discussion with a bunch of people on the Stan forums [12] and naming conventions in some R APIs for fixed arguments to functions with variable argument lists [21], γ in *ggdist* is spelled `.width`. Reflecting on my past mistakes, a better name would be `mass`.

[10] This was a slightly insane idea in itself, given the way that base R data types work. As R's classes work based on generic *functions*, rather than classes with a well-defined set of methods, this requires tracking down all the various functions implemented to work with arrays and vectors in the language and providing implementations for them for random variable arrays. Indexing functions especially are annoying, as there are half a dozen different ways to index into R arrays, each with myriad corner cases. Anyway.

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



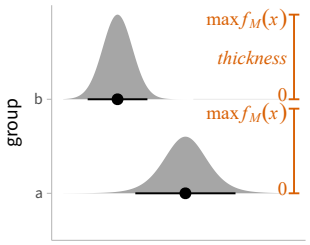
This set of k γ values is the same sequence generated by the `ppoints(k)` function in R, so we can pass `.width = ppoints(100)` to `stat_lineribbon` to get a gradient fan chart with 100 intervals. This stems directly from the choice to make γ into data that can be mapped onto aesthetics, and is one example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

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 have an implementation of 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.[11]

However, stopping at *just* densities seems the wrong level of abstraction:[12] 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 *group*s, *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:

$$M \to x_{\text{DIST}}$$
$$group \to y$$
$$f_M(x) \to thickness$$
$$\text{STAT} = slab$$



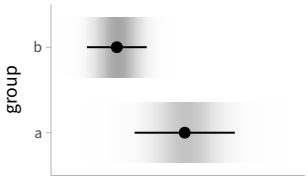
I also include a *pointinterval* with a median and 95% interval above (and elsewhere in this section) for reference. The *slab*s have a subscale for *thickness*, which is an orientation-aware aesthetic (if the distributions were mapped onto y_{DIST}, *thickness* would act as a width instead of height), 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.[13]

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

1. `aes(thickness = after_stat(pdf))`: This says to map the `pdf` *computed variable* onto *thickness*. Computed variables in *ggplot2* are calculated by *stat*s and made accessible in aesthetic mappings by wrapping the mapping specification in `after_stat()`.
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 $p(x) \to thickness$ in code. The `!!` pseudo-operator comes from the *rlang* meta-programming R package, and performs *unquotation* **CITE**: it inserts the expression returned by `p_(x)`, which in this case is `after_stat(pdf)`, into the `aes` call.[14]

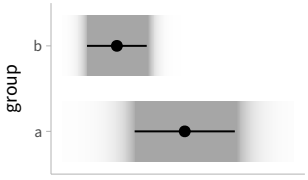
Slab geometries have several useful properties which help make them useful for creating a variety of uncertainty visualizations. One important property is that the *alpha* (opacity), *fill*, and outline *color* aesthetics of the slabs can have data values mapped onto them at a sub-geometry level.[15] 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 **CITE**, by mapping density onto *alpha*:

$$M \to x_{\text{DIST}}$$
$$group \to y$$
$$f_M(x) \to alpha$$
$$\text{STAT} = slab$$



Which translates naturally to `aes(alpha = !!p_(x))`. Fine fine, but let's get *weird*. Back in 2014, the inimitable Michaels Correll and Gleicher wrote that error bars should be considered harmful [4], and 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*:

$$-\max\begin{bmatrix}|2F_M(x)-1| \\ 0.95\end{bmatrix} \to alpha$$
$$\text{STAT} = slab$$



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. **TODO: mention consonance curves** But let's get *weirder*: in 2021, Helske et al [8], inspired by Correll and Gleicher, proposed combining this 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*

[11] You may be tempted to say: but Matthew, of course you could easily pre-calculate densities from a theoretical density function and plot them. 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. 5.4.

[12] See earlier footnote[6] about *error bars* versus *intervals*.

