EDITORS: Daniel F. Keefe, dfk@umn.edu
Melanie Tory, mtory@tableau.com

DEPARTMENT: PEOPLE IN PRACTICE

BubbleUp: Supporting DevOps With Data Visualization

Danyel A. Fisher, *Honeycomb, San Francisco, CA USA*

BubbleUp is a tool that lets DevOps teams—data analysts who specialize in building and maintaining online systems—rapidly figure out why anomalous data have gone wrong. We developed BubbleUp with an iterative, human-centered design approach. Through multiple rounds of feedback, we were able to build a tool that presents a paired-histogram view to help make high-dimensional data make sense.

The alarm pierces 3 A.M. sleep like a lightning bolt: somebody somewhere is having trouble using your service as they expect. As the human on-call, you need to evaluate the following.

- Who is affected (and how many)?
- How bad is the failure?
- What is *actually* wrong?

And you'd better do it fast. Do you wake up the rest of the troubleshooting team in the middle of the night, or can you go back to bed?

No pressure or anything.

The need to evaluate well and fast is the core premise of a group of tools generally called application performance management (APM). APM tools try to help DevOps teams—teams of developer-operators—understand the reliability of their online systems. At Honeycomb, our software product (also called Honeycomb) is one such APM tool: it supports DevOps teams in exploring complex instrumentation data from their distributed systems.

From the perspective of data visualization, DevOps work in a fascinating data analytics domain. They have deep domain knowledge of highly complex systems; they are responsible for both creating and analyzing a data stream dedicated to the task of monitoring and debugging distributed systems that are run on remote servers. The analytics challenges that they solve have impacts that can be measured in both dollars and hours of lost sleep. Most interestingly, because DevOps teams tend to repair bugs after finding them, each investigation is likely to be unique.

The data analytics tasks that DevOps carry out are familiar to the visualization research community, and the lessons that we learn from their work generalize well to other applications of data analytics. They are asking loosely structured questions of high-dimensional data and need to pursue analyses to solve complex problems.

This article discusses the design and development of BubbleUp. A core component of Honeycomb, BubbleUp exists to support DevOps. Its design is the result of working closely with our target users to understand their needs, iterating on the design, and then tracking the use of the tool over time. BubbleUp illustrates a way to help analysts navigate highly complex data; the process of working intensively with our target users helped us narrow down on a solution that would directly address their challenges.

EXAMPLE: A SLOW API

Figures 1–3 show a sample usage of BubbleUp. An operations team is responsible for handling an API that is exposed on the web; client applications call into it. This team is responsible for making sure that performance continues to run at satisfactory levels. They have been alerted that their system is handling some requests intolerably slowly. Fortunately, their system is well-instrumented, and so they can try to dig into their data to figure out what is wrong.

As shown in Figure 1, they issue a query in their dataset to get a heatmap of how long it takes to process requests. Each point in the heatmap represents the performance of a single request being processed. They note the unusual spike, where some requests are

Digital Object Identifier 10.1109/MCG.2020.3024039
Date of current version 12 January 2021.

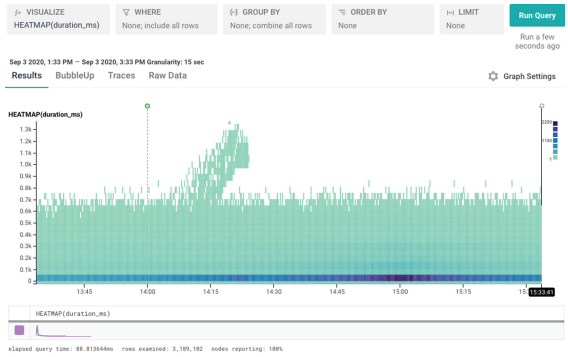


FIGURE 1. Heatmap of the latency for a request in Honeycomb. The darkness of a cell shows the number of requests that were served at that time and latency. The dark line across the bottom of the heatmap shows that most events were served very quickly, but an unusual spike across the top shows some that were much slower.

taking much longer than others, and want to know why they are different.

Using BubbleUp, they select those events (see Figure 2)—the selection is shown as an orange box. BubbleUp responds by showing them a series of histograms comparing the data within the selection to that outside of it (see Figure 3). Each histogram represents one dimension of the data. We compare the events that make up the selection to the events that make up the baseline—all the remaining events. The team can rapidly see that only one `endpoint` and one `app.user_id` were affected: there is only one selection bar on the histogram. In contrast, they can also see that `app.platform` and `app.build_id`, in the second row, do not seem to be important factors: the selection and baseline bars are very similar.

