metaplane

The essential data observability handbook

Proven techniques for modern data teams

FEATURING EXPERTS FROM:

SpotOn VERONICA BEARD

Table of contents

Preface

“When people buy a drill, what they really want is a hole.” Harvard marketing professor Theodore Levitt often used this phrase to nudge his students into looking beyond the immediate task at hand to consider the ultimate objective. What’s the *real* goal here?

So, let’s apply the same concept to data. If we’re honest, nobody's excited about managing heaps of data just for the sake of it. The real goal is to **extract value, drive decisions, and power our operations and products with real, actionable insights**.

Today, this isn’t easy: data teams centralize, transform, and expose an increasing amount of data from an increasing number of data sources that serve an increasing number of stakeholders in an increasing number of use cases.

The problem is that **more data means more surface area to maintain**. As maintenance overhead increases, data breakdowns become more frequent and severe. This causes the downstream functions that depend on data to falter, which leads to degrading trust in data by stakeholders.

Enter **data observability**.

> 💡 **Data observability** is the degree of visibility you have into your data at any point in time.

It exists to ensure data quality and increase trust by providing insights deep within your data pipelines and infrastructure and answering these generic questions:

- *Is the data “right”?*
- *How up-to-date is this dashboard?*
- *Why did the report break?*

The purpose of this e-book is to equip data folks like you with everything you need to be a **data observability champion**. We want to empower you to advocate for better data visibility, integrity, and accuracy—starting with the obvious question: Why now?

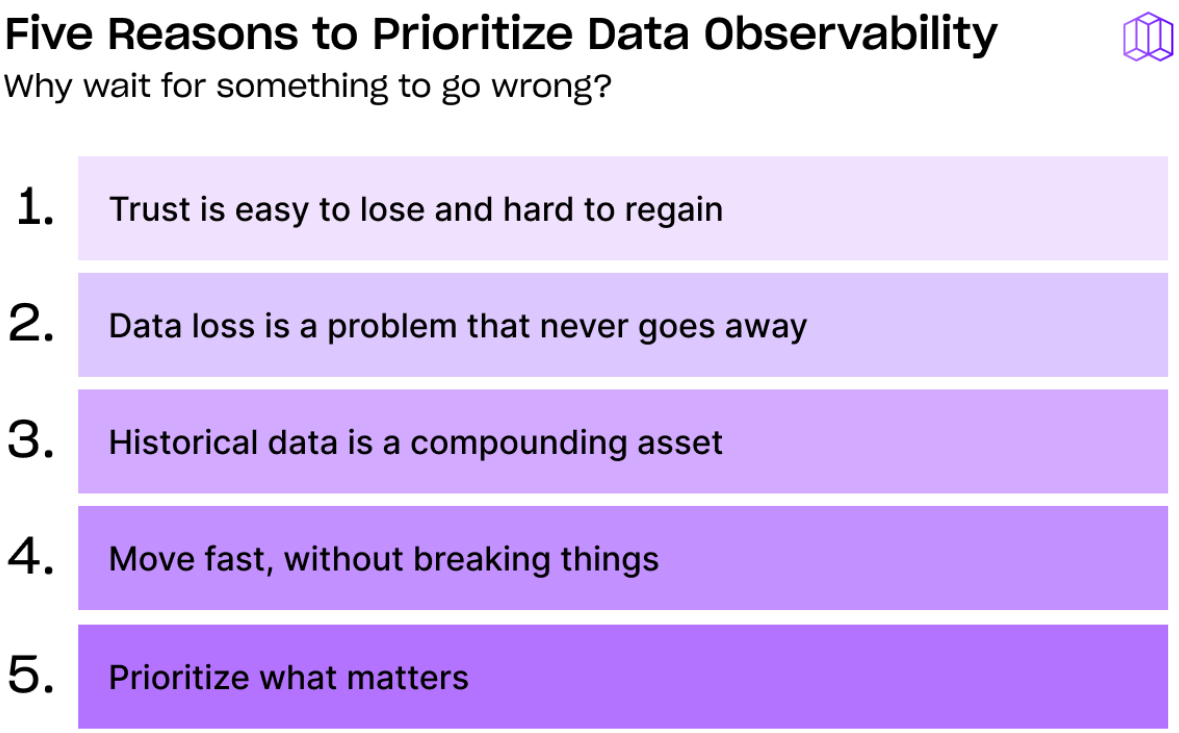
I. Prioritizing Data Observability: Why Now?

After speaking with hundreds of data leaders ranging from high-leverage solo teams at startups to decentralized data teams with 50+ members at Fortune 500 companies, one thing is clear: **data is becoming ubiquitous regardless of the size, industry, or makeup of companies.**

Every company we've spoken to uses data in various ways, **helping drive critical business decisions in product, marketing, sales, and finance**. As data is operationalized, data leaders have shared with us that data quality is one of their top priorities, and that better observability is a sustainable solution.

One of the most common questions we receive is: *when is the right time to become proactive about data quality by building a tool in-house or adopting a data observability platform?*

From our perspective, there are five important reasons why you should consider making data quality a priority now, instead of waiting until it becomes an issue.



Reasons to prioritize data observability range from the immediate cost of data to the abstract, but equally important, need to maintain trust in data.

1. Trust is easy to lose and hard to regain

As data people, our goal is to enable other teams to ask questions and make informed decisions. Two necessary ingredients are **data** and **literacy**. But the other necessary ingredient is **trust**. Without trust in data, stakeholders rely on their own splintered datasets and siloed data usage or second-guess your data. And while skepticism is good; lack of trust is not.

Not only is trust lost with individuals who made incorrect decisions using your team's data, but distrust in data trickles throughout organizations and results in less data-driven or siloed decision-making. Plus, as modern data teams increasingly adopt data mesh architectures and self-serve analytics, new, autonomous teams need to be able to use and trust data to make important decisions in their own domains.

2. Data loss is a problem that never goes away

Data loss is the bane of our existence. Losing data deteriorates trust, creates data debt, and honestly, can be painful. If data is missing from a data model, then it generally complicates joins and transformations, injecting conditionals and logic into what should otherwise be a simple SQL query. When data is used in dashboards, gaps need to be annotated and explained to stakeholders, or you're at risk of making incorrect decisions.

Worst of all: **missing data never goes away**, and backfilling what can be backfilled can be challenging. Every time you onboard a new employee, create or update models, or change dashboards, you need to consider missing data. Of course, s*** always happens, but **proactively monitoring data quality can decrease the frequency of data loss**, and early detection can decrease the severity when it happens.

3. Historical data is a compounding asset

Analytics products **provide a historical record of usage** to answer questions like: how do users use our product? How does the present moment compare to the past? What is the impact of this feature?