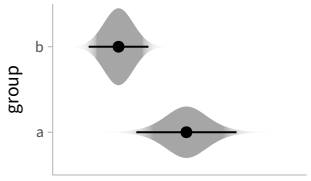
[13] This default is, most of the time, what you want, as it preserves the property that each distribution integrates to 1. For exceptions to this rule, *ggdist* provides a `normalize` option and a `scale_thickness_shared` function which allow finer control over how thickness scales are shared across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

[14] I added this syntax relatively recently, when *ggplot2* deprecated `stat()`, which used to be 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 at play but not when it is computed—with a procedural verb, `after_stat`. I feel that this name change violates the declarative foundations of *ggplot2*, and the mini-DSL for probabilistic expressions in *ggdist* is my small protest against it.

[15] Just as an implementation note, this happens to be an incredible pain in the ass. Prior to R 4.1, the R graphics engine did not have proper gradient support, so this involved manually sub-dividing geometries and interpolating thickness values at cutpoints between contiguous blocks of color. Now, with proper gradient support in some R graphics output formats (e.g. SVG and PDF), *ggdist* can output high-quality color gradients, and as of this writing is one of the only R packages to do so (even base *ggplot2* has not implemented it yet).

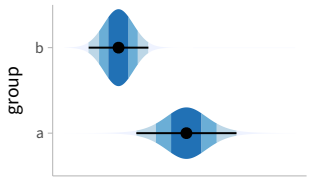
side, or *both*. I'll also adjust the tails to fade outside the 85% interval, since otherwise the fading is hard to see in the skinny tails of the violin:

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



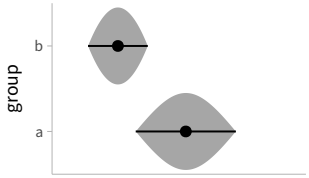
Okay fine, but let's get *even weirder*: Helske et al [8] further suggested using discrete intervals for the colors instead of a gradient, to aid discriminability. We can do that too! A key feature of the *slab* stat is that it *also* computes intervals, and for each point along the slab, retains a column indicating the mass (γ) of the smallest requested interval containing that point. This means that we can integrate intervals directly into the slab by mapping γ onto *alpha* or *fill*:

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



Helske describes these violin-interval plots as "more challenging to create"; *ggdist* supports them naturally through a combination of its features not *specifically* designed to create these plots. Another variation on violin plots is the *raindrop plot* of Barrowman and Myers [2], which maps log-density instead of density onto *thickness* inside a desired interval, say 95%:

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



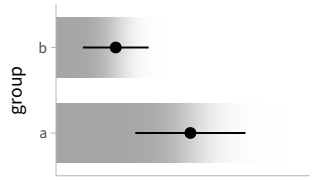
The rationale here is that some distributional features, such as fat tails (kurtosis), can be easier to see in log-density than density [2]. In *ggdist*, the above spec is written:

```
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[16][17] uncertainty visualization from the literature is Haber and Wilkinson's [7] *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 you can give for how the uncertainty encoding in the chart arises based on some generative process. Imagine we want to add uncertainty to a bar chart, and we have a distribution for the uncertainty in the value encoded in each bar. Now say we sample a large number of semi-transparent bar charts from these distributions, and overlay 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.[18] That insight leads to the following encoding:

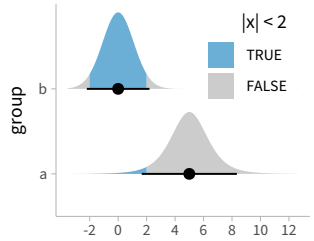
$$
\begin{aligned}
1 - F_M(x) &\rightarrow \textit{alpha} \\
\text{STAT} &= \textit{slab}
\end{aligned}
$$



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 of the distribution, believing the darker parts of the bar are more likely (a not unreasonable misinterpretation).[19] On the other hand, I do like the idea of *generative stories* in uncertainty visualization; for example, I've used Plinko boards to depict uncertainty in election forecasts **CITE**, which try to capitalize on a physical process—one that might intuitively[20] feel random to people—to depict uncertainty through a form of *sedimentation* **CITE**.

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 the Bayesian estimation community is to use regions of practical equivalence (ROPEs) [14]. Say, for example, you are interested in whether an estimate is "practically" equal to 0; you would define a ROPE around 0 of $\pm$ some small effect size within which you would consider the value so close to 0 it is effectively indistinguishable from it. Then you can ask questions like, what is the probability the value is practically equivalent to 0? We can visualize this probability by highlighting the ROPE on a density:

$$
\begin{aligned}
f_M(x) &\rightarrow \textit{thickness} \\
|x| < 2 &\rightarrow \textit{fill} \\
\text{STAT} &= \textit{slab}
\end{aligned}
$$



Here, $\Pr(|x| < 2)$—the probability the value is in a ROPE of ± 2—is directly encoded by the proportion of the area under the curve highlighted in blue. In *ggplot2*, logical conditions can be mapped directly onto aesthetics, so in *ggdist*, we can use `aes(fill = after_stat(abs(x) < 2))` to create the above chart.

3.4 Dotplots

These days, when every other uncertainty visualization paper is about *frequency framing* or *discrete outcome* representations—spaghetti plots [5, 15] hypothetical outcome plots [9, 11], quantile dotplots [6, 13] and the like—I would be remiss to write a whole toolkit for uncertainty visualization that can't make one measly discrete outcome uncertainty visualization.[21] The natural counterpart to the continuous encoding of the *slab* geometry is a*ggdist*'s *dots* geometry, which is designed to create Wilkinson dotplots [25] (on raw data)[22] and quantile dotplots.

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 [13]. Continuing our example from the *slab* geometries, here is two quantile dotplots using 100 dots each:

[16] AKA weird.

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

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

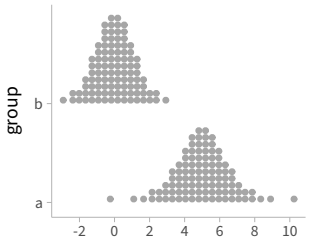
[19] I don't know if this is an example of within-the-bar bias [16] or not.

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

[21] Plus it would be a little silly of 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 [13].

[22] And minor variations thereof, like *beeswarm* plots.

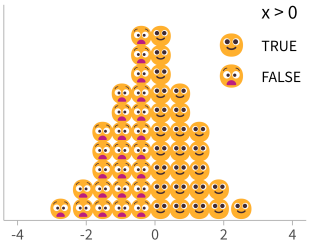
$M \rightarrow x_{\text{DIST}}$
$group \rightarrow y$
$\text{STAT} = dots$
$\text{QUANTILES} = 100$



It is worth noting that *ggplot2* has a *dotplot* geometry already, but that it is often awkward to use: because it uses a bin width specified by the user, there is no guarantee the dotplot will fit inside the available space, often leading to stacks of 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).[23]

One property of dotplots is the opportunity to encode additional information in the dots themselves. A trivial example in *ggdist* is to use emoji to encode just how you feel about possible effect sizes in the distribution:[24]

$M \rightarrow x_{\text{DIST}}$
$x > 0 \rightarrow shape$
$\text{STAT} = dots$
$\text{QUANTILES} = 50$



This plot does require specifying more particularly 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:

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

In *ggplot2* this is:

```
scale_shape_manual(
  values = c("TRUE" = "\textbf{TODO}", "FALSE" = "\textbf{TODO}")
)
```

This may seem manifestly silly, but there's something to the idea of making discrete outcomes more concrete or at least memorable through graphic depictions, whether it be with emoji or other icons **CITE**. Since arbitrary fonts can be used with *ggplot2* shapes, it is one avenue to creating more expressive dotplots: for example, I have used Alberto Cairo's *Wee People* font **CITE** to create dotplots with anthropomorphic icons. **CITE**

3.5 Combinations

TODO USGS, Cedric's slab+interval, raincloud, logit dotplots (plus mention that's in *see*).

3.6 Further examples

My hope in the preceding sections was to whet your appetite for the possibilities of an uncertainty visualization grammar grounded in distributions. I have attempted to give a taste for why ggdist is structured the way it is, its underlying formalism, and how it extends the grammar of graphics in a principled way to support fluid specification of a variety of uncertainty visualizations. What I've shown only scratches the surface of what *ggdist* can do: the interested reader (hopefully you!) can check out the vignettes in the *ggdist* documentation, which include a slew of examples not covered here. The *tidybayes* documentation is also worth checking out, as it includes more examples of the use of *ggdist* with Bayesian models.

4 Use in the Wild

TODO maybe: use in Cedric scherer tutorials, USGS, tidytuesday, ... ?

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 **CITE**). Yet, given the vagaries of research project timelines, most systems in VIS do not have this opportunity—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* **CITE**, *Vega-lite* **CITE**), few systems have both the generality and reach of those exemplars. *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 enjoys a modest 14,000 downloads per month from CRAN **CITE** (up from just a few thousand on original launch in 2020, piggy-backing on *tidybayes*' already-established community at the time). Per Google Scholar, it has been cited 46 times (and *tidybayes* 236 times).[25] 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. On Github, *ggdist* has 169 issues in its issue tracker, not counting the 305 issues in *tidybayes* (of which many issues prior to *ggdist* splitting from *tidybayes* were *ggdist*-related). 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.

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 I had with a user on another platform, often Twitter. [26] Thus, the majority of issues on *ggdist*'s tracker stem from user engagement with the package, and from a variety of users. I roughly categorized the types of issues (with some overlap) as follows:

- 38%: A feature request or an issue 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 textsctodo-tracking.
- 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 over all of these issues, but I'll highlight two instances that have had a salient impact on the overall design of *ggdist*, and which have some useful lessons for uncertainty visualization in the grammar of graphics.

[23] It also allows constraints on minimum and maximum bin width, and has various options for what to do if those constraints may be 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 that endless tweaking cycle to get the dot size right.

[24] 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 that idea, but have spared the literature a faithful rendering of the poopmoji plot by using smiley faces here.

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

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

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 creating a new *distribution* subtype of x and y scales (the way that continuous values, discrete values, dates, and times are handled), or creating new aesthetics, like x_{DIST} and y_{DIST}. We weighed pros and cons, and 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). In addition, other non-distribution variables must be able to share the same positional scale as the distributional variables (so, e.g., an x scale cannot be given a *continuous distribution* subtype, it must have a *continuous* subtype). After further conversation, I settled on the x_{DIST} and y_{DIST} design, which 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 noticed users struggle less with the new design: previously, the *dist* aesthetic would implicitly be mapped to the relevant positional scale depending on the orientation of the geometry, and this lead to confusion when this mapping was not as a user intended.

Second, issue #19—to automatically detect discrete theoretical distributions and render them correctly as histograms—had been a long-standing wishlist issue. 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. This lead to a productive back-and-forth around the design of discrete distribution displays, and resulted in 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*.

Both of these examples demonstrate how the broader *ggdist* community has engaged with the software and contributed productively to its improvement. Several issues and features have stemmed other R package authors using *ggdist* (15 other packages, not including *tidybayes*, depend on *ggdist*). Overall, I think *ggdist* has garnered a modest following, including numerous highly-engaged users who have contributed meaningfully to its improvement.

5 Reflections and Lessons Learned

ggdist has been under development in some form or another for about six–seven years, and over that time has evolved through user feedback and several waves of refactoring and rewriting. Through all that, I'd hope I've learned a few things that may be useful for others tackling uncertainty visualization to consider.

5.1 Distributional notation makes uncertainty visualization much less annoying

TODO distributions as core to uncertainty vis, unifying Bayes/freq, power of the syntax itself (making parameters into data, vectorization of arguments to create distributions made the dist+args syntax obsolete, etc)

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) [17], 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 [24], icon arrays [1], and dotplots [25], 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.[27]

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 abstraction for uncertainty visualization. An additional insight—that frequentist uncertainty visualization can be brought into that same framework under the remit of *confidence distributions* [27]—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[28] 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!

5.2 Something about shortcut stats/geoms, user feedback, and balancing abstraction and learnability, good defaults

TODO raincloud, whwere to draw the line on composite geoms / shortcuts. histograms as slab types or as density estimators. maybe a section on how annoying KDEs are? — include omething about seeing incorrect vis in the wild, finding good defaults, etc (see also: default points and intervals as median/qi, move to a default KDE that estimates bounds and uses bounded estimator, etc).

Maybe something about the various bits of machinery for distributions: discrete/not, determining support, determining bounds when plotting automatically, etc.

5.3 faceting versus thickness

faceting too awkward, needing thickness for dodging, etc.

5.4 Uncertainty visualization is tightly coupled with grammar of graphics scales

As an earlier footnote alluded,[11] 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:

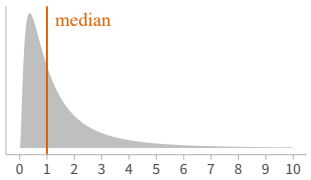
$$f_Y(y) = f_X\left(g^{-1}(y)\right) \cdot \left|g^{-1\prime}(y)\right|$$

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, consider a log-Normal variable X:

$$X \sim \text{log-Normal}(0, 1)$$

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)
  )
```

