EPISODE 1155

[INTRODUCTION]

[00:00:00] JM: Effective data science requires clean data. As data moves through the data pipeline, there may be errors introduced. Errors can also arise from code changes, database migrations and other forms of data movement. How can you ensure data quality within a fast-moving dynamic data system? Datafold is a company built around data quality management. It allows users to compare tables and databases as well as automate data QA.

Gleb Mezhanskiy is a founder of Datafold and joins the show to talk about the data quality space and what he's building with Datafold.

[INTERVIEW]

[00:00:38] JM: Gleb, welcome to the show.

[00:00:40] GM: Thank you, Jeff. Glad to be here.

[00:00:42] JM: As we both know, the quality of a machine learning model is highly dependent on the quality of the data that goes into building that model. What are the problems that can arise with data that go into training a model?

[00:00:56] GM: Yeah, absolutely. The types of problems that can occur are really diverse. So anything to first of all data just not being there or certain parts of the data, for example, you're expecting to have certain dimensions, for example, data coming from different device types coming in and you suddenly stop receiving some of the dimensions quietly. So that's just one type of the problem. And some other types of the data becoming wrong could be malformed values, for example, you expect certain fields to be a flow type, but suddenly you start getting that as a string type. And that can either break your learning process, break your dashboards or just fail silently, which is sometimes even more scary. And all in all, there are probably two types of failures that we see there. Some of them are loud failures when your pipelines just break.

I think the more scary and potentially more disruptive eventually types of failures is when data just changes the shape, changes the profile. Maybe some data becomes missing. Maybe the calculation has been changed upstream. And so the training data that comes into your model becomes skewed and not what you expect, which leads to feature drift and ultimately the model drift and suddenly you have a poorly performing model without knowing it.

[00:02:23] JM: So the data pipeline, there is a lot of different technologies that are involved in a data pipeline. Could you describe some of those technologies?

[00:02:30] GM: Yeah, absolutely. I think roughly the data pipeline, the modern pipeline can be broken into a few pieces. It all starts with how we collect data. And there, there are probably two main categories of data collection. So we have what we call event instrumentation. This is when you record certain events happening in your ecosystem. For example, a user visiting a page or clicking on a particular button, or a particular transaction happening within your billing system. So those events typically originate in various parts of your backend and frontend and then they are collection in the form of a stream through different event processing tools. And then ultimately they flow typically in your data lake.

And those events typically have a form of a JSON format, or sometimes they're serialized in like a Protobuf. And those events basically capture particular things occurring within your ecosystem, and typically you would have full control over how these events are structured, what kind of data you record in them. And on the second class of input data originates in third-party systems. For example, you may have a support system that has four tickets and then you have all sorts of metadata that comes from there or a CRM system that records your customers and activities with them. And so it's typically, we see that companies are also piping these kinds of datasets into their data lakes for enrichment of their internal data and for further analysis.

And the third type of the input data is probably your data that typically stored in your OLTP databases that basically power your application. So let's say every app or website would have a backend database such as Postgres, MySQL or NoSQL database and it's also frequent that companies would export the data that sits there into the data lake as well to get basically source of truth data. That's basically the data inputs.

And then it's customary that all that data is flowing into the data lake. This is probably the most frequent architecture. So all the raw data flows into a large database, can accommodate all of these data points. And then after that, it's typically the transformation part that's taking place. So teams would use different frameworks, for example, SQL, or Spark, or other frameworks like Apache Beam to transform the data that's now in a data lake basically aggregating, merging it and reaching it, filtering, to ultimately produce the tables, the datasets that can be used downstream by analysts or for machine learning purposes.

And then finally once those datasets are produced and they sometimes are called analytical model, sometimes they are called star schema, after that, those data sites are consumed either by machine learning models for training or they can be piped into the BI tools for dashboarding or can be used in ad hoc manner by analysts for one-off reporting. So that's basically I would say the gist of the classical data pipeline.

[00:05:45] JM: So let's say a data pipeline is being processed and some error occurs, some kind of data corruption might occur. What are the ways in which that data corruption can happen?

[00:05:56] GM: Yeah, absolutely. There are multiple ways, and one of the frequent cases for data corruption happened because of the raw data, for example, events changing. As you can imagine, companies these days typically are often architecting their applications in the form of microservices with different teams owning a particular service. And the events that are describing the business are also distributed into those microservices. For example, if I have a service that's responsible for billing transactions, I might record events about those transactions. Or either have a microservice that's responsible for user accounts, I will be recording things that are happening to the users.

