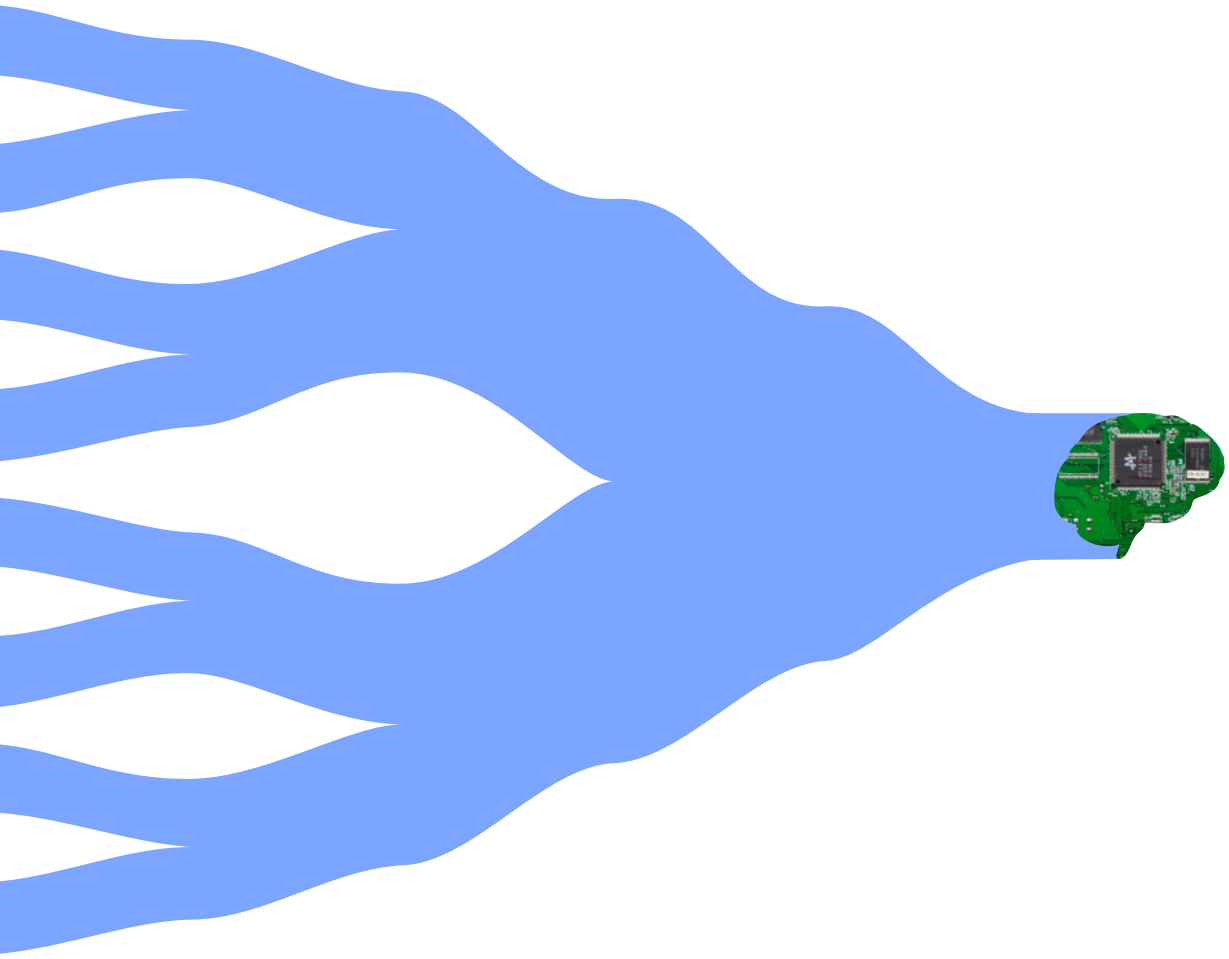
Building AI-safe Biotech Data Pipelines



Merelogic

Making Biotech Data Driven

ptp

Infinite Innovation

Contents

Brought to you by

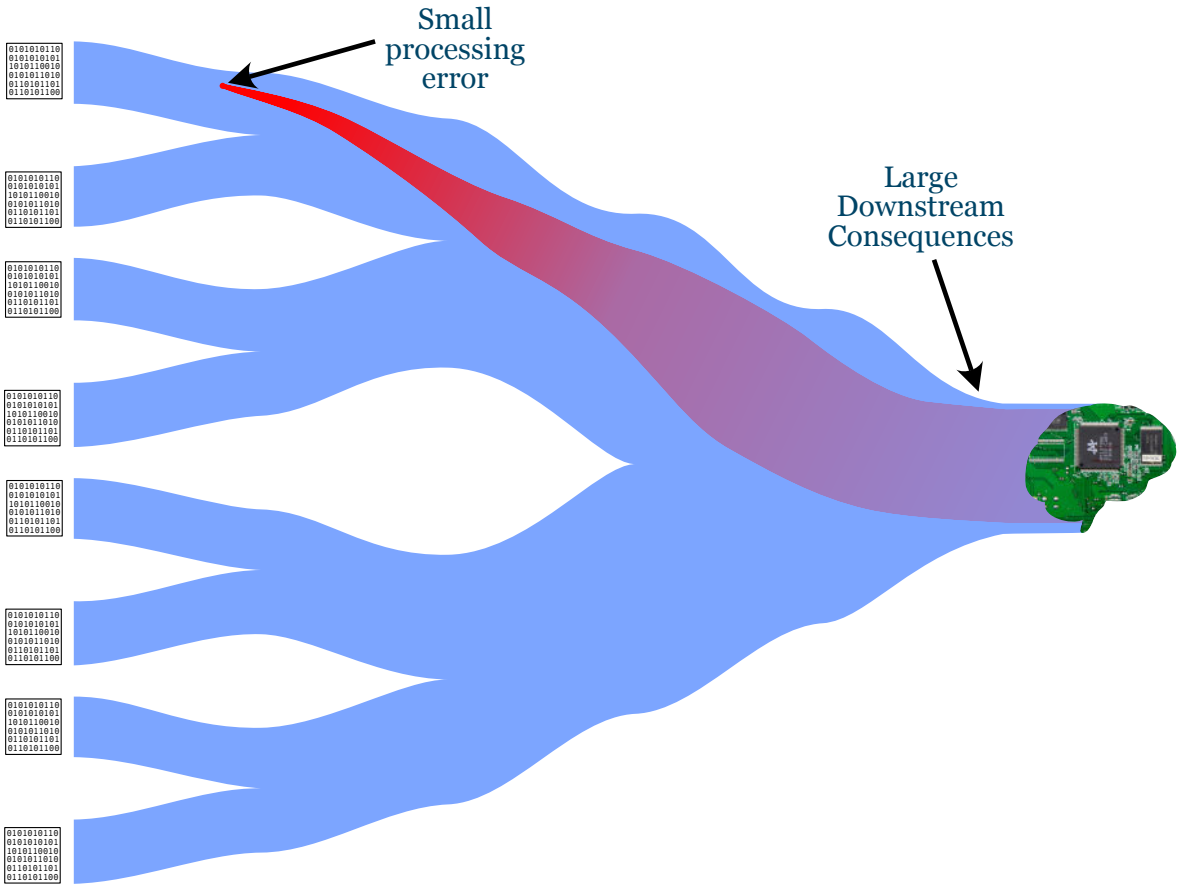
Merelogic
Making Biotech Data Driven
https://merelogic.net/

ptp Infinite Innovation
https://ptp.cloud/

New tools
New risks

Biotech and biopharma organizations are increasingly building AI models using diverse data sources including high content imaging, next-generation sequencing, spatial sequencing data and public ontologies. To do this effectively, this data needs to be AI-ready.

This paper introduces an important aspect of being AI-ready called "AI-safe" data. Biotech data pipelines that aren't AI-safe can introduce errors and bias into AI/ML models that are difficult to detect and expensive to fix, in terms of both time and budget. This paper explains what AI-safe means and introduces ten steps to ensure your data pipelines are AI-safe.



Why you need AI-safe data pipelines

One of the most common causes of delays and cost overruns when biotech/biopharma teams build complex AI/ML models is errors in the early stages of a data pipeline that propagate downstream in ways that are expensive, slow and difficult to fix.