DEBUGGING DISTRIBUTED SYSTEMS

Let's step back to discuss how BubbleUp fits into a broader domain.

Most of the web now runs on distributed systems. An online service might consist of dozens of different

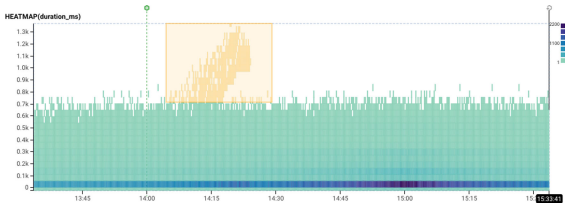


FIGURE 2. User selects a region of the chart.

FIGURE 3. Bubbleup's histograms, one per dimension, comparing the baseline to the selection.

microservices: front-end servers, back-end storage, authentication services, transaction processors, advertising management, and others. This complexity makes it difficult to figure out what has gone wrong when there is a failure. Which service caused a particular slowdown or error? DevOps teams try to instrument their code to describe what their systems are doing, and then try to diagnose and figure out what is going wrong when there is a failure.

The state of the art is to store important metrics—system-level metrics, like memory and CPU usage, and application-level metrics, like the duration of successful API requests—to provide a useful overview of how a service is doing. Each metric can be kept as a single time series. It can be useful to split these metrics out across multiple dimensions: for example, there might be a time series for every distinct API call, split further by whether the requests succeeded or failed.

This makes it extremely fast and effective to offer useful visualizations: a tally of erroneous requests, or the 95th percentile of request duration, for each API endpoint. A talented DevOps team grows experienced with the ways their system can fail and can recognize patterns in the metrics.

Visualization research has looked at this perspective on managing distributed systems. LiveRAC[1] and MeDiCi[2] visualize metrics for many systems simultaneously, for example.

High Cardinality and High Dimensionality

Unfortunately, this still yields a very shallow view of the underlying system and hides a lot of detail about what is actually going on. Accurate diagnosis requires richer information. For example, handling a user request may require a call to authentication, databases, and a webserver to be properly processed. To figure out what's

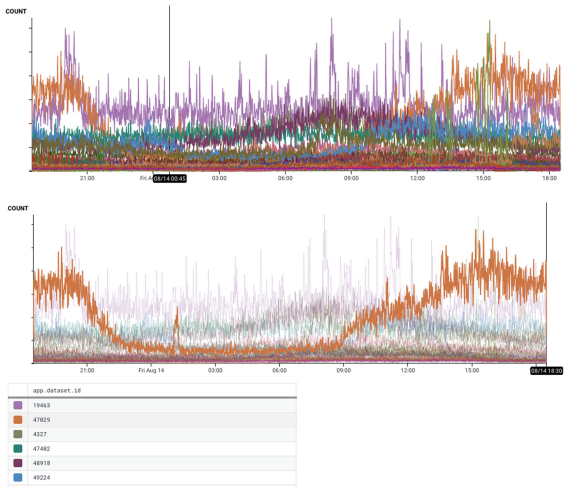


FIGURE 4. Count of events on a service, broken down by user ID. (Top) all of the customers, and (bottom) one customer, highlighted, showing a diurnal cycle. *Y* axes are obscured to hide actual traffic levels.

wrong with the system, it's helpful to know what user had made the request, which server processed it, what call was made to the database, and how long each of these took and what status code they returned.

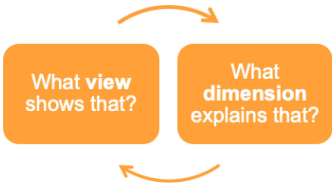
Each of those attributes—the call to the database, the user id—are different dimensions of the dataset; some of those dimensions, like the user id, are extremely high cardinality. Clearly, keeping a combinatoric collection of time series becomes prohibitive. A new generation of systems support those many dimensions, by using column stores. These can provide powerful tools to explore this data. Honeycomb is one of them; the general architecture for such systems follows the example of Facebook's SCUBA.[3] Now it is possible to provide that same count as before, but now split across many different dimensions.