And so what ends up happening because of the distributed nature of ownership is that engineers may change structure of events when they are iterating on their application code, and that can lead to breakages or other problems downstream because the other teams that own a data, once its sent to the data lake, may not be aware of those changes. And then another type of error can happen because some of the processing logic that is used on top of the data lake to transform data is also being changed. And it's also often that when changing the business logic that transforms data, the data developers may introduce certain regressions. And in this case, you may basically change some definition, for example, how you compute conversion rate in your SQL code. And you may end up with a dataset downstream, which is much different from what you would expect.

I actually have a person story from my time as a data engineering at Lyft, where I was on call and I was woken up at 4AM because one of the SQL jobs that computed some important table for Lyft was broken, basically the job failed because some malformed data entered the pipeline. So that's, to our first point, about breaking events.

And so what I did is I had to introduce a particular filter in a SQL logic to filter out that data. But in the process of doing so, I introduced accidentally some error, which ended up filtering out much more data than it should. Basically, it filtered out good data as well. And what happened was that next day we discovered that the majority of companies' datasets that were feeding from the dataset that I changed were all corrupted. And so that basically canceled all the meetings that were structured around dashboard that also halted all the machine learning training processes and ultimately caused the company a lot of damage.

And the problem was that I actually followed the process for making change to the SQL code. I even got a code review, but the issue was that we just didn't have good enough tools to catch such errors, to catch such regressions. Basically, understanding exactly how it changed in the source code that transforms data would impact the change in the data that gets produced by the job. So that is one of the focuses of Datafold, is how to give an ability to data developers to automate the regression testing of their data.

[00:09:23] JM: So you're talking about regression testing data. Can you go a little bit deeper on what that means?

[00:09:29] GM: Yeah, absolutely. The challenging part is that in today's analytical world, the code that transforms the raw data into business reports, so prepares the datasets to be used in machine learning, that code contains a lot of business logic, a lot of rules, and it's often that particular jobs would contain hundreds of lines in SQL. And so there's a lot of complexity in those transformations, and it's a lot of complexity for humans to be able to manage. And at the same time, the tools for transforming data and the frameworks, they are not providing a lot of visibility into what is actually happening with the data that gets produced. So basically, there are some raw datasets, then you run a multi-hundred line SQL scripts on them and then you end up with a resulting dataset. And you don't really get a lot of visibility into what gets produced. You actually have to spend a lot of time manually to slice and dice the resulting dataset to verify that the logic that you wrote in SQL actually is what you intend to do. Or if you're making change to someone else's SQL code, it's also hard to verify that nothing bad had happened to the data.

And the types of regressions that can occur when people are changing this transformation code are really – Basically, there are many things that can go wrong. So one, you can accidentally filter out data that it shouldn't, or you can introduce some side effects, such as find out when you may end up with duplicate records, or you may just use a wrong formula, so you end up with the wrong number on the other side. And that's also not to mention difference ways that databases can handle types. So sometimes you may accidentally have a wrong rounding logic in your calculation for things like, for example, revenue or profitability, and you may accidentally round up or round down values that on the scale of a large company can give you completely skewed financial results.

And so the regression testing is whenever you introduce changes to the source code, how do you make sure that you didn't break anything? And in the software engineering world, it's customary that engineers would write unit tests. They would also write integration tests, and finally there would also probably be a QA process that would check how the application performs from the end-user perspective. And that process is typically largely automated. So every commit that goes into application code is verified with a CI process running all these tests and then letting the engineer know what tests have failed. And if some tests have failed, the code is not shipped to production.

So in a data world, unfortunately, such process is fairly rare. So typically, the teams are relying on the data developers to verify every change manually. Basically going and writing some code just to verify the resulting dataset that everything is fine, and that can be fairly laborious. So it is not uncommon for teams to spend multiple hours to multiple days to verify a change in a SQL logic, because the complexity is high and mistakes are also high for getting the wrong number.

And so what we've done here is we created a tool which we called DataDiff, which essentially allows you to compare datasets before and after they change. For example, a dataset, which is produced by a master branch or physical code versus a dataset that's produced by a modified source code, and essentially it's like looking at the redline in your Word document. So you see exactly what changed both in terms of the individual values, but also in terms of data distributions. So did you end up with more rows, or lest rows? Do you still have the same keys? What changed in the columns? Did the data distribution change? And that gives the data developer's a lot of confidence whenever they're making changes to the source code that they indeed end up with the right data after the source code change is shipped.

