### Introduction

Many companies start their data quality journey by implementing SQL-level checks, for instance, [dbt tests](https://docs.getdbt.com/docs/build/tests), or [Great Expectations](https://docs.getdbt.com/docs/build/tests). These checks, usually run as part of ELT, help ensure that data pipelines work properly. They’re a good first step in addressing data quality issues.

However, they have their limitations. Without the right context, writing and automating more and more checks in a growing organization only leads to alert sprawl - no one knows where an alert came from or which check it corresponds to.

There are also limitations inherent to testing: you only get information from the things you decide to test. This is why ideally, at some point, organizations shouldn’t just be testing their data pipelines: they should be moving on to more continuous, contextualized monitoring with data observability.

In this blog post, we will cover what you do and don’t get from SQL checks like those in DBT tests and Great Expectations, and when you should make the shift to a data observability solution.

### What are DBT tests?

DBT tests are SELECT statements that try to grab “failing records” that violate one of your assertions. So for example, if you assert that a column is unique in a model, the test query selects for duplicates; if you assert that a column is never null, the test seeks after nulls. If the test returns zero failing rows, it passes, and your assertion has been validated.

DBT tests are usually colocated with the DBT model and run either before or after a model is run. This helps validate that a source table is clean, or that a transformation did what it was supposed to.

DBT comes with 4 generic tests already built-in – uniqueness, not null, accepted values, and relationships – and users can additionally define custom ones that incorporate multiple tables/views.

### What is Great Expectations?

Great expectations is an open-source documentation and validation tool that allows you to express what you expect of your data. You can create an expectation like, “the values in this column must always between 1 and 6”, and run that against your dataset.

Compared to DBT tests, Great Expectations provides more built-in tests. Since it supports tests written in Python, it enables [more complex testing](https://docs.greatexpectations.io/en/latest/reference/glossary_of_expectations.html) than in pure SQL.

Through an extension package, DBT-expectations, dbt users can deploy tests in the style of Great expectations directly to their data warehouse.

### Drawbacks of DBT tests and Great Expectations

#### Manual configuration

Each DBT test or Great Expectation has to be manually implemented, which can be fairly onerous. For example, if an organization has 100 important tables in their data warehouse and wants to ensure basic test coverage for all of them, with three basic tests for each column in every table and ten columns per table, they could be looking at over 100 hours of manual work just to get the basics in place. And this doesn't even take into account dynamic or seasonal data that require ongoing threshold maintenance.

#### Little to no context for failing checks

Say you have a DBT test or GE expectation that fails. Currently, if you are running on DBT cloud, you can get an alert on Slack that a test has failed (if you are using your own DBT instance, you will have to implement this integration yourself), but the alert will provide very little context. For example, there’s no easy way to see which records caused the failure.

The end result is that you end up with an overwhelming amount of alerts, each of which is less actionable. No one knows which alert should go to which person, or why the check exists.

### Making the move to data observability

DBT tests and Great Expectations are most useful for finding problems with data ingestion due to problems upstream. For example, prior to a transformation, you can run dbt test on the source table to ensure that everything is as expected.

Data observability, on the other hand, generally monitors your data closer to the destination, once all the transformations have been completed. Unlike testing, it’s more proactive and automated. Rather than waiting for you to assert something, it simply gives you a view into the current state of your data system. A good analogy is that data tests are like getting your engine checked, while data observability is like having a dashboard on your vehicle.

