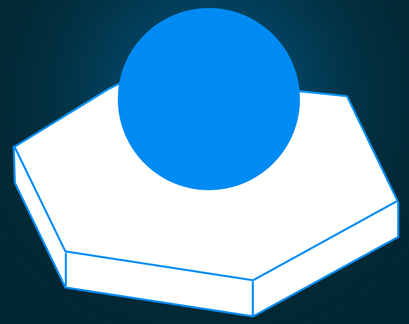
superwise.ai

# Fundamentals for efficient ML monitoring

A framework to scale AI with a production first approach



# Introduction

Machine learning models have become an integral part of business strategy, enabling incredibly diverse organizations to differentiate and excel. Data science teams strive to build models that benefit their organization, yet as models move from research to the real world; they end up going off the rails once in production.

As machine learning is drastically different from other software or traditional IT, models risk degrading the moment they are pushed into production where the hyperdynamic nature of the data meets the hypersensitivity of the models. These “drifts” in the data structure, or other properties that cause model degradation, are too often silent and unobservable.

From Instagram to Amazon, headlines keep providing us examples of the business impact that result from models degradation. In the last few months, triggered by the COVID-19 crisis, we have all witnessed companies struggling to fix corrupted and business-critical models. One of the most documented of such issues was Instacart, whose inventory prediction model’s accuracy declined from **93% to 61%**, leaving a sour after-taste for their customers and their teams.

Rare are the data science and engineering teams who are prepared for this “Day 2” , the day their models meet the real world, as they invest the majority of their time researching, training, and evaluating models. While it’s clear that teams want to address any potential issues before they arise, there is a lack of clear processes, tools and requirements for production systems. Today, the industry still lacks guidelines of what an optimal ML infrastructure should look like.

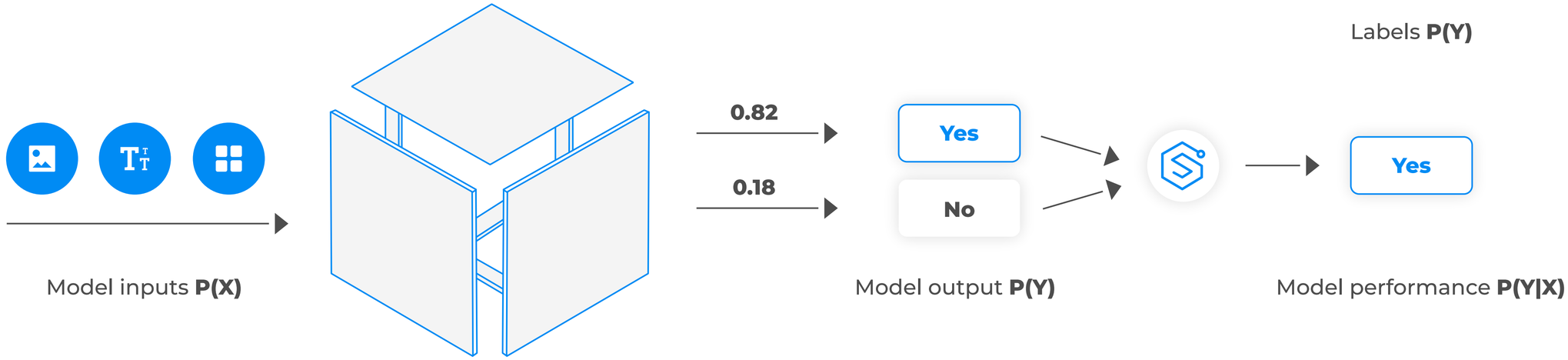
Without this framework, teams wind up building models that do not have the business impact they were designed for. This leaves already overburdened data science teams to spend their time troubleshooting and performing endless maintenance tasks, without a clear and timely view of the best corrective action. To achieve this one should invest in a robust infrastructure that includes a strong monitoring component that will enable them to gain insights, and control over the health of their models in production, and empower their data science and operational teams to scale their use of AI.

Leveraging years of experience in ML production, we have gathered in this ebook some of the most common best practices for monitoring your models in production and for the continuous optimization of your models. Building on our expertise and scars from the field, we hope to provide a framework for anyone who has an interest in building, testing, and implementing a robust monitoring strategy in their organization or elsewhere.

# Fundamentals

1. Look beyond performance measurements
2. Use different metrics for different features
3. Use a granular point of view
4. Avoid overflow and detect events automatically
5. Comparing different versions in parallel
6. Monitor protected features as proxies to ensure fairness
7. Remain platform agnostic
8. Empower the different stakeholders
9. Use your production insights for other stages of your ML process

# 1 Look beyond performance measurements




The intuitive approach to monitoring ML models, is to look at the performance of the model over time in order to detect degradations. However, and given that the performance stage is the “last” phase of the ML process, relying on it to discover quality changes can be very challenging, as once the feedback is received, it is often already too late!