To give an example of how it can look to start using high-cardinality data, Figure 4 shows 50 overlaid time series, representing different users of the system. The top chart shows all the different users overlaid; the bottom chart calls out the one with an unusual diurnal pattern.

High Cardinality and the Core Analysis Loop

This leads to a new analysis dilemma. How do people who develop and operate systems figure out which fields to look at? When there can be hundreds of fields, with millions of values, where do you look to figure out what caused a problem?

I was tasked with helping our users understand the complexity of their data. I went out and interviewed a dozen users, both internal and external to the company. The interviews focused on how they went about going from an alert to a response; in many of them, we chose a recent investigation that they had carried out and reconstructed their process of discovery—including looking at their dead ends. A single strategy recurred in debugging incidents, the *core analysis loop*.



Users would often start with an anomaly in the system that interested them. The core analysis loop started when the user visualized a basic metric that illustrates that anomaly—for example, they might have noticed that some requests were getting slow, so they visualized the median duration of events in the system to see whether it had increased. They would then iteratively try to group that metric by various variables. Their goal was to find a variable that had good explanatory power: that one particular value of one variable could show how the anomaly was different.

For example, let us say that we had encountered the graph at the left side of Figure 5. We wanted to better understand why the 95th percentile of data

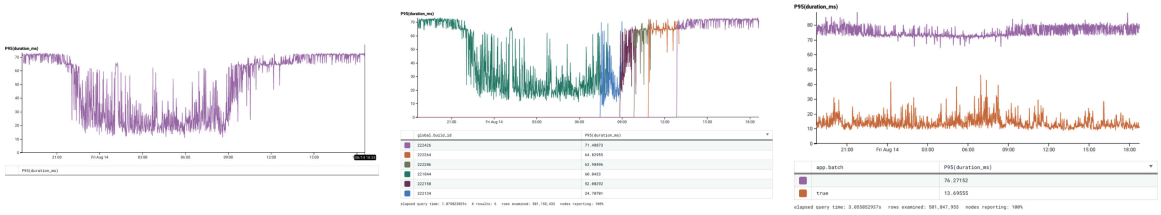


FIGURE 5. Left, the 95th percentile of the duration of requests to the service over time, which dropped from 22:00 to 10:00. Middle and right, grouped by `build_id` and by whether the `app.batch` flag was set. Build does not seem to be a factor, but batch does.

processing time for this query was high at some times and had such a strong dip from 22:00 to 10:00. I suspect that some dimension in my data might explain why the number had dropped. To find out, I might test a series of hypotheses: could this be caused by a specific build of the software? Might it be that a certain set of requests—perhaps those with the `batched` flag—are acting up? For each hypothesis, I would group the data: how does the line look when I aggregate only within builds? What about within the batch flag?

The process of choosing a good variable to check relies on the analyst's experience: if they saw error types that are often associated with database issues, they would turn first to fields that were related to databases. Others would use trial and error, or test hypotheses about their senses of different classes of bugs.

Because Honeycomb is designed as a high-performance query engine, with most queries returning in a few seconds, users could quickly try many different dimensions, looking for an answer. The process could be taxing, as users had to try multiple fields. Honeycomb offers a very loose schema—users can send in whatever sorts of events they want. Many dimensions had no meaningful values, or had too many distinct values, or simply were not relevant to the investigation.

This is a novel problem to the domain we were working in. Many of our competitors simply restricted the number of dimensions that users could send in, often limiting them to under 10. In contrast, it was not unusual for our customers to create datasets with hundreds or thousands of dimensions.

This was a competitive advantage—but also a pain point for our customers. In a low-dimensionality system, it was never hard to find the interesting dimensions. For our customers, there was a risk they would get lost in the noise.

Designing BubbleUp

Honeycomb decided to take on the problem of shortening the core analysis loop. If we could make it simpler to iterate through choices, they would more rapidly converge on their final result. As a secondary advantage, many of our users were unaccustomed to working with high dimensionality data; an experience that helps them understand how powerful their data were would also help them differentiate Honeycomb from our competitors.

We drew inspiration from Scorpion[4] and Macrobase,[5] which highlight the value of explaining anomalies by comparing them to other data.