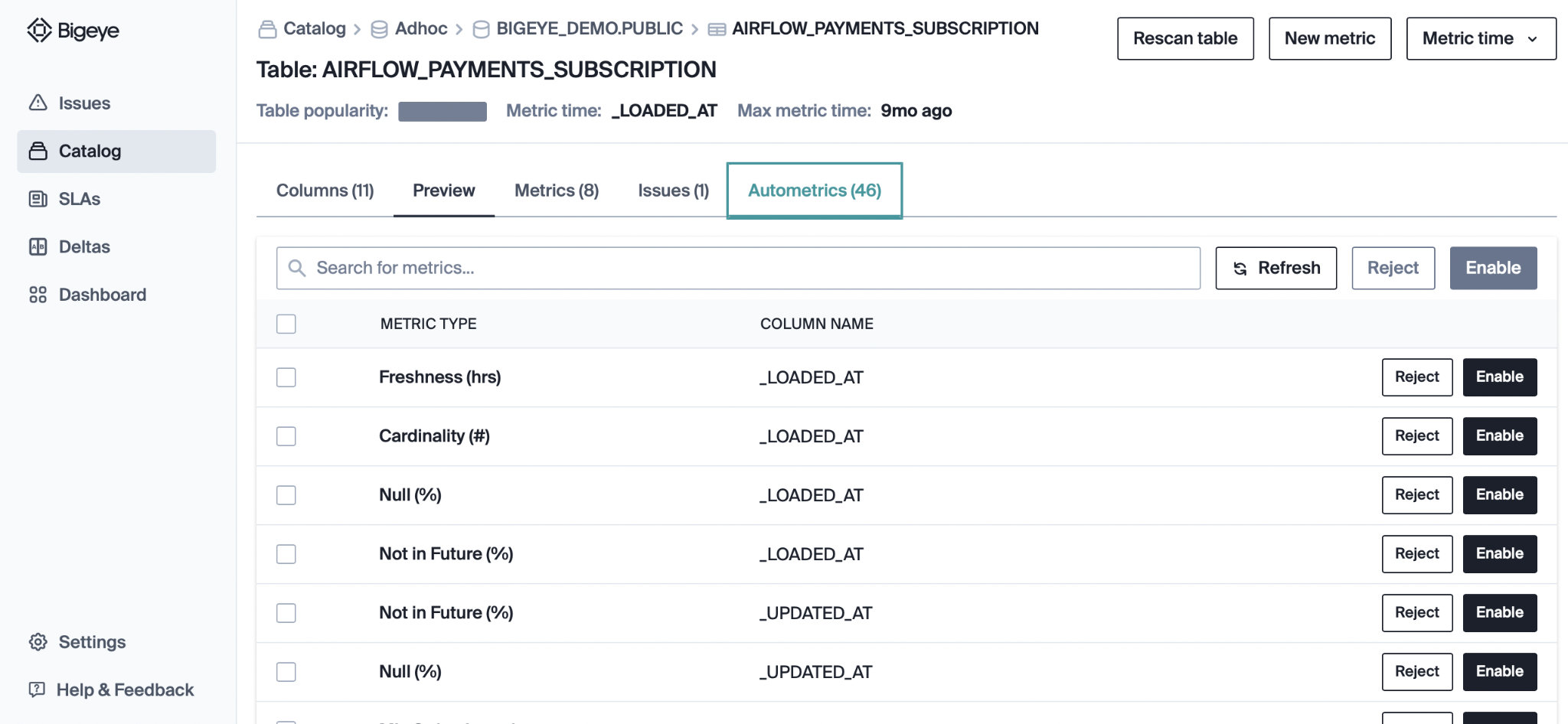
### What you get from data observability

#### Better metrics

When you connect your database, Bigeye begins profiling your data to understand what it looks like. Bigeye can then *generate* Autometrics for the table based on the content of each of the columns. For example:

* + If you’ve got three values in the column, it’s probably an enum
  + If you have no duplicate values, maybe you never want them to be duplicates
  + Maybe it looks like an ID column, which means you’ll want to check for duplicates
  + If the column is full of strings, maybe it’s a column or timedate column.