For example, one of PTP's clients received a patient dataset in which males were indicated by a 2 instead of by the expected value of 0. One of the early processing stages interpreted anything other than 0 as female, meaning that every male entry in the batch was now corrupted. And any model trained on that data was now unusable.

Errors like this can come from many sources: using the raw version of a dataset instead of the processed one; using a new version of a software library that no longer processes certain subtypes the same way; the list goes on.

It's usually straightforward to correct such a mistake or remove the affected data at the point where it happened. But by the time the mistake is found, the data will have been processed, translated and aggregated into downstream forms where it has become intertwined with other batches and data sources.

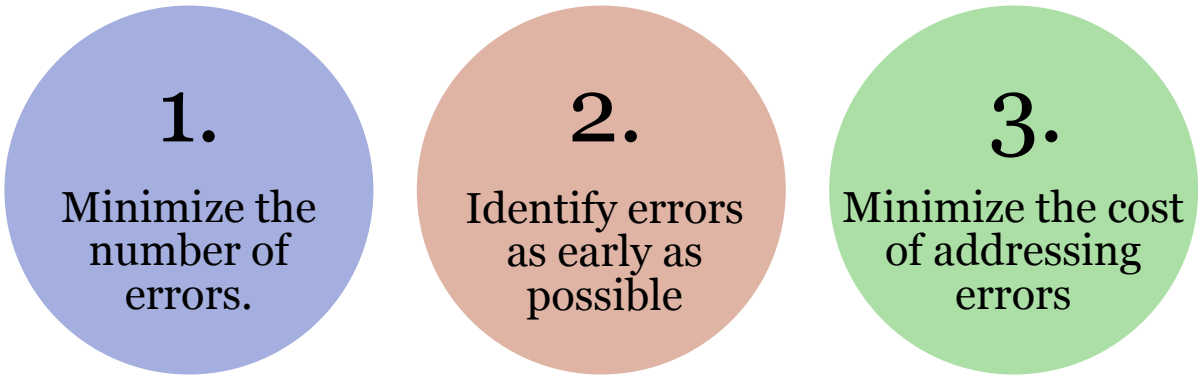
In the best case scenario, extracting only the affected data from these down-stream, aggregated datasets is complex and time consuming. In the worst case, it's impossible. So teams that encounter these errors must reprocess a much broader swath of impacted data, including the expensive task of retraining large models.

By the time a mistake is found, the data will have been processed, translated and aggregated

What is AI-safe data?

AI-safe data in biotech isn't about the data. It's about the systems that process every stage of the data from source to model.

An AI-safe data system does three things to minimize the risk of excessive costs from mitigating processing errors:



This white paper lays out concrete steps you can take to design internal systems that create AI-safe data.

Many of the steps outlined below are general-purpose best practices that apply to any data-focused application, from business intelligence to computational biology to AI. That's because AI is just an extension of data science, which is an extension of statistics, and so on. At a fundamental level, what makes data lead to good results hasn't changed that much between these disciplines. The saying "garbage in, garbage out" still applies and this guide will help you get rid of the garbage.

The Typical Biotech Data Pipeline

To better understand the context for the steps below, here's a quick overview of the steps involved in a typical data pipeline in early discovery biotech.

Each pipeline stage has unique risks and requirements

Lab Data Collection

For any models that use data collected from a lab, the data starts as readouts from instruments. Most instruments use proprietary formats and offer minimal options for transferring this data to centralize storage. So while some labs automate this transfer using commercial or internally-developed software, many labs rely on bench scientists to manually transfer the data.

The consistency of this process can vary widely, particularly in labs that rely on manual processes. Labs that work at a larger scale or in a GXP setting typically have detailed procedures and conventions in place. But smaller scale labs doing early discovery often don't follow consistent conventions.

Metadata Capture

The readout data generated by lab instruments is typically labeled in a way that can be linked back to the biological context. For NGS/sequencing data, this may be a UMI or barcode. For plate-based assays from simple plate readers to high content imaging, it's a plate barcode and well id. But these labels don't provide any context on its own. To interpret the data, analysts need to understand how samples were generated in the lab: cell lines, treatments, incubation times, reagents, stains, and so on.