The heart of the concept is comparing two high-dimensional datasets. Since we knew that our users had already identified anomalous data—such as the dip

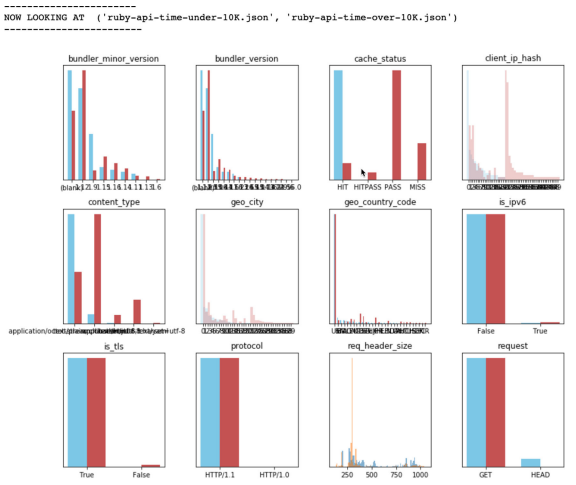


FIGURE 6. First prototype of BubbleUp, built in a Python notebook and visualized using `matplotlib`. Iterating in the notebook allowed us to rapidly explore the design space and validate our decisions.

from 22:00 to 10:00—the question was whether there was a way to separate these groups of points. It would be possible to use high-dimensional analysis techniques that would extract sophisticated, multidimensional explanations. However, we suspected that most explanations we were interested in could actually be much simpler: in many cases, a *single dimension* could distinguish between the anomalous and normal data.

We collected operational data from our servers and started experimenting with prototypes. The first prototypes ran in a Python notebook and generated sheets of side-by-side histograms: each dimension's distribution, shown for the data labeled as anomalous against the baseline.

In these exemplar datasets, it became quickly apparent that there were some dimensions that carried important signals and others that did not. (In Figure 6, which shows our first prototpe, mentioned above, for example, `cache_status` seems relevant, while `bundler_minor_version` probably is not). It also became quickly visible that many dimensions were boring: they had only one value or no values. Perhaps more interestingly, some dimensions had a meaningful value only in the outliers, or in the exceptions.

Admittedly, these were only samples of a single dataset. Every Honeycomb customer has distinctive data, with custom fields that relate to their own business cases. We needed to validate our beliefs about whether this was more broadly true for our customers, too.

One of our users granted us permission to run the code on their data; we sent them a PDF of the python

output. We were delighted at their positive feedback: they instantly understood what was interesting about some of the dimensions, and were able to diagnose a previously unintelligible problem.

This was validation enough to get started on building an implementation. Our design team worked to try to figure out how to incorporate the experience into Honeycomb—a challenge, as the UI did not have a simple way to select a region of the heatmap.

We deployed early (and unstable) betas, first to internal users, then to external users who opted into the experiment. Many of our external users participate in a customer-facing set of Slack channels; we were able to reach out to those users via Slack to build a group of interested users who could exchange feedback.

The feedback we got was a fascinating mix: users would send us long bug-lists and complaints about UI issues and pieces that were difficult to use—and then casually comment that they had used the tool to resolve an incident and that their time to detect issues had dropped from hours to seconds. (One of our insights was that if a user has spent enough time in a tool to complain about small details, then that implies they are finding enough value to dig that deep.)

While beta testing, one user wrote (in a Slack conversation), *[I] was seeing some big latency spikes … look at that it's mostly from one IP address …oh look, it's one IP in Australia.*

Another said, *it "automates" my previous workflow of breaking down and hovering over the table. Some of the use cases we had so far are pinpointing that a specific web process is being slow or seeing that the slowness is being caused by DB queries on a specific endpoint or job.*

The sales team also had a strong reaction to BubbleUp. They had been accustomed to showing how Honeycomb could handle a wide range of data by trying a series of wrong guesses before finding the right answer. BubbleUp allowed them to create a shortened demo (*there's an anomaly, we found it*)—or to walk through the slower process, and then show how BubbleUp short-cut it.

Arguably, the hardest part about releasing BubbleUp was the name: it went from *Smart Drilldown* to *Anomaly Detector* to *Copilot* before we settled on *BubbleUp*; different names were meant to both explain what it did, but also have a personality.

Decisions in Design Iteration

BubbleUp went through numerous rounds of design iteration. It is particularly interesting that we made a number of substantial changes after we released BubbleUp, in response to internal and external feedback on the tool, and our own experience with it.