Analogously, we see data observability tools like historical baselines (e.g. row counts over time), anomaly detection, and incident reporting, to name a few, unlocking new powers for data teams and helping prove how valuable their work is for the larger organization.
The good news is that, like data itself, **once you begin collecting metadata, it becomes a compounding asset**. Each day imparts a new data point for richer historical comparisons and increased statistical power. That leads to a better understanding of and trust in your data.

4. Move fast, without breaking things

In the pre-observability world, engineering teams commonly deployed infrastructure with simple heartbeat checks. Fortunately, observability products granted detailed visibility into all aspects of infrastructure, so teams could proactively detect system degradation and have the metadata ("traces") needed to debug. This cut down on time along four dimensions: **time to identify an issue, time to diagnose the root cause, time to fix the issue, and finally the time to verify the fix**. All of these steps roll up into time-to-resolution.

Teams don't want to just decrease time to resolve individual issues; they also want to reduce the frequency and severity of issues. With more observability, the root causes of issues can be identified and fixed, helping engineers **spend less time debugging and fixing issues and more time on the things they actually want to work on**.

5. Prioritize what matters

How do you make decisions about what to work on when you don't know what the baseline or bottlenecks are? For example, without knowing the runtimes of your dbt DAGs, which models need to be optimized? Without knowing which tables experience freshness issues, how do you know which transformations to prioritize? Without knowledge of the most important dashboards, how do you know which data quality issue to prioritize first?

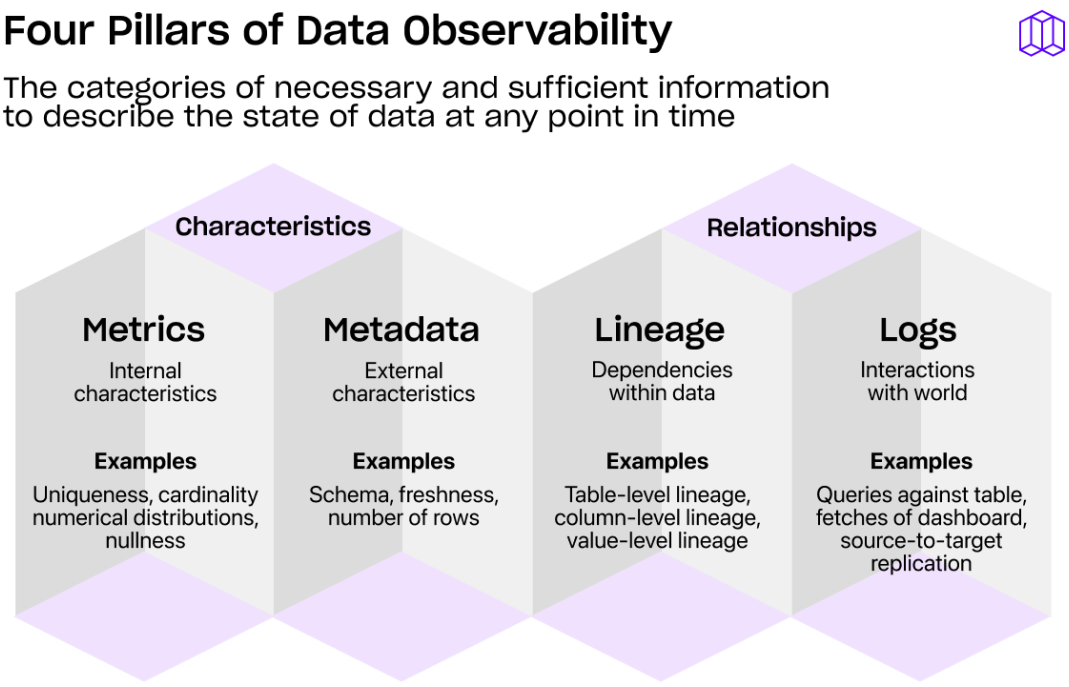
> Data teams can help other teams prioritize their work. But what about our own work? Spending time putting out the most recent big fire is not only **unenjoyable**, but also **unproductive and unsustainable**.

Now that we've established the urgency of prioritizing data observability let's get into the meat of effective data observability practices: **the Four Pillars of Data Observability**, which serve as the cornerstone for building a robust, transparent, and resilient data infrastructure.

II. The Four Pillars of Data Observability

Data Observability draws inspiration from Software Observability, though there are important differences like the lineage between pieces of data and the components of a data system.

Because of those differences, the three pillars of Software Observability don't quite address the needs of data teams, whether they're DataOps, data engineering, data science, data analytics, or analytics engineering teams. Like my old computer science professor would say, compared with software, data management is "the same, but completely different." The overarching similarity, though, is the goal of **increasing visibility into their data systems over time**.



How can we derive the four pillars?

In that spirit, we ask: what can we know about your organization's data to derive its state at any point in time? We add two additional constraints: we want to minimize the number of pillars to be maximally concise while making sure that they're orthogonal to maximize the information value of each.

We look at a concept of thermodynamics: the **intensive and extensive properties** of materials. Intensive properties do not depend on the size of the material. For example, the

temperature, density, pressure, and viscosity of a material do not depend on how much material there is. One cup of water can have the same temperature as an entire ocean.

Metrics: internal characteristics of the data

In the world of data, the analogy to intensive properties are properties of the data itself.

If the data is numeric, properties include summary statistics about the distribution like the **mean, standard deviation, and skewness**. If the data is categorical, summary statistics of the distribution can include the **number of groups (e.g. the uniqueness**).

> Across all types of data, metrics like **completeness and accuracy** can be computed to describe the data itself.

These are all different data quality metrics that describe some aspect that **summarizes the underlying data**, whether they're calculated for data tables at rest in a warehouse or data in transit in data pipelines.

Metadata: external characteristics of the data

In contrast, metadata is an external property, frequently defined as "data about data." We'd take this a step further and add that metadata is "data about data that is independent of the data itself."

Direct analogies to the world of data include properties like **data volume (number of rows), the structure of data (schema), and the timeliness of data (freshness)**.

> While the **volume, schema, and freshness** of data have an impact on the internal metrics, they can be scaled independently while preserving the statistical characteristics.