For early discovery labs, the sample sheets and plate maps that record this context are often collected informally in spreadsheets and ELNs. In larger scale labs and GXP labs, this is often traced more formally in a LIMS. In either case, this information needs to be collected alongside the readout data so the two can be merged for analysis.

Analysis

Depending on the nature of the source data, analysis is often split into three categories:

- Primary analysis takes the immediate/raw data from lab instruments (primary data) and translates it into a form that can be processed by statistical tools (secondary data). The primary data is typically much larger in volume than secondary data. Some types of readouts won't require primary analysis.
- Secondary analysis involves basic calculations on the numbers that come out of primary analysis. These calculations are determined by the nature of the experiment, but are often relatively standardized.
- Tertiary analysis dives into the meaning of the data and explores any anomalies that are discovered. This often revolves around graphing and visualization, with the goal of telling a story about the data.

Interpretation

Analysis is all about extracting information from data. The interpretation step turns that information into knowledge by making discrete decisions/determinations about the data. For example, when screening drugs, calculating the z-scores for each drug candidate would fall under analysis. Deciding which candidates are hits, leads, etc. falls under interpretation because it's making a judgment.

Interpretation is important for models that rely on context and judgment such as knowledge graphs. Analysis is enough for models that require more objective information such as neural networks.

Aggregation

While a single dataset can be valuable on its own, the real value comes from combining multiple data sources to create more context and higher confidence. So once individual datasets have been analyzed and interpreted, the data is typically aggregated. This may involve merging multiple datasets of the same form, to provide the same kind of data about more entities. Or it may involve datasets that measure different aspects of the same entities.

In either case, this is where AI-safe data gets tricky: If any of the data that feeds into an aggregate dataset has errors, then the entire dataset may become corrupted.

Featurization

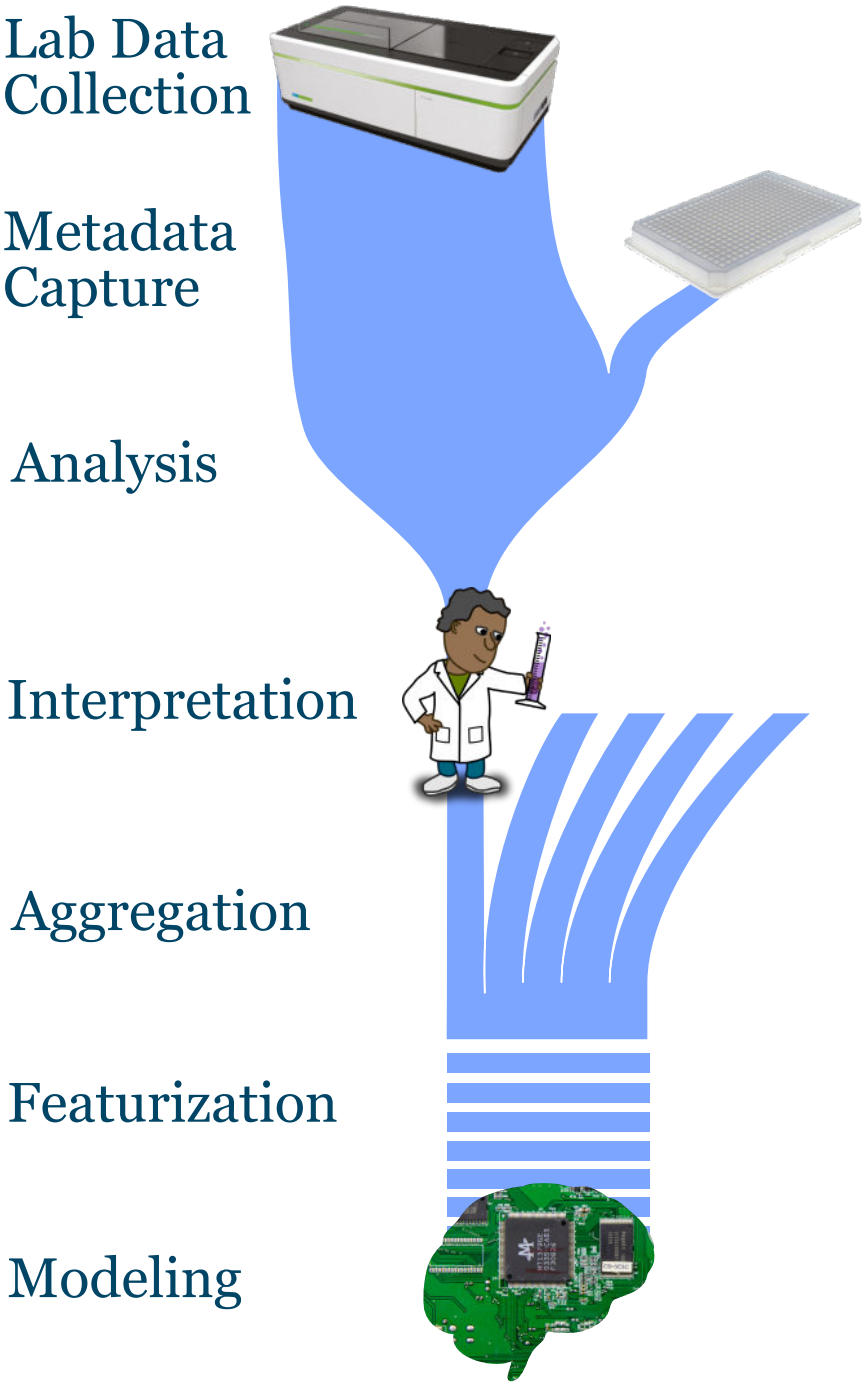
Most AI/ML models work on data that has been converted into vectors whose dimensions are called *features*. In some cases, these features represent concrete measurements such as a patient's height, weight and age. In other cases, the features are abstract, calculated from the data in a way that makes the models work but is difficult or impossible to interpret.