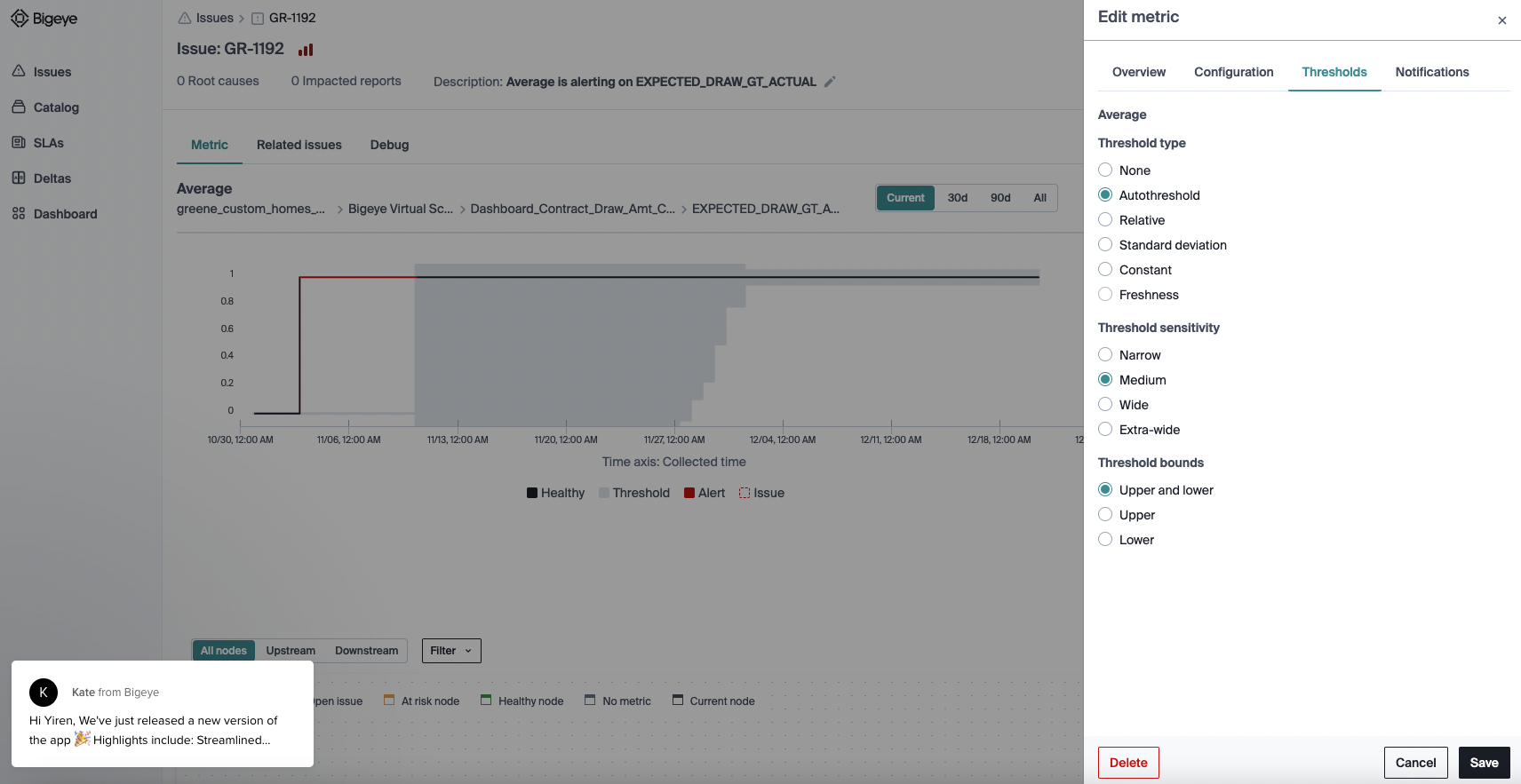
In total, Bigeye has [80+ column-level metric types](http://v) that it tailors to your data and recommends as autometrics. This includes common metrics like percentage null, freshness, and cardinality, and also less common ones, like kurtosis and geolocation.



All of this means that you have more personalized, more comprehensive coverage for less effort. Rather than having to manually set up tests for each model, you simply navigate to your core tables and enable the metrics you believe are relevant.

