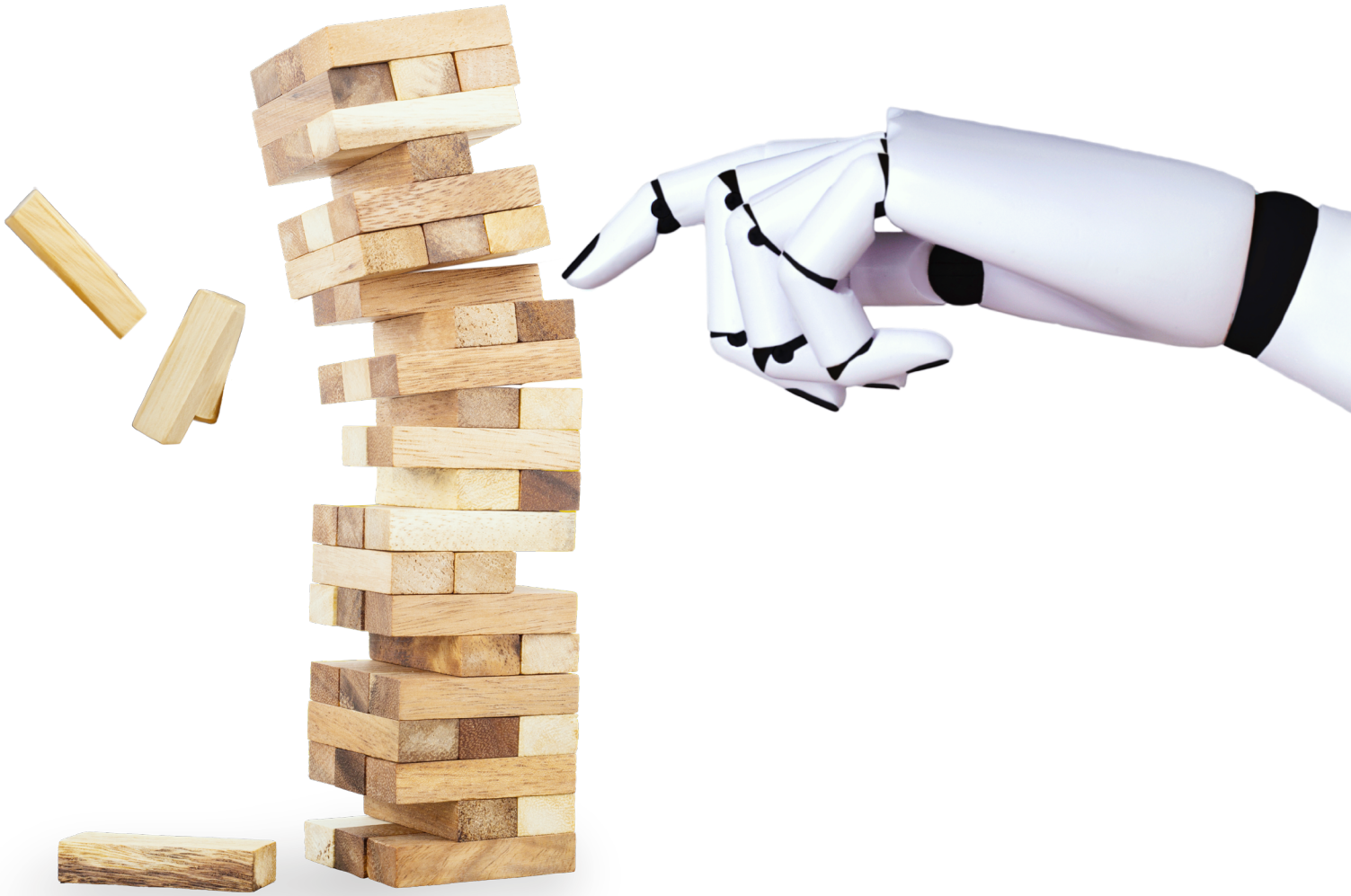
Open for Innovation
KNIME

6 fault lines of deploying analytical models

& how to address them with ModelOps



In 2020, a televised Scottish football game between Inverness and Ayr United had fans, worldwide, cursing at their television sets. Except it wasn't because of a tight score, a poor referee call or foul play. It was because of a malfunctioning machine learning model. The cameras, trained to identify and stay focused on a moving soccer ball, were instead **focusing on the bald head of a player** – missing the action of the game entirely.