[00:14:00] JM: Could you tell me about the problems that using data QA like Datafold, what kinds of problems does it help prevent?

[00:14:09] GM: So the main problem that specifically currently we're talking about, the regression testing, which is one of the tools that we provide, the problem that it tries to prevent is people breaking code accidentally. So how do you make sure that whenever you are changing the business logic, you end up with the right results after you apply the change? And it's, like I said, very similar to the QA process and the software engineering. Whenever you, let's say, ship a new feature, you want to make sure that you did not impact any existing functionality in any bad way. But in terms of what it means for data teams beyond that, it's actually, one, enables them to move faster, because they don't have to spend multiple hours to verify every single change. Two, it reduces the probability of data incidents. Whenever someone breaks dataset and then someone may find out that something got corrupted long time after, and all this time the business could have relied on that data. And three, it also enables companies to involve a greater number of people to contribute to the analytical codebase, and that's important, because data engineers and data analysts are very scarce resource and companies are always putting them on the highest input projects.

But the common problem that we see in large companies is that as other roles, such as product managers, and software engineers, and operational analysts become more data sophisticated, they also want to contribute to the analytical codebases. Basically, writing SQL, doing reports and potentially also scheduling their scripts to run as part of data pipelines. But data teams justifiably are often anxious about other team members contributing to the codebase because they feel that it's really hard to verify that what other teams are contributing is of high-quality, and I don't want to be responsible for someone's broken code or someone else breaking analytical code.

And so by automating the regression testing, data teams can enable larger number of people to contribute freely to the analytical code, which in turn speeds up the entire organization. So now everyone can take advantage of ETL tools and create datasets, and that also release pressure from the data team. So basically, just beyond the workflow of data engineers and data analysts, we also get positive affects for the organization as a whole.

[00:16:52] JM: Who would actually be using Datafold? Would it be data engineers, or data scientists, or software engineers? Who's implementing these tests?

[00:16:59] GM: Yeah. So we see that the most frequent users of Datafold would be people who in the roles who are working with data daily, basically, data producers or data developers. And typically in the organization, there are a few roles that are doing that. So first of all, these are data engineers, because it's their job to build data pipelines and write the business logic that creates datasets, but also data analysts and data scientists depending on what kind of projects they're working on can also be contributing a lot to the ETL code and building datasets. And in this case, they also can take advantage of our features to speed up their quality assurance.

I would also want to kind of expand the scope of our conversation a little bit, because Datafold is not necessarily or not only a tool for regression testing. We see ourselves as a data monitoring platform. And so one way in which we enable teams to monitor the data and the quality of data is DataDiff, which allows you to track the changes in your datasets when you modify the source code. But we also have other features that enable teams with greater visibility into their data ecosystem.

For example, we provide a tool called data profiler, which enables any user of analytical data to instantly discover datasets, discover columns within those datasets and be able to explore the data distributions within them. And as basic as it may sound, it's actually a fairly big problem in more organizations, because companies, once the companies reach certain maturity, they typically have fairly complex data ecosystems, meaning that they have probably a few thousand tables each with few dozen columns, which means that overall you have hundred thousand columns and more in your data lake, and that obviously is a very large amount of information to deal with.

And so a lot of the time that whenever people want to work with new dataset or they’re trying to find the right data to use in an analysis or for future generation, a lot of the time would be spent on first finding the potentially relevant datasets and then exploring them. Understanding how is the data distributed in every particular column. What are the values that exists in that column, for example, what product types are in this column or what CDs are there? And whether the column actually reliably provides data or maybe it's sparse, meaning that a big percentage of the values are empty.

And so we help streamline this process by profiling data in advance and providing the data users with a convenient view where they can see distributions of data in datasets in a matter of a click. And the theme of data observability is we see it as an emergent category of tools. So just like today in a typical data stack, companies would have systems to ingest data, to store data basically, data lakes. They would also have systems to integrate datasets from third party tools. They would have transformation of frameworks to kind of run transformation logic in SQL, or Python, or PySpark.

We also see the need for an end-to-end data observability platform that would essentially tap into every piece of data infrastructure and provide teams with visibility of what data they have, how these data are changing? What are the distributions of values in those datasets? Is the data up-to-date, or is it stale? Where does data come from? And so we see as our mission to enable visibility into, one, every piece of data infrastructure, but also into every dataset that the company has.