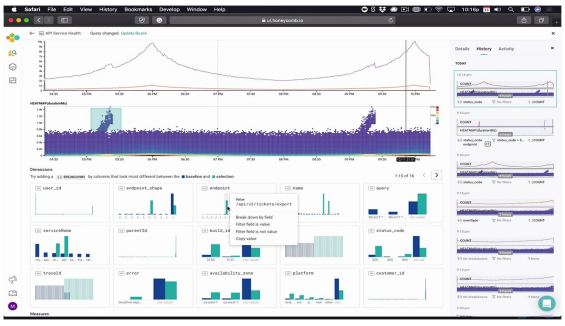


FIGURE 7. BubbleUp in the blue-green color palette. It was difficult to determine which were the selection compared to the baseline, as both colors were used in the heatmap.

Color Coordination

In the first iteration of BubbleUp, we had users compare blue to green histograms (see Figure 7). We got very strong feedback that this was confusing: both colors were well within the color palette of the heatmap, and so users needed to look hard to figure out which was the selection and which was the baseline. By changing to the yellow-and-blue color palette, the questions went away instantly: users understood the yellow mapped to the yellow highlighted area.

Histogram Ranking

We wanted to ensure that the histograms were ordered usefully, to help ensure that users could identify important dimensions. We played with several different metrics and even ran A/B tests comparing ranking algorithms. In the end, we picked a *relative risk* metric (adapted from Macrobase[5]), asymmetrically weighted to highlight fields that had low cardinality values in the selection.

Null Pies

Honeycomb data can be nonrectangular: not all rows of the dataset have all the same fields. For example, a dataset might contain some events that use the Internet—and, therefore, have fields like `http.status` and `http.request`—while other events might use a database, and so have fields like `b.request` and `db.response`. We rapidly found that for many of the most interesting datasets, a single bar for "no meaningful value" turned out to dominate much of the UI. In Figure 8, for example, `status`, `batch`, and `batch_num_datasets` were often empty but dominated the display.

We worked with a designer to create donut charts that could represent *how many nonempty* values there were, instead. In these three images, we see that every event has a defined `trace.trace_id`. Interestingly, most events in the ■ selection have a defined `trace.parent` –

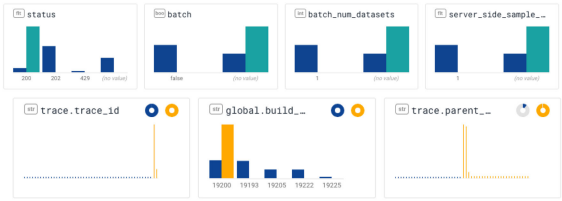


FIGURE 8. Bubbleup with *no value* columns, top, and null pies, bottom. The pie chart shows the percentage of rows that have a valid, nonnull value; the histogram only shows the valid values.

but very few events in the ■ baseline have a defined `trace.parent`. This can help a user rapidly understand how the two groups differ.

Removing Background Events

When we first designed BubbleUp, we contrasted the ■ selection against *everything*. That meant that we counted points inside the rectangle twice: once in the ■ baseline then a second time in the ■ selection. While this had a certain mathematical elegance, it made it actively harder to recognize signals in the data, because data in the selection would also appear in the baseline. Removing points in the selection from the baseline made it far easier to see what was different.

Interacting With BubbleUp

We rapidly realized that one of the most interesting next steps from a BubbleUp was issuing a second query: allowing users to ask *when I eliminate this factor, what's different?*, or *when I focus on this factor, how does it look?*. Following user feedback, we added click through interactions that allow users to create filters and groupings from BubbleUp bars.

BubbleUp for Lines

BubbleUp was efficient to build because it can compare two well-known sets of data points. Still, the most common request to Honeycomb is not comparing heatmap regions—it is understanding why a count, or 95th percentile request, failed. Unfortunately, it is much harder to compute that difference. In those aggregated graphs, we longer know for sure which points sit in the baseline and the selection. Crude techniques—like picking only the slowest points, or all the points in the time region—proved to be insufficiently accurate to provide useful signals. The techniques in Scorpion[4] can help with that computation, and were considering how to incorporate them. Still, BubbleUp-for-Count has been one of the dominant feature requests from our users.