Conversely, the internal characteristics of data can change without impacting the volume, schema, or freshness. Together with metrics, metadata can be used to **identify data quality issues.**

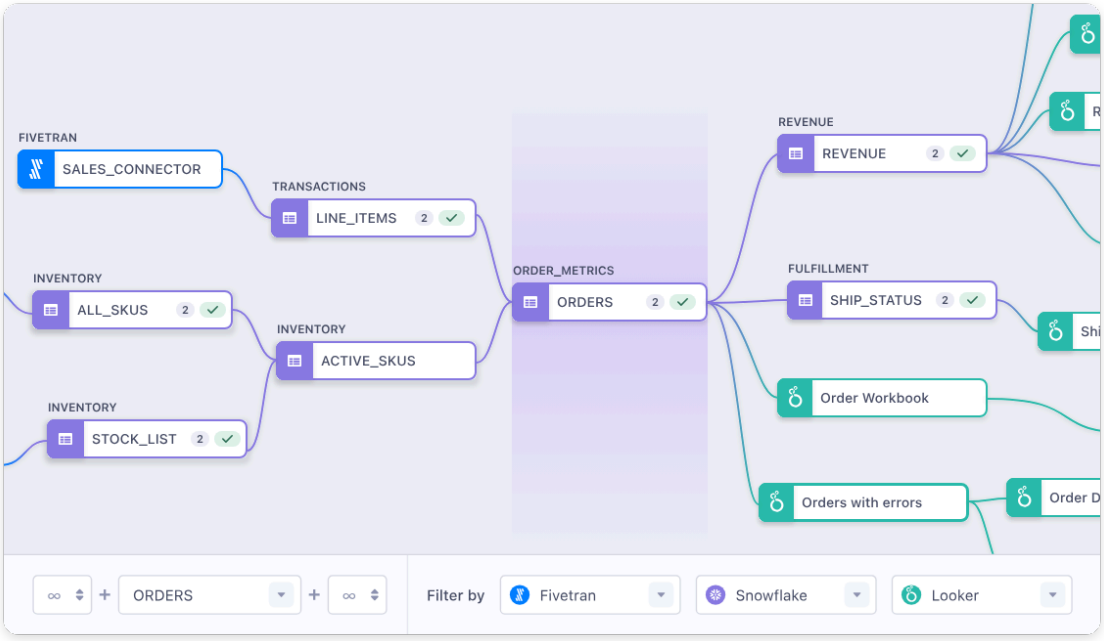
Lineage: dependencies between data

Using metrics and metadata, we can describe a single dataset with as much fidelity as we desire.

> However, datasets in the real world often **do not exist in isolation**, landing in a data warehouse with no relationship to each other.

We can draw another analogy from the physical sciences, where systems can be modeled within themselves, but our understanding can be enriched by modeling interactions. For example, thermodynamic systems have smaller components with internal interactions, and also have **interactions with the external environment**.

Within the data world, the primary internal interaction is the derivation of one dataset from another. Datasets are derived from upstream data and can be used to derive downstream data. These **bidirectional dependencies** are referred to as the lineage of data (also called the **provenance**), and range in level of abstraction from lineage between entire systems (this warehouse depends on those sources), between tables, between columns in tables, and between values in columns.



Example of column-level lineage visualized in Metaplane (Source: Lineage & Impact Analysis)

Logs: interactions between data and the real world

With metrics describing the internal state of data, metadata describing its external stage, and lineage describing dependencies between pieces of data, we're only missing one piece: **how that data interacts with the external world**. We break these interactions into machine-machine interactions and machine-human interactions.

Machine-machine interactions with data include movement, like when data is being replicated from data sources like transactional databases or external providers to an analytical warehouse by an ELT tool.

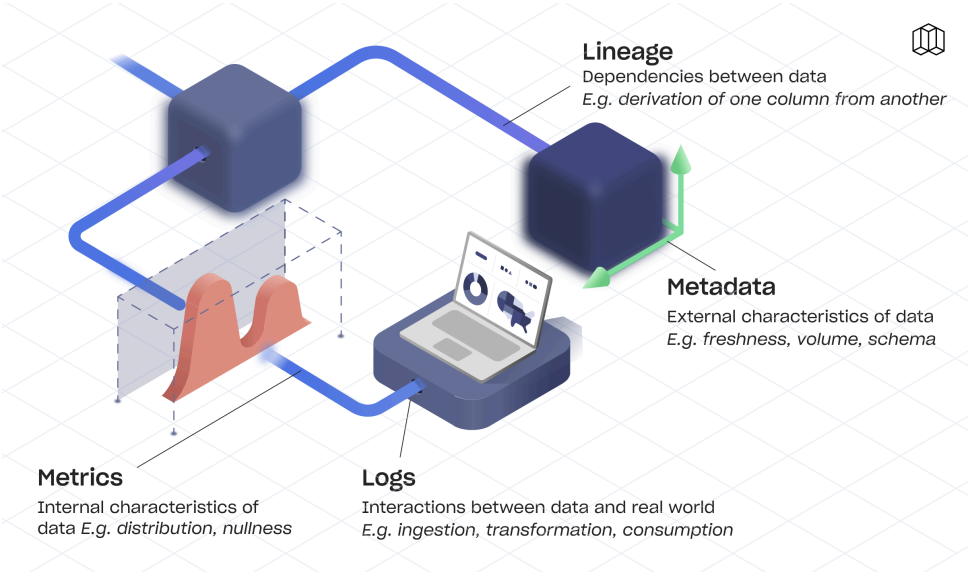
Interactions also include **transformations**, like when a dbt job transforms source tables into derived tables. Logs also document **attributes of these interactions** (e.g. the amount of time that a replication or transformation takes, or the timestamp of that activity).

> Crucially, **logs capture machine-human interactions between data and people**, like data engineering teams creating new models, stakeholders consuming dashboards for decision-making, or data scientists creating machine learning models.

These machine-human interactions contribute to an understanding of who is responsible for data and how data is used.

Putting it all together

With metrics describing the internal properties of data, metadata describing the external properties, lineage describing the dependencies, and logs describing the interactions, we have four levers that we can pull to fully describe the state of our data at any point in time.



The four pillars of data observability: metrics, metadata, lineage, and logs.

Without any one of the pillars, our ability to reconstruct the state of data is incomplete. For instance:

- **Without metrics**, we don't know about the internal properties of the data itself, making alerting based on real-time anomaly detection on metrics impossible. If we only had metadata, we would know the shape, structure, and timing of data, but not necessarily whether we had bad data.
- **Without metadata**, we don't know the structure, structure, or timing of data, making use cases like schema change detection or satisfying Service Level Agreements (SLAs)—which are critical to improving data reliability and decreasing data downtime—impossible. If we only had metrics, we would know whether the data is correct, but not necessarily if it was refreshed in an appropriate amount of time
- **Without lineage**, we don't know how different pieces of data depend on each other, making it difficult to conduct triage workflows like upstream root cause analysis and downstream impact analysis. If we only had metrics and metadata, we have a holistic view of the health of data, but an incomplete picture of how issues are related
- **Without logs**, we don't know how external systems like ELT/ETL and transformation tools impact our data and how external users are impacted by our data. We know how data is related (via lineage), but not necessarily how important those relationships are or who is responsible for upstream changes.