[00:21:07] JM: Let’s talk a little bit more about how Datafold actually works. So if I’ve got data in two different tables, how does Datafold compare those two tables?

[00:21:16] GM: So for comparing the tables, Datafold actually works in two different modes. So if you are comparing tables that live in the same physical database, in this case, Datafold would leverage that database to compare those dataset. Basically, it would issue queries to your database to compare those datasets in detail both in terms of the primary keys. So identifying whether both tables have the same unique IDs for tables. Maybe if you're working with a table that describes users, you may want to know whether both tables have the same sets of user IDs. We also are comparing values across every column and we can tell you what columns have differences, and we also give you visibility both in terms of the individual values and in terms of distributions of value in every call. For example, maybe after you made a change to how you compute your margin, the distribution of that column has changed. And so we will show you that. But there's also a mode in which you may want to compare datasets between physically different databases. For example, as we talked earlier, you may be copying data from your transactional database, such as Postgres, to your Datalake for further analysis. And because typically we're talking about hundreds of tables and for large data volumes, frequently, data gets corrupted in the process of transfer.

And so would Datafold can help here is to identify where in the process of transfer, you end up with a same data in your data lake once the copy was done. So in this case, Datafold would go into both data bases and pull data from them. We also have a sampling mode so we don't have to necessarily pull the entire dataset, but we can intelligently sample the dating both ends, and then we would tell you whether statistically we believe that both databases have the same dataset or not.

[00:23:22] JM: What's been the hardest engineering problem you've encountered in building Datafold?

[00:23:27] GM: So one of the harder engineering problems that we're working on is how to identify dependencies within the data ecosystem of a customer. So for example, in order to enable the visibility into your data ecosystem, we have to understand how the data flows throughout your system. And typically, once the data gets in the data lake, and that can take multiple stages, data teams start transforming it into tables and then aggregating those tables into further aggregated tables. And so we're talking about layers and layers of transformations.

And in order for us to be able to tell our customers whether the data is of high quality and if there are any issues to be able to tell where do we believe the bad data originates. So is it the raw event that got corrupted? Or maybe this is some job in the middle that now emits wrong data. We have to understand how data flows not just on the table level, but also on the column level. So if you look at a particular column, maybe the channel through which the user has come to your website, you want to understand where does this value come from, and that's a hard problem, because typically, like I said, the codebases that contain those data transformations are fairly complex.

And so being able to construct this dependency graph is a fairly hard engineering problem, because you have to collect all sorts of different metadata. You have to tap into the SQL code, the source code to understand exactly how every value is built. So this is probably the hardest engineering problem that we've dealt so far.

One thing that I wanted to highlight is why did we start working on Datafold and focusing on the data monitoring and data quality, is that over the last 10 years, we've seen a lot of things changing in how companies are working with data and what kind of problems they encounter. So even five years ago, one of the major problems for data teams was how to integrate all different datasets into the data lake. So how to actually implement connectors to CRM systems and to their transactional systems and reliably move data to the data lake, or how to ensure that the data lake actually can store all the datasets and how can they make sure that whenever analysts are working with SQL or with BI tools, the data lake is actually capable of processing all of these requests.

And so I think over the last five years, many of those problems actually got solved. So now we have fast databases. We have great BI tools that help teams visualize, create content. We have great ETL frameworks. And actually almost every month we now see a new ETL framework popping up that has particular strengths in terms of how to process the data. So all of that led to the ease of creation of new data. Now it's very easy to collect data. It's easy to store it and it's easy to create new datasets off it, which in turn led to the explosion of data within the companies, explosion of datasets in their variety and their complexity, and that created a new problem of how do you organize and how do you keep track of all the data assets. How you understand what data assets can you rely on and what to use in your job?

And also the speed of iteration with which companies are changing their data pipelines also massively increased, and that led to the problem of how can we move fast with development of our data pipelines and aggregations without breaking things, because more and more of the business now relies on high-quality data. And so that's why we saw an opportunity and the need for the data monitoring platform that would help teams both understand what data they have, how it flows, where it comes from, as well as monitor its quality and have a way to automatically track changes. And just to provide a limited background on why we're working on what we're working on.

[00:27:53] JM: How do you see the process of data engineering changing in the next few years?