While in real-time bidding, one can get prompt and clear feedback (i.e.: the user clicks the ad), in most cases, labels are obtained in delay, or not obtained at all. Take the use case of a loan approval model. For the approved loans, the real labels (whether the loan was paid back or not) will only be available after several months, whereas for the declined loans, the labels won’t be back at all, making it nearly impossible to evaluate if the model was right or wrong.

> To detect different possible issues and to do so on time, we recommend measuring the stability of the entire ML flow: including the input of the model, the output or the model’s decisions, in addition to measuring the general performance rate of the system.

# 1 Look beyond performance measurements

The list below gives examples of the different types of drifts: concept, input, decision and labels.

**Concept Drift** P(Y\X) - The impact of features on the outcome may change. In a loan approval example, even if there is no input drift, a crisis can impact the probability for a borrower to pay back a loan. For example, a borrower's income level, which was enough to ensure a loan return before COVID-19, may not be enough during or after COVID-19. The financial market has changed and there is a great deal of uncertainty for the stability of that income in certain markets.

**Input Drift** P(X) - The statistical properties of the predictors (i.e. features) change. This may happen due to technical reasons, such as a change of data sources pipeline, sensors that become noisy or inaccurate over time, etc..., or changes in the real world, like users' preferences.

For example, and continuing with our loan approval use case, let's assume that the marketing team launched a campaign targeting young families. If the latter were not a subpopulation represented while the model was trained, one can only expect that the performance will not be optimal for this sub-population as it is a whole new input that the model was not trained on.

# 1 Look beyond performance measurements

**Decision Drift** P(Y^\X) - The distribution of the model decision is changed. This can occur due to small or technical changes in the input which might not be significant enough to be detected as input drift, but still significantly change the model outputs. For example, in the loan approval use-case, and regardless of the existence of financial crisis, if the model starts to approve more loans, even if the performance doesn't change and the loans are legit, the fact that more loans are approved increases the risk the company is exposed to, which should be brought to attention

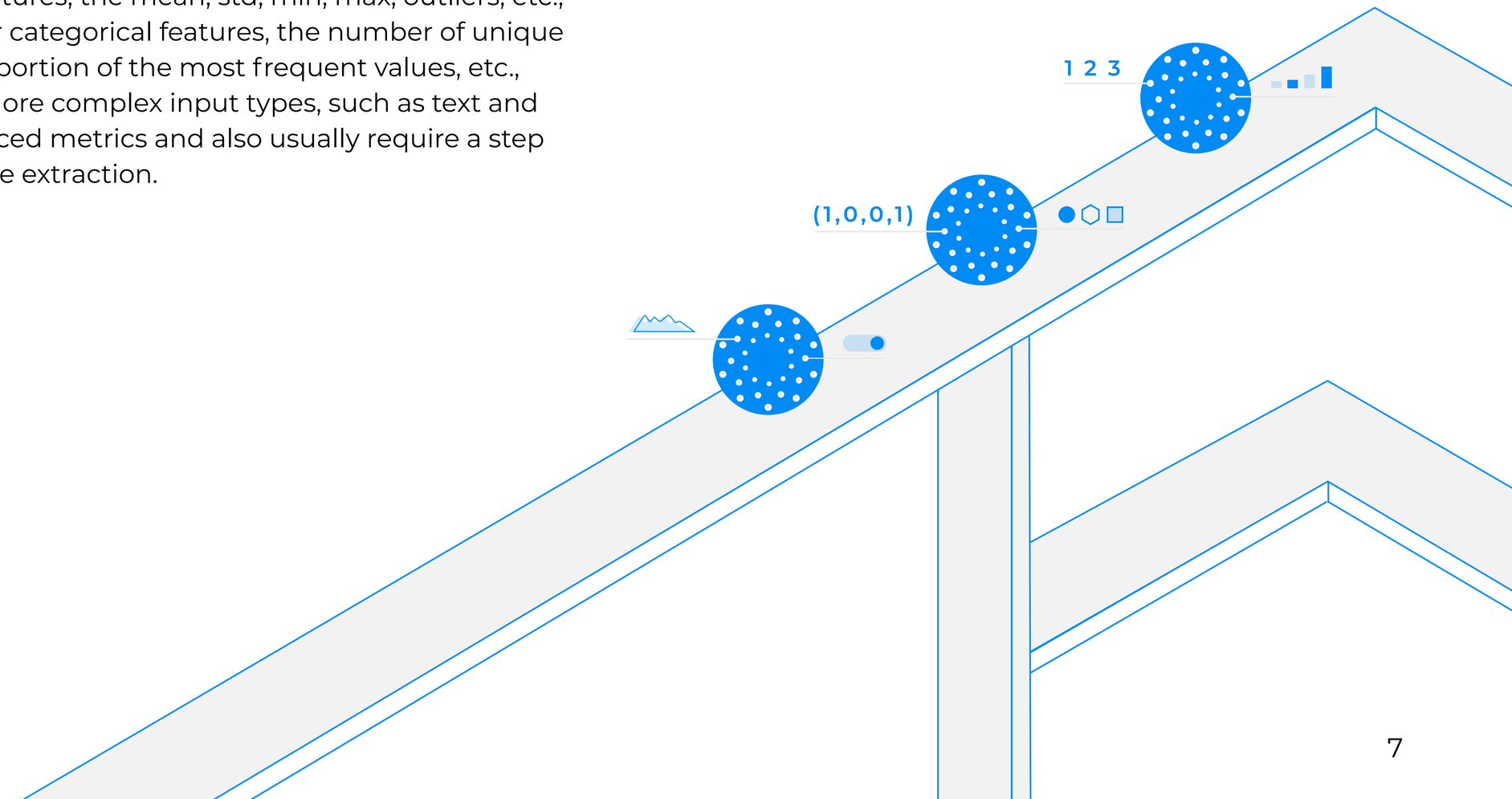
**Label Drift** P(Y) - The distribution of the real labels, the outcome we are trying to predict, is changed. For example, the increase of online shopping due to COVID-19, may attract an increase in fraudulent transactions. Detecting drifts in the labels can indicate potential model optimization, even if there is no performance decrease.