Missing any one of four pillars would be a meaningful gap in a data observability tool or program. But at the same time, other categories of information are redundant with one of the four. Therefore, these four pillars are necessary and sufficient.

Putting the pillars to work

Getting started with building up these pillars is a project of not letting the perfect get in the way of good. Most data teams in 2023 have little information about their systems, so the first step is to **just get started**. Here's how to get going:

1. Start by identifying the **most important tables and the data quality metrics** from those tables. Typically the easiest place to start is with data at rest within a data lake or data warehouse, then extending upstream into data in motion. Like with the rest of the pillars, profiling at one point in time is useful but insufficient for spotting poor data quality, requiring periodic checks over time.
2. Metadata is often provided out of the box by your data warehouse, with warehouses like Snowflake and Google BigQuery providing snapshots of the row count, schema, and last update time of tables in INFORMATION_SCHEMA. You can begin by **storing that metadata in a separate table and then tracking changes over time**.
3. Lineage can be a bit more challenging. Inferring the lineage between tables and columns is a difficult task that involves detailed query parsing, but some open-source libraries provide a starting point. If you use a workflow tool like dbt or dagster or Prefect, you're in luck—you can **start ingesting the transformation metadata generated by those systems**. Data integration tools like Fivetran also increasingly provide lineage metadata.
4. Storing and parsing logs is highly dependent on the tools in your data stack, though often ELT/ETL and BI tools in the "modern data stack" will provide an API for accessing those logs, and **your warehouse will likely store query history**. Sometimes, these APIs will even provide you with analytics about when and how data analysts and stakeholders are using your data products.

Now, if you don't have the bandwidth to build systems that collect metrics, metadata, lineage, and logs of your data assets, don't sweat it. There's a whole ecosystem of commercial and open-source data observability solutions that automate the collection of this information, synthesize it into a usable form, and integrate it with the tools you already use.

And that's exactly what SpotOn capitalized on to save engineering time and continuously receive context about potential root causes and downstream impact when data incidents arose.

Case study: How SpotOn reduced time to actionable data using Snowflake, dbt Cloud, and Metaplane

SpotOn

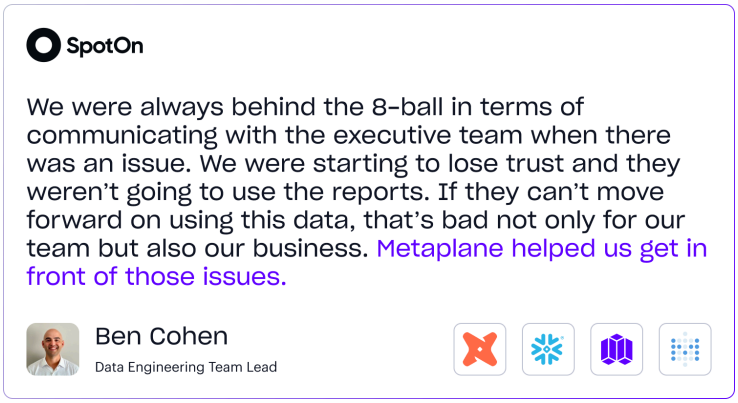
Industry
Mobile payments

6x
Decrease in time to actionable data

Size
1200 employees

8.5x
Increase in engineering contribution

SpotOn is a rapidly growing business that offers mobile payment processing and management software for restaurants and small businesses. As the company has scaled, data has increasingly become a differentiator to drive the business forward. More than 500 team members rely on data to make daily decisions and SpotOn's customers rely on data for merchant reporting and a recommendation engine to power better online ordering experiences.



With this widespread integration of data across the business came new challenges for the data team. Ben Cohen, the Data Engineering Team Lead at SpotOn, and his team were running into bottlenecks in the performance, accessibility, and engineering workflows for iterating on data.

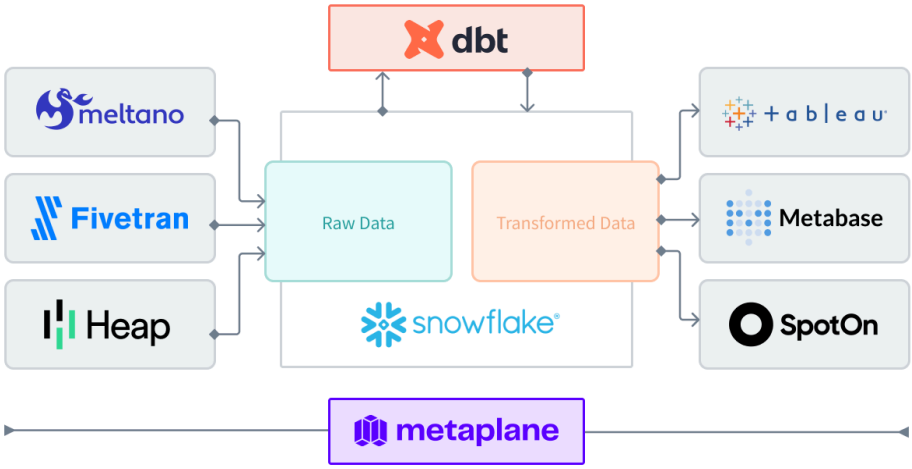
The performance of their Postgres database was routinely slow, causing delayed ETL jobs and degraded BI reporting

experiences. The database was undersized and tuning was difficult; data was either missing or delayed.

These performance issues had a ripple effect—data was not accessible because it was often slow or broken. Users were unable to run queries because resources were being consumed by upstream ETL jobs for several hours every morning. New and advanced analytics use cases were impossible to create on top of the existing warehouse because Postgres couldn't handle reprocessing large-scale data aggregations.

With these challenges in mind, Ben decided to implement Snowflake, dbt, and Metaplane to scale the analytics capabilities and create a new team culture, all without adding complexity or cost.

Scaling analytics capabilities with Snowflake, dbt Cloud, and Metaplane



SpotOn built a modern data stack using tools like Fivetran, dbt, Snowflake, and Metaplane.

When Ben and his team migrated from Postgres to Snowflake, new advanced analytics use cases were immediately possible. For example, the recommendation engine for online ordering platforms is powered by large-scale pre-aggregated data that is augmented from multiple sources. Whereas Postgres could not support these types of aggregations, Snowflake was able to handle this gracefully because of the scaling capabilities provided by the separation of storage and compute.