So what went wrong? It could be that the model wasn't trained on enough data–there weren't enough images of soccer balls and humans. Perhaps it wasn't trained on the right data–there weren't enough bald heads in the data set. Perhaps the model worked perfectly on the test data set, but failed to maintain accuracy in production. Perhaps there was model drift–when the model was trained, it was less fashionable to shave your head, and thus there was less likelihood of a bald head.

Getting a machine learning model to work in the wild–i.e., regularly provide predictions on new data–is not easy. The process of taking a model and making it available for software or people to regularly access is known as the "deployment" or the "productionization" of models. And it's not made easier with a large AI initiative budget or the number of data science PhDs in your organization. The practical application of analytics doesn't stop at the "science" of developing a model.

Productionization requires many steps under the hood. In many companies that are deploying a handful of models, only a few data and IT people are concerned about this process on a case by case basis. After developing their model that works on historical data, the team needs to (1) test the performance of their model on test data, (2) validate that the model complies with internal standards and regulations, and, ultimately (3) rework and prepare the model such that it can easily be invoked by a user or system. And even once the model is in a production environment, the work is still not over. Data and IT teams need to keep an eye (or "monitor") that the model continues to perform, and that it doesn't, for instance, confuse a bald head with a soccer ball.

Deploying a model is a complex, cross-disciplinary project involving data and IT experts that takes weeks if not months or quarters to pull off. Making errors in this process, however, can have severe consequences, costing a company its reputation, its bottom line or even posing an existential threat.

Over time, however, line of business teams and stakeholders get tired of waiting weeks for each solution. Despite the incredible complexity of building and deploying models, data teams get pressure to increase the time-to-value of their models, and solve more business problems in the organization.

It's at this point that data teams need to start thinking about ModelOps–or systemizing and automating the process of productionizing models.

Identifying fault lines as the first step towards ModelOps

Starting to think about building a ModelOps process can feel overwhelming, but it's really just taking small steps to start minimizing the opportunity for error and reducing decisions made on an ad hoc basis. You don't need to replace every manual process tomorrow, but you can start identifying where common issues might arise and putting in place standards and important safeguards.

At first, taking these precautions will improve the efficiency–or the time-to-value–of the models produced by data experts. If your team is already taking a **low-code approach to model building**, then in the long term, it'll also decrease the amount of software infrastructure and scripting know-how necessary to deploy a model in the first place, opening up data science to more people in the organization. Carefully managed, this means more data science bandwidth, encouraging more data-driven decisioning. At an organizational level, this takes the pressure off data experts, freeing them from solving basic automation and prediction problems, and allowing them to spend more time researching and adopting new, innovative technologies.

In this e-Guide, we identify 6 common fault lines that can be addressed by starting to build a ModelOps process:

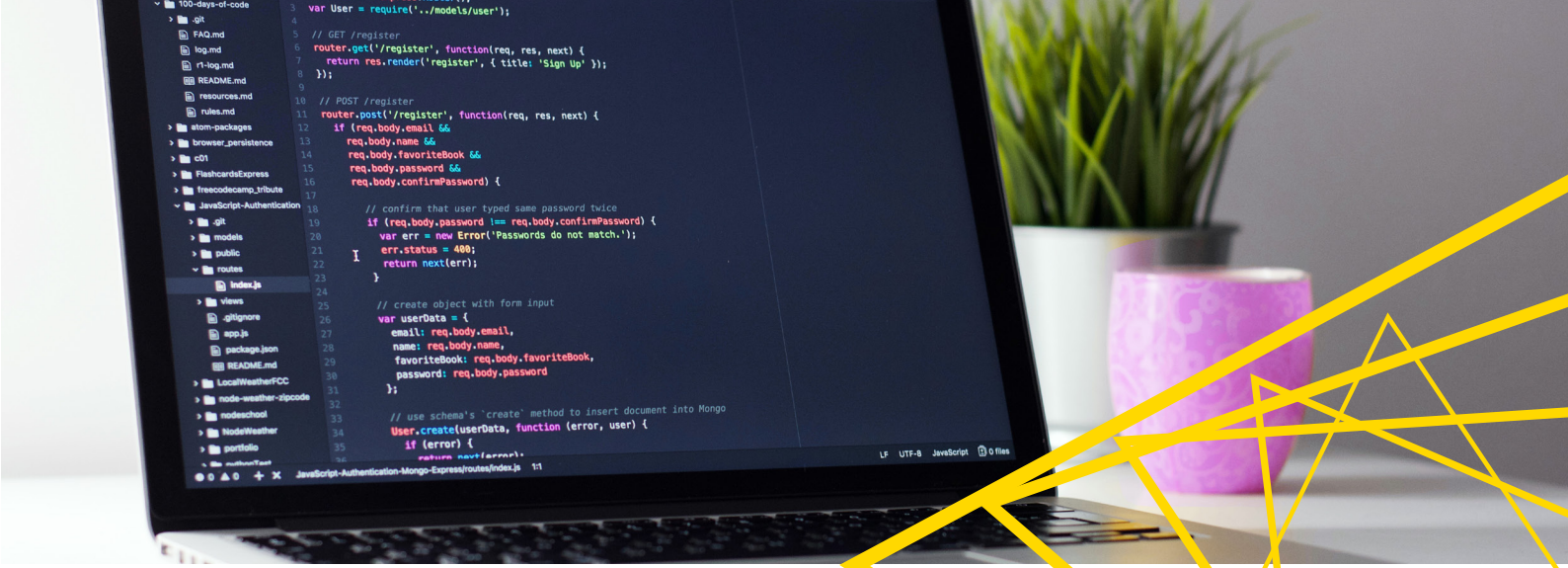
1. **Poor model choice and parameterization**
2. **Inconsistency between "train" and "predict" models**
3. **Security holes**
4. **Undetected model drift**
5. **No record of changes to models**
6. **Over-reliance on software infrastructure know-how**

1. Poor model choice and parameterization

One of the first and most obvious things that can go wrong is that after training, the quality of the model is not accurately assessed. For example, someone inexperienced with machine learning models might train and validate their model on the same data set. As a result, in the validation step, the model isn't being tested for correctly generalizing for "new" data.

There are many reasons for a model underperforming, like poor outlier handling, over and undersampling, overcorrecting for bias, etc. Regardless, the model developer would want to know that their model is overfitting their testing dataset as early as possible, to avoid doing too much work to prepare it for deployment.

To catch this early, you can set an automated process to ensure every model developer is checking for overfitting. Ensure that each model has a training data set, a testing data set, and a validation data set – and, if necessary– put in place requirements for more splits for the cross validation of ML models. You can further minimize a team's errors by setting up a central repository with re-usable cross-validation methods that model developers can adapt as needed.



2. Inconsistency between trained and production models

For many teams, the process of productionization often involves reworking, or even entirely re-coding their model for production. After creating their models and realizing their development environment isn't designed for running models in production, data scientists often turn to their software engineering counterparts to re-create the model so that it can be easily invoked as a service by another software.

The result is then having to worry about two models: one that was used for training, and one that's used in production. It's not uncommon for small discrepancies to arise when reworking the model, especially when it was reworked by someone other than the person who built it. The result is having to maintain both packages and knowing what and how to make distinct changes to the production package, as the need arises.

One aspect that's often overlooked is that the prediction package needs to comprise not just the model itself, but also all the pre- and post-processing steps required to get from raw data to meaningful prediction. One solution is to build out a mechanism to derive a prediction package from the training model. Select low-code, end-to-end platforms for data science include this functionality (known as "**integrated deployment**") out-of-the-box, automating deployment and ensuring that models can never get out of sync.

3. Security holes

Another common risk is inadvertently opening up access to sensitive information, like personal data that isn't meant for a wider audience. This can be done inside an organization, inadvertently, giving the wrong users access to data or data products. Or, worse, data can be opened up to 3rd parties outside the organization.