These abstract features are one of the fundamental concepts of generative AI. The problem they pose for AI-safe data is that they can be expensive to calculate, and almost impossible to debug. So if errors or bias are discovered in the input data, the embeddings may need to be completely re-calculated from scratch.

Modeling

The final stage is training the model itself. This doesn't produce directly usable data but it does produce a trained model, and the predictions from the model often become data for later processes. Because of the noise of predictions, it can be very difficult to identify errors and bias at this point, so it's vital to ensure that the earlier stages are as correct as possible.

Abstract features are one of the fundamental concepts of generative AI. They pose new problems for AI-safe data



The seven steps of a typical early discovery data pipeline

Minimize Errors

This section describes steps you can take to ensure that your biotech data processing pipelines minimize the number of errors.

Step 1: Minimize manual processing steps

Each step in processing biological readouts from raw data to trained model is typically carried out in one of three ways:

- Completely automated - Version controlled code is automatically run based on a pre-defined trigger, such as on a recurring schedule, or when new data is available.
- Manually run with code - A person manually runs code that loads, transforms and saves the data. The operator defines parameters including the source and output data locations, but every substantial step is done by code.
- Completely manual - A person manages the process through a mix of code based tools and manual processing/analysis in which one or more substantial steps are not captured in code.

Every manually run step is a problem for two reasons:

1. It introduces the potential for errors. Even something as simple as selecting a file can be done incorrectly if the operator misinterprets the naming conventions, is distracted or accidentally clicks in the wrong place.
2. It reduces or eliminates the audit trail. This is more relevant for identifying and minimizing errors, but it's still important here: Code can have bugs, but once you see an issue, you can find the bug in the code. You can't debug an undocumented manual process from a month ago.

Steps 2-5 below are focused on minimizing errors that mostly arise from manual steps. So the more you can automate processing steps, the less you have to worry about them.

Replacing completely manual steps and manually run code with completely automated steps can be an expensive investment. But in the long run it can save costs by reducing both ongoing manual work and the cost of mitigating manual errors. Trying to automate tasks too early can introduce its own set of problems, but as soon as a process is ready, it's worth the investment to ensure your data is AI-safe.

Step 2: Keep software dependencies lean

Many steps in the data pipeline will depend on specialized software or software libraries to execute. Using these libraries reduces the amount of custom code you have to write yourself. But new versions of these libraries may introduce changes in the underlying logic that have unexpected impacts on the data produced by your pipeline. When different people on the team use different versions of a library or when an automated pipeline switches to a new version, these changes can manifest in subtle ways that are hard to identify, hard to track down and hard to fix.