Snowflake also became a key part of data integration after SpotOn acquired Appetize, who was also already a Snowflake customer. The data team was able to securely and instantaneously share data between SpotOn and Appetizes' Snowflake instances. The performance improvements led to faster reporting load times and more advanced analytics use cases. To make this data even more accessible and improve engineering workflows, Ben and his team migrated all existing models to dbt Cloud—financial, operation, sales, and product analytic data were all transformed using dbt. With this core business logic in dbt, the data team's workflow became much easier due to greater collaboration and accessibility.

SpotOn

As stakeholders use more data and have new capabilities, they ask more from your team, and you need to move quickly. You can't test everything yourself. The other way to scale is to hire more people that add data quality checks, but that doesn't scale well from a cost and efficiency standpoint.

Ben Cohen

Data Engineering Team Lead

After SpotOn switched to dbt Cloud, creating data models went from taking days to hours. dbt quickly became the workhorse that powered the entire SpotOn warehouse, internal BI, and analytics. But as they improved performance and made the data more accessible, Ben noticed that they created a data feedback loop: more capabilities allowed the entire organization to move more quickly, which resulted in more teammates asking for data and analytics.

On the one hand, this feedback loop was driving SpotOn's entire organization to leverage data and make more informed decisions. But with this came more attention and scrutiny to the data team's work, and trust in data became top of mind for the data team.

Metaplane: the final piece of the puzzle

Metaplane was able to help the SpotOn data team scale this feedback loop. By providing observability across their data stack, the team was able to build and retain trust so the feedback loop and usage could continue. With Metaplane's machine learning-based testing approach and ability to automatically add hundreds of tests, they saved engineering time and always received context about potential root causes and downstream impact when data incidents arose.

By proactively catching data incidents, Metaplane helped Ben's team get in front of any issues that would impact downstream stakeholders, helping the data team regain trust in

the data. After receiving data incident alerts, Ben and his team could pull back scheduled reports until they were able to verify that the data was fixed after an issue.

Ben's team went from chasing down data bugs and data anomalies to proactively finding out about them and spending more time actually fixing the issues. Time-to-identify data quality issues went from hours or days to seconds.

👉 Read more stories at: metaplane.dev/customers

Though they now have an ideal data stack, Ben's team started with little more than the four pillars of data observability.

SpotOn

Metaplane is key to preserving trust in our data. You've spent so much time to move to this great modern stack, but if the end result is you lost people's trust and they won't use it, that work is for nothing.

Ben Cohen

Data Engineering Team Lead

Now, if you're wondering how SpotOn decided whether they should build a tool themselves, borrow an open-source tool, or buy an out-of-the-box tool, here's the consideration framework they (and many others) followed.

III. Should You Buy, Borrow, or Build a Data Observability Tool?

Developers often debate whether they should buy or build the software they need to do their jobs, and data engineers are no exception.

Now, here's the tough news: **There is no one-size-fits-all answer to the debate**. Instead, you must consider all relevant factors when making a decision.

To make things a little easier on you, we'll lay out the pros and cons of building your own tool, leveraging an open-source tool, or buying an out-of-the-box tool, considering everything from time and money to customization and compliance (and everything in between).

Option #1: Build a custom, in-house tool

Building a custom, in-house data observability tool is exciting to some but daunting to others. Like any initiative, it comes with clear benefits, costs, and risks.

👍 Benefits

1. **Context:** When you build something from scratch, you have an intimate understanding of what it is and how it works.
2. **Customization:** Building in-house allows you to design a tool that meets your organization's unique needs. It's a great option if you want to embed the technology into a custom workflow or build features that aren't widely available!
3. **Compliance:** If security and compliance are important to your company, you may prefer to build in-house. After all, that's the easiest way to retain custody of your data.

💰 Costs

1. **Time:** Developing a custom-built tool requires lots of engineering hours, both upfront and on an ongoing basis. So, unless you have a dedicated data infrastructure engineer, this work will always be a distraction from your team's core responsibilities.
2. **Money:** While not as expensive as commercial solutions, custom-built tools do cost money. You need to pay for hosting and account for the wages of the team members who build the tool (both upfront and on an ongoing basis).
3. **Expertise:** When you build in-house, you lose out on the product and engineering expertise that commercial and open-source tools have to offer (and which have been pressure-tested by countless organizations).

😬 Risks

You will probably underestimate the number of engineering hours it will take to build and maintain the tool. It's also likely that your team doesn't have the skills necessary to optimize the uptime and stability of the application. After all, they aren't DevOps experts.

Option #2: Leverage an open-source tool

Using an open-source tool to build a custom application can offer the best of both worlds, but it can also be a risky venture.

👍 Benefits and Costs 💰

1. **Time:** By leveraging an open-source tool, you can spend much less time building the application, but make sure you account for the added time spent integrating your custom solution with that other system, plus the hours it takes to continuously update to the latest version.
2. **Money:** Like with custom-built tools, you must pay for the hosting yourself. You'd also need to pay the open-source tool provider which often increases over time.
3. **Support:** You don't get commercial-level support when you leverage an open-source tool. No one will tell you exactly how to fix your problem, but you will get access to a passionate community of developers who regularly report known bugs and openly share their knowledge with each other.
4. **Customization:** You may be able to customize your tool to meet your needs—or you may not. It really depends on the architecture used by the open-source tool.

😬 Risks

It's not unusual for open-source tools to go dormant. If supported by a for-profit company, the company's strategy may change, or they may simply go out of business. If supported by an individual or group of individuals, their passions may change, or they may direct their attention to more profitable ventures.

Option #3: Buy an out-of-the-box tool

Buying a commercial tool is the easiest way to adopt data observability. You pay money, and you gain access to ready-made software.

👍 Benefits

1. **Time:** Buying an out-of-the-box tool requires the least amount of engineering hours and offers the fastest time to value. In fact, you don't have to build anything. All you need to do is purchase, implement, and configure the tool to suit your organization's needs.

2. **Expertise:** When you buy a tool, you automatically get the product and engineering expertise that comes with it. With professional software engineers continuously monitoring and improving the uptime of the application, most days you'd have nothing to worry about—and any downtime that *does* slip through would be minimal compared to an in-house tool.
3. **Features:** Commercial tools typically have larger feature sets than in-house tools, since they must cater to a wide range of customers. Data observability tools like Metaplane offer testing and anomaly detection, schema change and job monitoring, as well as lineage and usage analytics features, among others.
4. **Support:** No tool is perfect, and commercial tools are no expectation. But when something goes wrong, technology companies typically offer some type of support. Here at Metaplane, for example, we create shared Slack channels to ensure your concerns are addressed as quickly as possible, and offer service-level agreements (SLAs) to ensure we meet your expectations.
5. **Longevity:** Commercial tools are more likely than either open-source or custom-built tools to still be around years from now. Their founders and leaders are motivated to make their companies successful, whereas a new employee may not care enough to maintain your in-house application and the company that sponsors the open-source tool may stop supporting it.