Overall, by monitoring the input and inferences as proxies for changes that may impact the model's performance, drifts can be detected early in their formation, as these are available in real time. Additionally, monitoring the input helps shed light on what has been changed, which is the starting point to diagnosing a performance issue and remediating it, before serious damage is done.

# 2 Use different metrics for different features

To detect drifts, the first step is measuring the statistical properties of the data. Yet, different data formats require different measures, and each variable should be measured using several metrics for different statistical properties.

For instance, for numeric features, the mean, std, min, max, outliers, etc., must be monitored while for categorical features, the number of unique values, new values, entropy, portion of the most frequent values, etc., are what matter the most. More complex input types, such as text and images, require more advanced metrics and also usually require a step of pre-processing and feature extraction.



# 2 Use different metrics for different features

Let’s assume we’re analyzing the sentiment of Twitter “tweets” which are limited in characters and contain different types of media - text, images, emojis, etc....If suddenly the model receives data with longer texts or new emojis, it will be less effective. Or for an object recognition use case that leverages images, properties such as the incoming image resolution, sharpness, contrast, etc.,  should be considered to detect issues when the model operates “in the field”.

Performance is measured differently for different ML tasks (e.g. regression, classification, or recommendation). But different use cases of the same type require different metrics and even the same use case might be measured by more than one KPI. For example, one classification problem is measured by accuracy, while a second one, like an imbalance use case, is measured both by precision and recall.

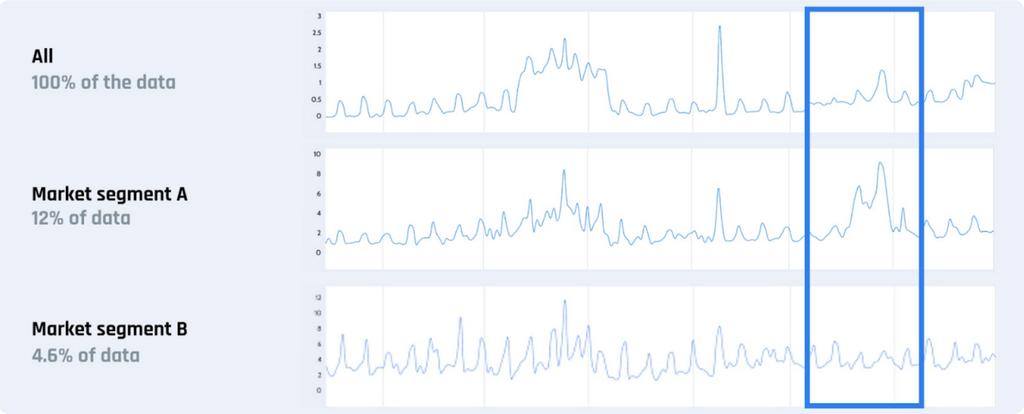
> When building a monitoring service that aims to serve more than a single model, organizations should take into account different types of data and use multiple performance metrics.

# 3 Use a granular point of view

As described in point 1, the changes that impact data can be very subtle, and thus, they may be hard to detect. In many cases, drift occurs only for a specific subset of the population and could easily go "under the radar".

In the figure below, each row shows a measure of the stability of the input over time relative to its training set.