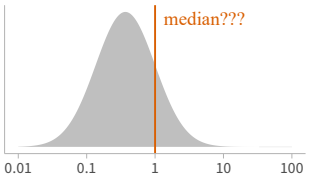


[27] On the other hand, *ggdist* still does not implement the conditional probability syntax we had in PGOG—though the mini-DSL for probability expressions was inspired by it—and it does not support product plots or stacked densities. In a way, *ggdist* has to be more conservative at integrating crazier 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.

[28] Pedants get one properly typeset "formulæ" per paper. There it was.

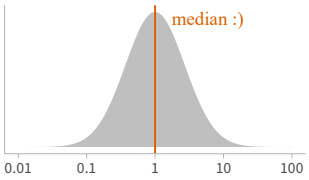
If we plot this on a log scale, we should hope to see a Gaussian density with a median of $10^0 = 1$ (since medians are preserved under transformation). However, because *ggplot2* has no way to know this function is a density (nor does it necessarily know the derivative of the scale function), the resulting 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 should if it were the median. I have no doubt this has led to errors amongst users of *ggplot2*.[29] Fortunately, *ggdist* does know how to correctly transform densities. It uses a combination of symbolic and (as a fallback) numeric differentiation to calculate derivatives of *ggplot2* scale transformations to adjust densities.[30] 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.[31] This is one example of what Xiaoying Pu and I[32] termed a *tight coupling* of data transformation and visualization specification in a study of expert *ggplot2* users [18]: the data processing (calculating the density) is tightly coupled with the visualization spec (the scale transformation), and these must be kept in sync. Moving the specification of densities into the visualization spec itself is one way to ensure this, eliminating a whole class of errors.

5.5 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 fairly well-defined corner of it: largely univariate uncertainty visualization. Through *lineribbon*s it does support some multivariate stuff: besides being able to visualize many conditional distributions at once as ribbons, it can visualize joint uncertainty bands in the style of functional boxplots **CITE** through the `curve_interval` function combined with `stat_lineribbon`.[33] 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 *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 particularly relevant to. One obvious example is spaghetti plots **CITE**—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) **CITE** 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.

[29] For example, issue #4783 in the *ggplot2* repository 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 might never have noticed.

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

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

[32] Let's be honest, Xiaoying did all the hard work.

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

6 Conclusion

See above.[34]

Supplemental Materials

All supplemental materials are available on Zenodo at **TODO** and on GitHub at github.com/mjskay/ggdist-paper, released under a CC BY 4.0 license. 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).

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, 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] 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

[2] 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

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

[4] 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

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

[6] 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

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

[8] 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, 6

[34] This final footnote is my small protest against the inane act of repeating the abstract as the conclusion to one's paper. For an abstract, see the abstract. For a hopeful message about where to go next, see the end of the previous section.

[9] 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

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

[11] 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

[12] 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

[13] 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

[14] 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

[15] 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

[16] 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

[17] 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

[18] 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

[19] 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

[20] 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

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

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

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

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

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

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

[27] 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. 8