💰 Costs

1. **Money:** Purchasing an out-of-the-box tool is the most expensive option. If you have the cash to spend, it's a great solution—but many data teams still struggle with tight budgets.
2. **Customization:** Commercial tools offer the least amount of customization. That said, you can always provide product feedback and make requests. You just can't guarantee that they'll be answered in a timely manner.

😬 Risks

While it's more likely that a commercial vendor will still be around 10 years from now, it's possible that they could go out of business or be acquired by another company with different priorities.

A recommended order of operations

At the end of the day, you know what option is best suited to your needs. If you're unsure, consider this order of operations: **Look into commercial options first, open-source options second, and in-house options third**. You know better than we do how busy data teams are. They usually lack either the size or capacity to build a custom, in-house tool.

If that's true for you, going with a commercial option is probably your best bet. It has the most benefits, the fewest costs, and the least risk. On the other hand, if customization and compliance are critical to your team's success, or you have a limited budget but a larger team with the time it takes to build from scratch, a custom, in-house solution may be a better option. Only you can make that decision.

But if you *do* opt for an out-of-the-box tool, you might be wondering, *"How do we justify an investment in data?"* The folks at Veronica Beard had that question, too. Here's how they did it.

Case study: How Veronica Beard set up a data stack (almost) as good-looking as their clothes

VERONICA BEARD

Industry
Fashion

Size
340 employees

Business leaders **trust data quality** for decision-making

Eliminated errors while merging pull requests

Max Lagresle, Veronica Beard's first data hire and currently the Director of Data, was originally brought in to implement Segment as a standalone Customer Data Platform to help the eCommerce and Marketing teams better understand how to drive more sales. In the process of setup, he found that there was more opportunity for Veronica Beard to grow through the use of data. One notable example that stands out is the need for leadership to understand company-wide sales. They were willing to put up with one entire week of latency required for financial reconciliation across those CSVs.

Setting up a net-new data stack

It's hard to get budget for new initiatives and tooling. When Max first joined Veronica Beard at the end of 2019 and pitched a data stack implementation, he had to answer "How do we justify an investment in data?"

This boiled down to 3 core questions:

- *What's the scope of this project and who are the owner(s)?*
- *What human resources and technical skillsets do we need for this project?*
- *Which tools are you recommending, and what's the order of implementation given their use case(s)?*

VERONICA BEARD

By starting with the ecommerce and marketing departments, we were able to prove that we could answer their questions and use cases. In the process, we were also able to demonstrate how much easier it was to answer questions that couldn't even be answered before setting up a warehouse, such as the difference in lifetime value between single-channel customers (e.g. ecommerce or retail) vs multi-channel customers (i.e. omnichannel).

Max Legresle
Director of Data

VERONICA BEARD

We wanted a solution that we could grow with, and the Metaplane team and product has been amazing to work with so far. Metaplane's team are true experts in their domain and data industry in general and their white-glove onboarding has been truly amazing.

Max Legresle

Director of Data

Over the course of a year, Max was able to successfully implement a warehouse to answer recurring questions and address questions that were previously too cumbersome to answer through Excel. Fast forward 4 years, and in addition to the eCommerce and Marketing teams, the data team now also supports 6 additional departments (CRM, Planning & Buying, Retail, Merchandising, Finance, Wholesale).

Using Metaplane to guarantee data quality

At this point, Max and the growing data team are trusted by multiple departments to handle questions such as:

- *How do we think about planning development?*
- *What should store inventory levels be at for maximum opportunity capture?*
- *How can we improve company operational efficiencies and spend money wisely?*
- *What sort of product lines have done well and how should we plan for the future?*
- *Where are the biggest opportunities for store opening?*

As you can imagine, these questions had direct impacts on the company's bottom line, which was why they began to evaluate data observability solutions to ensure that questions were answered accurately.

Like other teams, they had already been using dbt tests alongside their model builds, but ran into problems with:

1. Scaling the number of tests that they had, both in terms of types of metrics and coverage for all of their new models to support additional departments.
2. Interpreting the nuance in results so that they wouldn't just be alerted to whether there was a null value, but seeing what % of null values would be considered normal, given past history.
3. Data quality issues being created throughout the day, but only being tested at the beginning of each day during a dbt build.

Despite evaluating other vendor solutions, Max and his team ultimately chose Metaplane due to being able to solve the issues above, commercial agreement flexibility that could accommodate Veronica Beard's growth, their interactions with the Metaplane team, and the ease of use in the product.

Now, more than a full year after their initial Metaplane implementation, the data team at Veronica Beard has taken advantage of all of the new offerings that have come out since they started. In addition to our core feature, data quality monitors, Max specifically mentioned:

- Data CI/CD - This is the feature that broadened his initial scope of data observability evaluation beyond just data quality monitoring. With a native Github app, Metaplane is able to forecast how downstream tables and BI dashboards would change given an update to a dbt model.

- Lineage - Metaplane uses metadata to generate column-level lineage so that users can understand where their ingestion tool is loading data down to which business intelligence dashboards are impacted by a data quality incident.

VERONICA BEARD

Data is all about trust. Every time the business users identify an issue that the team hasn't proactively found, we naturally lose a little bit. It's not a problem with making a mistake; everyone makes mistakes. But being able to tell others (about an issue) first can actually flip the situation and increase trust, which Metaplane's helped us with.

Max Legresle

Director of Data

👉 Read more stories at: metaplane.dev/customers

When it came time to select a data observability tool, Veronica Beard chose Metaplane. But how did Max take a thoughtful, structured approach to evaluating the many tools on the market? He followed the below framework.