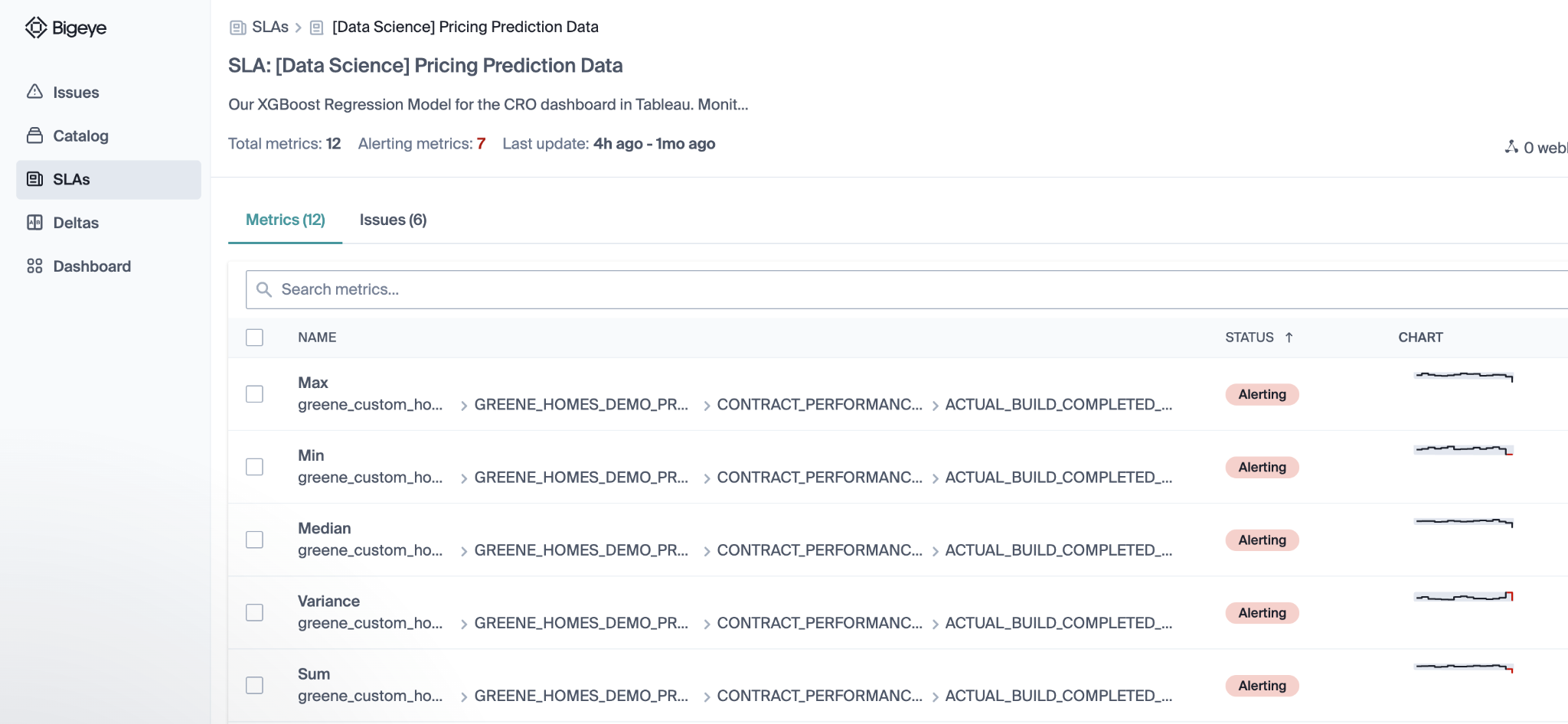
#### Better manageability of metrics

With a data observability solution like Bigeye, metrics are much more manageable. Rather than having to manually set thresholds for all the alerts, autometrics come with autothresholds. These are automatic thresholds calculated from historical patterns.



Auto-thresholds free you from having to manually set, tune, and update potentially thousands of sets of thresholds. Thanks to Bigeye’s anomaly detection engine, these thresholds are also dynamic – they adapt to business changes, seasonality, and your feedback.

Additionally, you can organize your metrics into meaningful groupings with SLA’s, and easily triage issues that arise from a number of different notification points (UI, Slack, email).



#### Suite of other benefits

Finally, a data observability solution like Bigeye provides a suite of additional context that helps you track down the exact origin of your data quality issues.

One of the most common asks by users of DBT tests is whether there’s a way to quickly retrieve failing records: with Bigeye, we provide debug queries within the issue page that allow you to do just that.



The lineage and related issues features, meanwhile, help highlight if the root cause is actually in a table upstream. It also helps visualize potential downstream impact.