This is not to say you shouldn't use specialized libraries. But you should be careful about how you introduce them, and how you manage them. With automated pipelines it's fairly common to define formal compute environments, e.g. Docker images, with libraries pinned to specific versions. But there are ways to do the same thing for manual steps. The fewer libraries you can include in each compute environment, the easier it will be to keep them consistent. And the sooner you introduce these mechanisms, the easier the transition will be.

Be careful how you introduce and manage specialized software libraries.

Step 3: Keep file structures intuitive

The simplest error that you can introduce into a data pipeline is to start with the wrong data. Whether it's data from the wrong date, the wrong stage of processing, the wrong data source, or something else, if an operator picks the wrong file, anything after that will be wrong. So when a person is manually processing data, they need to worry about selecting the right input files and saving the output to a place where it can be correctly identified later.

The most impactful thing you can do to help ensure users pick the right file paths is to keep the overall directory structure as intuitive as possible:

- Name folders based on how users think about the data, not how IT thinks about it. In particular, use the terminology that users use in their daily work.
- Keep directory structures shallow, only adding nested layers when it makes the overall structure more intuitive.
- Whenever possible, give users alternate ways to find/select files without navigating a directory structure. This is only an option for certain tools and workflows.

More intuitive folder structures aren't always optimal from a technical perspective. But technical tools and scripts can usually be configured to work with any folder structure. Manual processes with unintuitive folder structures will just breed errors.

Keep the directory structures simple to help users pick the right file paths.

Step 4: Document what and where your data is

Even with the most intuitive path conventions, new users will always have questions or want to verify their understanding. Plus, technical team members who aren't domain experts or don't understand the terminology will need to understand what's going on to debug issues and support downstream processes. Therefore, it's important to keep detailed and up-to-date documentation on how and where all your data is stored:

- Organizational conventions for how directories are organized and the criteria for which files go in which folders
- Naming conventions for both folders and individual files
- Formats and schemas (column names and types) for each type of file
- Quality expectations for each type of file/data

This last bullet, the quality expectations, is a whole topic in and of itself and will be important in Step 7. The more detailed you can be here, the better. But any information you can collect will be helpful.

Finally, with any kind of documentation like this, the hard parts are keeping it up to date and making sure that new team members review it and remember where to find it.

Step 5: Enforce a least privileged access model

The more people have the ability to add or modify data, the greater the risk of accidental changes and the harder it will be to identify and debug the changes later. So only give the smallest possible set of users permissions to write or update each type of data. This approach, called the least privileged access model, is a standard concept from data management whether you're working with AI/ML models or otherwise.

Clearly define conventions and expectations from the beginning

Identify Errors Early

You can't prevent every error before it occurs. But if you can identify issues before they propagate, you can minimize the cost of regenerating corrupted datasets. So the steps in this section are focused on identifying problems as early as possible.

We expect some variability and errors in these models, so it can take weeks or months to recognize patterns that suggest deeper data problems. One unexpected answer from a generative model can be dismissed as a quirk. Two is frustrating but nothing out of the ordinary. It's only when answers are consistently unexpected or wrong that most teams start looking for underlying issues. And by then, the bad data has propagated throughout the system.

So it's important to deliberately and systematically monitor data for issues.

Step 6: Do statistics before AI

AI/ML models are typically good at smoothing out noise from individual errors and outliers. So the kinds of errors that most often trip up these models are large statistical shifts. A single male misclassified as female will get treated as an outlier. But a population going from 50% female to 100% because every male is misclassified is going to cause problems. A single mis-reported value will get averaged away. But accidentally using the numbers from two years ago because someone selected the wrong file is going to break long-term trends.

The easiest way to identify large statistical anomalies is with basic statistics:

- Calculate averages and variances for individual variables.
- Compare histograms across batches.
- Ask simple questions of the data and sanity check the answers.

In addition to helping identify data quality issues, exploratory statistics can also help determine what kinds of models you can build and how best to build them. So this should start before you begin designing models.