IV. How to Evaluate Data Observability Tools

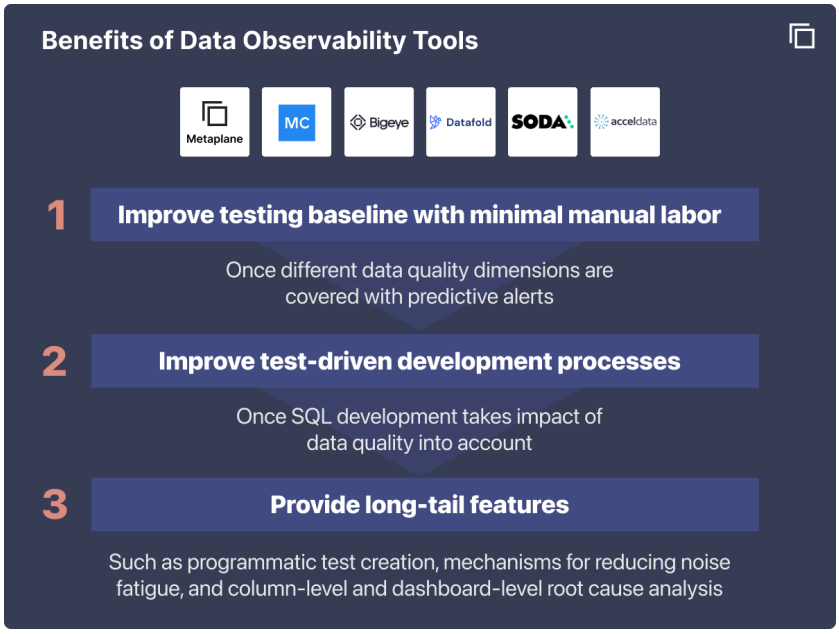
Data engineering teams far and wide are swamped with massive tech debt, a labyrinth of dependencies, and ambitious roadmaps. These roadmaps usually cater toward more obvious goals: creating curated datamarts for analysts and data scientists or establishing robust data pipelines to new data sources.

As a result of massive backlogs that are heavy on feature building, **stakeholders of data engineering teams often discover data quality issues before engineers do**. This erodes trust in the data platform. And the less reliable that data is for analysts and data scientists, the more likely that they will go directly to source data with poor-performing queries to do their job. **Stakeholders need confidence, and engineering teams have to earn it.**

Benefits of data observability tools

While most data teams write some tests in their ETL orchestration tools, they're often too basic to accurately account for the complexities of data quality (like machine learning for row count thresholds and freshness variability).

Even if you write basic testing for every column and table, tweaking and fine-tuning tests can be a massive waste of precious engineering time, making the testing unfeasible without automation. Don't get me wrong, dbt is a great start! But there are **no super easy ways to programmatically interact with tests**, and don't get me started on the amount of YAML...



Benefits that a data observability tool should provide

There are 3 main things to look at when you run a proof-of-concept (PoC) on data observability tools:

1. **Baseline Testing Improvements →** A tool should improve your baseline testing and alerting strategy by utilizing predictive models to describe anomalous behavior with machine learning-based anomaly detection. While dbt is great at what it does, it's not smart enough to meet your data quality needs at scale. Here are some criteria to consider:
   - ☐ *Can it self-update its alerting behavior based on how your data changes?*
   - ☐ *Can your team provide input on specific data points to adjust model behavior?*
   - ☐ *Can it cover different data dimensions of data quality including freshness, row counts, and completeness (e.g. nullness)?*
   - ☐ *Will it be able to scale for higher data volumes associated with complex and growing data ecosystems?*
   - ☐ *Does it give you a realistic, scalar set of acceptable values for your data quality metric?*

2. **Test Driven Development →** Ideally, a Data Observability tool will help orchestrate complex tests on custom cadences and help facilitate test-driven development (TDD). Because testing strategies need to be comprehensive in order for data to be trustworthy, here are some criteria to consider when you're evaluating tools:
   - ☐ *Does the tool integrate with your existing development environment?*
   - ☐ *Can you use it to write, execute, and automate tests as part of the SQL development process?*
   - ☐ *Can you customize test schedules to align with your development cycles?*
   - ☐ *Does it offer comprehensive test coverage—including unit and regression tests—to thoroughly validate data integrity, performance, and business logic within your data pipelines?*

3. **Feature Priorities →** A tool will provide better out-of-the-box features than dbt (or whatever ETL orchestration tool your team uses), but which ones do you care about? Simple pass/fail testing won't cut it when your team size and table counts double, so the more features a tool has the better! Ideally, a tool can provide a global view of the health of your data that satisfies these criteria:
   - ☐ *Will your tool be able to ensure data quality for values of a column both within the context of that table AND within logical groups of that table?*
   - ☐ *Does the tool minimize false alarms and provide precise, actionable alerts that enable your team to respond effectively to real problems?*
   - ☐ *Does it offer comprehensive data lineage capabilities so you can quickly diagnose and rectify issues (even in complex data pipelines)?*

Scope of Evaluation

PoCs for other tools usually require less data, where you explore a smaller sample in greater depth. But data observability tools demand **a broad implementation** in order to be successful. If you PoC an observability tool against 10 data tables or data assets in 1 data system, odds are that the models won't fail during a trial period. In order to get an accurate understanding of how healthy your data is, you need a pulse on all of it. Every organization has imperfections in their data, it's finding them that is challenging

> **The length of the trial period for data observability platforms depends on how long the models take to train against historical data.** Most tools take about 1-2 weeks to sufficiently train. Add about 2 more weeks of monitoring alerts and exploring UIs, you're looking at about 30 days to get a grasp on the "bones" of the tool and how your team can leverage it.

Strategies for Evaluation

Here's a quick and easy way to start the evaluation:

1. **Manually trigger errors to see how different tools interpret the same problem.** It's up to you to evaluate whether it's worth triggering this sort of alert, or whether there's enough errors occurring already.
2. **Get frequent team input to help encourage adoption** once you finalize your selection. The more people that are excited about better quality data, the faster you'll get to a healthy state.
3. **Make sure alerting is smart to be scalable**. *Are notifications easy to configure? Does this tool route the right errors to the right channel? What dashboards are affected? Are expected values flagged as anomalous?* The last thing you need is a tool that alerts you too often over discrepancies that are too insignificant. Crying wolf can numb the team to real outages and bad data.

Notification fatigue has never been more present in the remote age, and making sure that your team can develop a workflow for monitoring alerts is crucial, so having a flexible tool is key. When your alerts are expected, you want to make sure there are easy ways to relax a model's sensitivity. For unexpected alerts, you need to make sure that a tool can successfully facilitate team intervention. Of course, no one (and no tool) is *perfect*. We all make mistakes, but here are some you definitely want to avoid when implementing data observability in your organization.

V. Mistakes to avoid when implementing data observability software

There's a lot to take advantage of with your new data observability platform—beyond finding the issues that you are aware of, you can discover others that were previously unknown, use context to prevent mistakenly creating future issues, and even enable others outside the team on data.

But while you're out exploring this new territory, **here are 5 mistakes you'll want to avoid during implementation.**

1. Lack of Clear Objectives

Your company's finance policy requires you to answer "How do we know your implementation was successful?" while requesting a budget. The first, natural thought that comes to your mind is "If we catch a data quality issue."

But then you think about your morning commute—more specifically, waving goodbye to your family through the front door camera. You installed that device last week and haven't caught any burglars, but have definitely slept better as a result because you know it's pointed at the statistically most likely place your house is broken in from. Drawing parallels from your security camera implementation, it can be helpful to break down your objectives in a similar way.