[00:27:58] GM: Yeah, that's a great question. I think that if we think about data engineering as a process or as a craft, nothing has really changed fundamentally in terms of how it works in, I would say last 30 years. So people basically take data, analyze it and then take the business requirements and then turn them into some sort of code, SQL code, Spark code or whatever framework they use to transform their data. And then they start maintaining it, changing the logic and making sure that the quality of data is high.

I think one of the biggest opportunities for streamlining data engineering process is, one, to automate the quality assurance bar, which is what we're working on right now, because right now it takes a lot of time for data engineer to verify, validate every single change. And I think the seconds opportunity that will probably come in the next three years is automating the process of data modeling itself, because right now the needs for the business are evolving much faster than the analytics. So businesses want to see their launch features fast and they want to see very soon after they launch the particular feature, what's happening with it. And that puts a lot of pressure on data engineers to create the corresponding data models. For example, if a company would launch a new product that has new ways of people interacting with it, that probably demands the data engineers to create a new table that would describe the user flow or maybe a number of tables that would describe the user flow so that the analyst can analyze that user flow.

And I think that a lot of the time that data engineers are spending right now is spent in understanding what are the fundamental building blocks, like what are the fundamental events that should go into this table? Maybe a user clicks a button. Maybe a user goes to the next page. Who is the user? Where did they come from? And filling a lot of these contexts takes a lot of manual work to explore every single source dataset and then glue them together in a single table. And I believe that in the next few years we'll see systems that would create those data models more or less automatically without the human needing to do all the manual work of finding the data, finding that corresponding contexts and joining it all together in the code and then QA-ing.

For example, frequently for marketing activities, companies would want to collect as much context as possible about their users. So where did they come from? What do we know about them? What demographics we have on the users? What is their customer segment? And other associated traits. What did they do on the website? And so pulling all of these contexts together is almost impossible in the modern organization, just because how fragmented the data is. And because, for example, the CRM tool may have a particular way of structuring your user data, and it has a particular user ID that associates to your users. Your support system may have a different user ID and different set of columns on the users.

Finally, your own system may have third way to describe users. And so gluing all these, gluing the data from all of these disparate systems together requires a lot of effort and a lot of manual work to reconcile and connecting all the contexts together. And a lot of the times, it's just beyond human capacity to be able to bring dozens of different columns that describe a user into one table.

And so companies end up with multiple teams building their fragmented ways of looking at the users just because they would just pull the things that they care about, and a lot of times they would duplicate the work. And I think that if we really have a system that understands how data works, where it comes from, and is able to semantically understand the meaning of a data, then the data models basically have a large table about users can be crated semi-automatically without a lot of human intervention. And so I think that overall, 40% to 60% of what data engineers doing right now can be automated.

And here we're mostly talking about the process of modeling the data itself. Basically, how do we organize it? We're not talking about the ways, or the frameworks, or the technologies used to sort of process the data. And I think that the separate part of how the data engineering evolves is also to do with what kind of systems are in paradigms I used for that? For example, the traditional model of data engineering is to process data in batches. For example, every nights you would run a SQL query that would compute all information about the users and produce a dimension table that describes users. And then every night you would essentially recalculate the entire table.

What it means is that, one, you're processing a lot of data redundantly, because oftentimes you would re-compute everything every night. And two, you typically have a large lag for data availability. For example, you wouldn't be able to use raw data or data that came within the last 24 hours until that process has run. And more and more we see companies introducing streaming processing into their data transformation flows, where instead of being processed in batch, data would be processed continuously. So I think that's probably the second trend that we'll see. And I think the most powerful trend there is our systems that are merging the batch processing and the streaming processing, for example, Apache Beam, where you are able to define the business logic once and then you can either process that in batch or in streaming, it doesn't really matter.

[00:34:13] JM: Could you see more about Apache Beam and why you see that is important? We've done shows on Apache Beam. We've done shows on all the streaming frameworks. I was never able to really grasp what made Beam special relative to Flink or Spark Streaming.