Many data teams consistently experiment with new techniques and test out new libraries to leverage the most state-of-the-art developments. However, installing packages provided by 3rd party contributors are often not checked for license or security holes. Malicious players can use packages to inject command executions, retrieve sensitive information under the guise of "debugging," or just install malicious packages.

Setting up tests to ensure models are built on current (rather than outdated) packages, as well as putting in place checks for common vulnerabilities and exposures, are easy preventative steps you can add into your productionization process. You can also choose to separate your dev, test and prod environments entirely, to make sure that any malicious activity is contained.

You can also tighten security around the accessing of databases, defining when (if ever) it makes sense for authentication to be passed through a script. Ensure that individual credentials are not shared with anyone that they shouldn't be. Here, again, you can share best practices in a library for how to give access to specific datasets.

4. Undetected model drift

Machine learning models are built on a number of assumptions and are trained on a historical data set. If something about those assumptions changes, the model's predictions become inaccurate–a phenomenon known as model drift. For instance, a model might be trained to detect fraud based on known suspicious activity. If fraudsters come up with new phishing tactics, however, the model will fail to predict them. For this reason, fraud detection models need to constantly be retrained.

The data expert who built the model typically has all the business and technical context to determine whether the model has drifted. Yet, in many cases, the data expert doesn't have ongoing visibility into the performance of the model in production.

Here, a couple of actions can be taken to improve this process. First, the process can require that each model is deployed with a mechanism that provides an accepted accuracy threshold. When the model doesn't perform, then an alert-based system can be instated to keep the data expert in the loop. Second, a dedicated environment for retraining of the model can be used to quickly adjust and re-deploy the solution.

5. No record of changes to models

When models are deployed ad-hoc by various team members, it's easy to sneak in a change to the model into production. Since many data teams work with typical software development protocols, they have mechanisms in place to version the code that created the model. However, they lack the mechanisms to store and roll back to previous model versions that had resulted from that code. Software development allows for code version control, but doesn't have the same set up for storing models.

However, as the number of people who deploy data solutions grows, it's critical that a version control process that tracks changes and enables you to roll back to previous versions of models is installed.

6. Over-reliance on software infrastructure know-how

When a data team is first formed, it's often expected that data scientists will own not only model development, but also model productionization. Rare "unicorn" data scientists are expected to not only be experts in statistics and machine learning, but also learn about software infrastructure and become familiar with the complexities of making a model available to systems and people across the organization. Not only is this a misused resource, squandering the time of fairly limited expertise–but it puts a lot of pressure on a single individual to not make a single mistake in this process. It can also result in costly and fragile setups.

Set up a process that makes productionization easy for data scientists, while reducing the burden on IT. Here, an end-to-end software for building and deploying data science solutions can make a big difference.

Preparing for scale

Despite what the term “ModelOps” might suggest, the process for productionization doesn’t just apply to AI or ML models. It’s simply a way to control and safeguard the use of data–whether it’s simply cleaning and assembling datasets, building dashboards, or leveraging machine learning techniques to make predictions.

While many teams try to set up a software development practice just to productionize their models, others instead opt for a low-code/no-code approach to building and deploying their analytical models.

An end-to-end platform like **KNIME** provides users with a single environment to first build, then automatically deploy their analytical models as data apps or REST APIs made available to the wider organization. KNIME’s CDDS extension lets data scientists set up dev, test, prod (or any number) of environments, create manual or automatic testing and validation requirements–and, ultimately, monitor and retrain their models. The extension is completely customizable to fit with any organization’s unique governance and deployment practices.

This approach eliminates the software development work that a data expert has to do, and allows for IT to set controls without burdening them to help with re-building and monitoring the model. At the same time, this process, if set up right, empowers the data and IT team to govern and scale deployment to more people, while incorporating other expertise like compliance and legal.

In the end, your organization has many more data problems than it has data scientists. Inevitably, data and IT teams need to prepare for a future in which every desk worker can interact with data and create insights for their wider department or organization.

The question is do you prepare now – or do you wait until your fault lines cause damage?

Learn more about **ModelOps** with **KNIME**

knime.com/continuous-deployment-of-data-science