> Think about installation (i.e. creating an account and integration) as your first step, and then think about where you'd want to place data quality monitors based on **which objects are the most critical to your organization**.

If you want to artificially create an "incident" through data manipulation, you can add a column that'd trigger a schema change alert, which also has the added benefit of confirming "installation". Continue breaking down the rest of your objectives in this way.

The last note here is to be cognizant of your timeline. You'll want to ensure that any stakeholders have time to acclimate to their roles during the implementation period.

2. Neglecting Data Quality Issues

Begin with what you know but don't constrain your thinking as a result of it. While it is important to understand how you'll be able to capture "that" issue next time it occurs, it's also important to remember how you got there.

There's a good chance that you already have unit testing set up, but maybe you weren't testing for that particular type of issue, the test frequency had too large of a gap, or you simply didn't have unit testing enabled for a particular table.

All of those considerations are meant to help expand your scope beyond "find X issue on Y table." Many companies will choose to **implement freshness and row count monitors on "landing zones"** (e.g. ingestion zones, bronze schemas) as they're usually early indicators of data quality issues, and can often **discover silent data bugs**.

3. Overlooking Scalability Requirements

One of the downsides of a simple setup is that eager users may choose to monitor for every single type of anomaly (20+ with Metaplane!) on every single table in the warehouse. While Metaplane does support automatically adding freshness and row count monitors for every object, we don't recommend this for larger organizations with hundreds or thousands of tables, due to alert fatigue—it can be hard to cut through the noise.

> While you *could* find every potential data quality issue in your warehouse, an important part of implementation is **being rigorous about what alerts count as incidents when considering the object in question**.

Over time, you'll want to be stringent about re-evaluating your monitor types and placement, both to ensure that new objects have coverage, but also to **evaluate whether your old monitor placements are still relevant**.

As a bonus, while evaluating your old monitor placements, **consider how they're sampling data**—if it's directly querying tables as a necessary step to sample values within a column, your warehouse compute will also be affected.

4. Ignoring User Training and Adoption

One often overlooked consideration is **user involvement**. Beyond familiarizing people with navigating the UI, users, even those who sit outside the data team, should be involved in improving data quality. The common ground between all teams is usually found in a place like Slack or Microsoft Teams.

As a result, in addition to some training on platform usage for data team members, it's also useful to create a **clear notification channel strategy** that also incorporates business users. This time of training can include: what slack channels receive alerts, what types of alerts are sent (e.g. freshness, schema change), and roles of people involved in the slack channel.

5. Neglecting Continuous Monitoring and Optimization

By default, your data quality monitors created through Metaplane will run at the **frequencies that you specify in the platform**, enabling continuous monitoring by default.

> While our machine learning models lead the industry in anomaly detection, **no machine is at a point where it understands your business as well as you do.**

As a result, especially when a monitor is being set up for the first time (i.e. early after it's finished its training period) or when you've changed how your data behaves or looks, it's important to **provide feedback to the model**.

That feedback can be given to Metaplane's machine learning models either through Slack or the app itself by marketing particular data points as normal to inform the acceptable range.

Implementation Tips & Best Practices To Follow

A successful implementation of a data observability platform like Metaplane could be summed up with "**We caught a revenue-impacting incident on the first day!**" But that'd be discrediting all of the work that you've done to get it to catch critical incidents.

Instead, following these implementation best practices can not only help you to capture those P0 incidents, but also help you outline your work:

1. Outline capabilities
2. Integrate your entire data stack
3. Create clear objectives with a focus on critical objects and not root causes yet)
4. Review rollout
5. Review again as you continue to succeed and scale

Now, we've come this far. We know the ins and the outs of data observability in theory and in practice. It's only fair that we tackle your final burning question: What's the ROI of data observability?

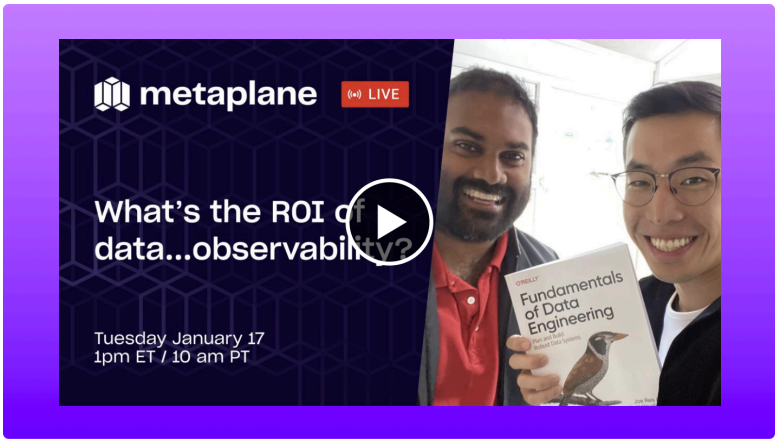
VI. What's the ROI of Data...Observability?

Unsurprisingly, the ROI of data observability and the ROI of data itself are closely intertwined—the **full potential of data as an asset can only be unlocked when it's supported by observability frameworks.** Because here's the thing: data, no matter the quantity, holds limited value if its quality, reliability, and contextual relevance are not continuously monitored and maintained.

> Without data observability, organizations base critical decisions on **flawed or outdated information** (and diminish the potential ROI of their data initiatives along the way).

Now, we won't give away all the answers. To learn everything you need to know about the ROI of data observability, **check out the previously live data chat** between David Jayatillake and Kevin Hu below! In it, they tackle some tough questions like:

1. *What's the return on Data Observability?*
2. *What's the investment required for Data Observability?*
3. *Is that return on investment (ROI) worth it?*
4. *How is the ROI of Data Observability related to the ROI of Data itself?*
5. *What is the ROI of Data? (this deserves a chat of its own)*



👉 Watch the recording 👈

What's next?

We like to boil down the importance of data observability into one question: *are any teams making business or product decisions based on the data your team ingests, stores, transforms, or visualizes?*

If the answer is "yes", then adding **observability should be a requirement** for this quarter.

But we've been there—sometimes teams don't have enough bandwidth and wait until they are inundated with P0 issues before they prioritize data quality. We, as data engineers, need to get ahead of data debt and create reliable and resilient systems so that we can **empower our teammates to make accurate and business-improving decisions every day.**

If you aren't convinced of the importance of data observability, we'd love to hear your opinions and take you out for a virtual coffee. If you are, now is a better time than ever to explore any of the many open-source or commercial tools in the space.

For high-leverage teams that want a fully managed solution with end-to-end lineage that doesn't cost more than your warehouse, you can try Metaplane for free or book a demo for your specific use case!

metaplane

Learn more at metaplane.dev