Continuing Life of BubbleUp

BubbleUp continues to be an integral part of the Honeycomb experience. Interestingly, it has had a secondary effect: it is so different from features offered by competitors that it has caused people to see Honeycomb as more substantially differentiated. It emphasizes the value of high dimensional data, and how value can be derived from something that had been seen as out of reach.

The core value of BubbleUp is that it makes it easy to ask novel questions, e.g., *What is special about that particular point?*. This is a key question in the DevOps world—most likely, in many other fields, too—and it drives action. Knowing why a data point is special can help figure out what parameter needs to be tuned, what server needs to be rebooted, or what line of code broke.

We have also begun to build BubbleUp into our product's workflows because it answers the fundamental need to know what happened. For example, Honeycomb recently released features to support service level objectives (SLOs). An SLO computes the ratio of *good* and *bad* events. A key panel of the SLO view shows a BubbleUp of good compared to bad events; we have found that this comparison can rapidly highlight important changes that have caused the SLO to degrade.

CONCLUSION

Comparative Histograms

The heart of BubbleUp is comparing two pools of data to each other as paired histograms. While the applications we have discussed here are for specialized domain, the broader questions about comparing two sets of data should be broadly applicable. I would encourage data analysts in other domains, too, to pick up this sort of a cross-dimensional comparison tool, and to consider paired histograms as a powerful starting step.

Iterative, User-Centered Design

Much of the strength of BubbleUp came, in part, from iterating on feedback from our users throughout the process. Honeycomb users—internal and external—were involved in every development stage, from the first prototypes in Python, through the first release, and then through the incremental improvements. Because of their feedback, we were able to verify that we were going in the right direction, could decide when we were good enough to release, and could prioritize improvements.

This gave us the confidence to massively simplify the fundamental concepts. Comparing sets of histograms on points is computationally simple, rapid to implement, and easy to understand. While techniques like Scorpion[4] and Macrobase[5] are powerful, the cost of building that infrastructure and the complexity of maintaining it was intimidating to a product team in an early stage startup. Building out BubbleUp allowed us to accomplish that the level of value at a fraction of the price.

Fundamentally, BubbleUp has helped our users discover the value of high-dimensional data and to deeply understand their challenges. When that alarms wakes them at 3 A.M., they can find precisely where to look—and faster remediation means better responses to crises.

ACKNOWLEDGMENTS

All of Honeycomb had a hand in BubbleUp, but particular thanks go to E. Freeman, C. Sun, C. Toshok, and I. Wilkes, who implemented the front and back ends for BubbleUp, with visual design by C. H. Chua. BubbleUp was named by D. Mahon. The inspiration of the tool came out of invaluable conversations with E. Wu at Dagstuhl Workshop 17461, *Connecting Visualization and Data Management*. The author is particularly grateful for the feedback from the Honeycomb Pollinators Slack Community.

REFERENCES

1. P. McLachlan, T. Munzner, E. Koutsofios, and S. North, "Liverac: Interactive visual exploration of system management time-series data," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2008, pp. 1483–1492.
2. D. M. Best, S. Bohn, D. Love, A. Wynne, and W. A. Pike, "Real-time visualization of network behaviors for situational awareness," in *Proc. 7th Int. Symp. Vis. Cyber Secur.*, 2010, pp. 79–90.
3. L. Abraham *et al.*, "Scuba: Diving into data at facebook," *Proc. VLDB Endowment*, vol. 6, no. 11, pp. 1057–1067, Aug. 2013.
4. E. Wu and S. Madden, "Scorpion: Explaining away outliers in aggregate queries," in *Proc. VLDB Endowment*, vol. 6, no. 8, pp. 553–564, Jun. 2013.
5. P. Bailis, E. Gan, S. Madden, D. Narayanan, K. Rong, and S. Suri, "Macrobase: Prioritizing attention in fast data," in *Proc. ACM Int. Conf. Manage. Data*, 2017, pp. 541–556.

DANYEL A. FISHER is currently a Principal Design Researcher with Honeycomb.io, San Francisco, CA, USA. He was with Microsoft Research before joining Honeycomb. His work focuses on bringing powerful data visualization tools to end-users. He received the Ph.D. degree from the University of California, Irvine, Irvine, CA, USA. Contact him at danyel@honeycomb.io.

Contact department editor Daniel F. Keefe at dfk@umn.edu or department editor Melanie Tory at mtory@tableau.com.