[00:34:28] GM: Right. I think probably the most important feature of Beam that distinguishes it from other system that you mentioned, such as Flink and Spark Streaming, is that it's a higher level framework. So the main advantage of Beam is that it decouples the source code or the business logic. Basically, how do you express data transformations from the underlying engine that actually processes the data? So that means that you can write code once and then you can choose the right engine to execute it on. For example, you can write it on Spark, you can write it on Flink, and that gives companies a lot of flexibility in how they choose their infrastructure and how they optimize the infrastructure. And this is a very powerful concept, because one of the biggest bottlenecks in analytics is poor performing infrastructure. And the reason why companies stay with poor performing infrastructure and not able to move to new systems, for example, they would use a legacy database and not able to migrate to something of higher performance, is because the migrations are really, really hard. Because you essentially have hundreds of thousands of lines of code that transforms the data that is bound to a particular database framework or a particular framework. And the advantage of Beam is that it allows you to move between technologies seamlessly and plug-in a particular underlying data processing engine according to its strengths. Maybe you discover that a particular using Flink insetad of Spark can save you 20% off your infrastructure costs or maybe your streaming jobs. But Spark is more efficient when you use it for batch jobs.

And so you can optimize infrastructure in that way and avoid the vendor lock-in. And I think the second important paradigm that Beam brings is something that I mentioned earlier, which is blurring the boundary between batch and streaming. So before Beam, if a data team wanted to take a particular job that runs overnight and computes a particular table and turn it in a streaming job, they would likely have to rewrite everything from scratch. Basically, we write all the code just for it to comply with how streaming engines work. But in Beam, again, you can write your code once and you can start executing it in batch mode, because it’s simpler or cheaper. And eventually you can turn it into a streaming job to get data almost real-time without needing to rewrite your business logic, which is very complex and expensive.

I think one of the challenges of Beam though is that it's a fairly new framework that offers teams a completely new way of defining the jobs. And so it has a fairly steep learning curve. And so it hasn't showed signs of becoming mainstream just yet. But I think it has a potential given that it's pushed by Google Cloud platform, which obviously has a large momentum behind it.

[00:37:46] JM: Yes, and it seems like they've been pushing for a pretty long time. It hasn't gotten as much traction as you would expect. I admit, I've always thought like they must know what they're doing, they’re Google. But why do you think it hasn’t gotten as much traction as other streaming systems?

[00:38:01] GM: Yeah, I think that that has something to do with how technology evolves and gets adapted. The mainstream for data transformations is probably still SQL, probably followed by Spark with its [inaudible 00:38:15], PySpark and ScalaSpark. And I think that there’s a still huge number of teams and companies that are just learning SQL, just learning batch, and Beam is just two steps ahead of this in terms of its complexity and in terms of the maturity of tools that you would use in order to develop in it.

And so what if seen as the mainstream of analytics, basically the majority of data engineers and analysts would probably stick with SQL as their framework and running it on one of the more widely-spread databases that are offering SQL interfaces. And then for jobs that require real-time, typically, engineers would pick a mature framework such as Flink or Spark-structured streaming in order to build those jobs just because they're going with something, which is mature, that has mature ecosystem and a large community behind it. I think Beam is just a little bit early both in terms of the tools and its maturity as a platform. But I think it has potential, and especially once they will offer more support for Python, which is beloved and probably almost mainstream language for the beta science community these days, they will probably see more traction, and obviously integrations with other tools and technologies in the ecosystem. Because if you, let's say, you have your data in a particular flavor or database or a messaging queue like Kafka, but you're working with a framework that has limited support for that technology, that can easily be a prohibitive challenge for you in order to be able to adapt technology. That's why more mature databases and engines that have a greater interoperability probably have win over Beam right now.

[00:40:16] JM: Well, just to wrap up, what do you see is the near future for Datafold? What are you working on?

[00:40:21] GM: Yeah. So we are working on – So far we've been offering standalone features such as DataDiff, data catalog with profiler and a system that tracks your metrics and then detect anomalies in them. And the major next release for us would be tying all of these features together into comprehensive end-to-end monitoring platform, where we not only give you a tool that you can use for verifying your data whenever making changes, but we can actually track your data as it moves throughout your data pipeline and automatically proactively alert you on any anomalies it would discover there. And not only tell you that a particular column in a particular dataset is wrong, but also pinpoints into potential sources of those issues and show you how anomalies propagate throughout your later pipeline. For example, if we detect that a business report that is fed into BI tool has stopped populating a particular column, we will show you where in your system maybe in a raw event table or maybe in some transformation job that likely occurred. So tying all of these together is the next major milestone for us.

[00:41:40] JM: Okay. Well, Gleb, it's been a real pleasure talking to you. Thanks for coming on the show.

[00:41:43] GM: Thanks so much, Jeff.

[END]