But the biggest benefit of doing statistics first is that you may learn you don't need AI after all. In multiple cases, clients who hired PTP to prepare their data for AI discovered that they didn't have enough data to train the kind of model they wanted. But they could get 90% of what they wanted with simple statistical models that would be less error prone and easier to maintain.

Step 7: Validate the data at every step

Each stage in a data pipeline has the potential to introduce errors or reveal errors that were introduced upstream. Therefore, it isn't enough to run quality checks at the beginning or at the end. The more stages at which you calculate, evaluate and report statistical checks, the sooner you'll identify errors and the easier it will be to track down.

This becomes harder as you move downstream, particularly to trained models where it can be hard to define the expected behavior and define what kind of variance should be considered out of range. But every time you identify and fix an error, you'll get a better understanding. Each time that happens, add a check to prevent it from happening again.

Tracking down errors becomes progressively harder at every step.

Minimize the Cost to Fix

Because errors propagate as they move downstream through your data pipeline, a small issue can often require a very large fix. So the steps in this section are designed to help you minimize the area that you need to fix when you identify a problem.

Step 8: Track every step

At various stages in a pipeline, data from different sources and batches get mixed together. If one of the inputs to that stage had an error then everything that was output from that stage will potentially have errors, as will everything downstream from it. If the inputs to a processing stage had errors, you will almost always need to rerun it. In fact, if you're not sure if the inputs to a processing stage had errors, you'll probably need to rerun it.

So the best way to minimize the damage of data errors, particularly in the early stages of a pipeline, is to be able to identify exactly where the error was introduced and which intermediate steps were affected. Every dataset you can rule out is one less processing step you'll need to rerun. And some of these processing steps can be quite time consuming and expensive, particularly if they involve training an AI model.

So for each step in the pipeline, you'll need to track:

- Data inputs and outputs
- Code and software used, including software/library versions
- Configuration parameters
- Quality control statistics

This information should be organized in a central place where it can quickly be accessed and evaluated. The more quickly you can narrow down what data was affected, the quicker and easier it will be to clean up the problem.

Step 9: Make data flow only one way

For the operations of an organization, it's often necessary for the same information to be stored in multiple places. Different teams use different software and systems, so information at the interfaces between these teams needs to be available in all these systems. But when the data in different systems gets out of sync, the problem is to sort out what's right and what's an error.

This problem is particularly difficult when information flows both ways between systems: When one user updates information in System A, an automated process sends the update to System B. When a user updates the same information in System B, another process sends it back to A. This is all well and good until one of those processes fails. Figuring out which one failed, and what the information should be is easier said than done.

Because of this, many organizations avoid storing different versions of data in different systems. However, this restriction makes everything else more difficult. For example, AI/ML models need to access batches of data via flat files while most other applications need a relational database that would be inefficient for AI/ML.

So instead of picking a single storage solution that only works for half the use cases, a better solution is to define a single source of truth: Users are able to change the information in System A and an automated process will then update System B. But users can't update System B directly and there's no flow of information from B back to A. System A is the source of truth.

By restricting data flow to a single direction and defining which system is always considered correct, it's easier to understand where errors were introduced and where they have propagated to.

Multiple copies of data can be safe, provided a clear source of truth.

Step 10: Make data read only

Data in operational systems needs to evolve as users add and update information. However, AI/ML models are trained on static datasets. So the data pipelines that feed these models are typically built around snapshots of the operational data.

While these snapshots could be updated at intervals to sync them with the operational systems, this only makes tracking and debugging issues more difficult: To track down an issue, you need to know exactly what data was used at each step of a pipeline.

The solution is to store the snapshots that feed into your data models as read-only, version-controlled records. Each time you take a new snapshot, create a new file and/or folder with a timestamp and version number. Record these when you track the data pipeline, following Step 8.

Being able to reproduce the exact inputs for each stage in your pipeline will make it significantly easier to track down issues and determine exactly what needs to be reprocessed.

When data changes, create a new dataset.

Conclusion

The 10 steps described above to create AI-safe data pipelines for early discovery biotech are relatively straightforward, and align with more general best practices. By following these recommendations, you'll minimize both the risk of errors and bias in your AI/ML models and the cost of mitigating any errors that slip through the cracks.