Wherever the line is higher, the data differs from the training set.




The top strip is the level of distribution change, relative to the training data for the entire population. One can see that there is clear seasonality, and that in the past, two major global events occured that impacted all segments. In the period framed in blue though, there was only a slight overall impact. When looking at a segment level, one sees that while a few segments remained stable, segment A experienced a huge drift, resulting in suboptimal results.

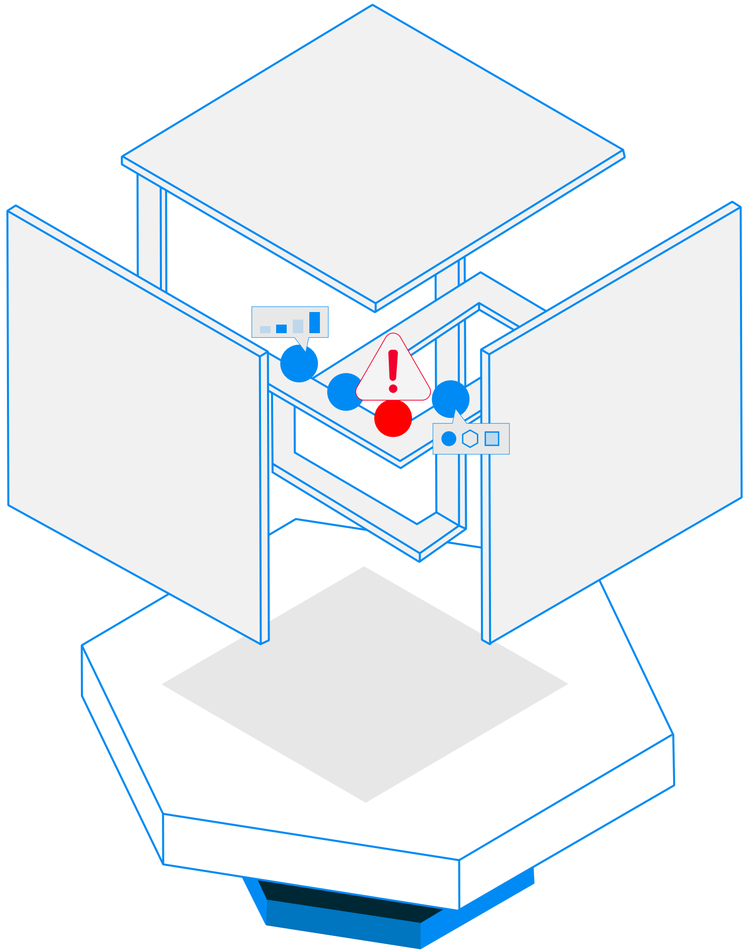
To be able to detect anomalies, one needs to monitor subsets of the data. Also, these sub-groups are often of paramount importance for your business/ operational users and may have a tremendous impact on your organization.

# 4 Avoid overflow and detect events automatically

As described in points 1 to 3, to detect different types and levels of drifts, each variable must be measured with multiple metrics. Given that a standard ML model usually uses dozens of features, it results in hundreds of measurements over time. With all these measurements, it's easy to get overwhelmed by information and sadly, become inefficient.

The requirements for ML monitoring systems must be different from traditional IT monitoring frameworks, as it is impossible to manually define the expected normal values or the rate of change that trigger alerts in ML monitoring.

With dozens of features, each measured with multiple metrics, that all have different natural distribution changes over time, and given that the natural distribution of each feature must be learned, including taking into account possible seasonality to perform adequate monitoring, going at it manually is a doomed mission. For the monitoring framework to be really useful, it should automatically detect different types of issues and alert the user(s)



# 4 Avoid overflow and detect events automatically

As described in points 1 to 3, to detect different types and levels of drifts, each variable must be measured with multiple metrics. Given that a standard ML model usually uses dozens of features, it results in hundreds of measurements over time. With all these measurements, it’s easy to get overwhelmed by information and sadly, become inefficient.

The requirements for ML monitoring systems must be different from traditional IT monitoring frameworks, as it is impossible to manually define the expected normal values or the rate of change that trigger alerts in ML monitoring. With dozens of features, each measured with multiple metrics, that all have different natural distribution changes over time, and given that the natural distribution of each feature must be learned, including  taking into account possible seasonality to perform adequate monitoring, going at it manually is a doomed mission. For the monitoring framework to be really useful, it should automatically detect different types of issues and alert the user(s).

To actually get alerted whenever the unexpected happens, different time series ML techniques should be applied. Metrics and KPIs should be scanned in a way that is smart enough to know when there's a real problem, regardless of whether it's a trend or To actually get alerted whenever the unexpected happens, different time series ML techniques should be applied. Metrics and KPIs should be scanned in a way that is smart enough to know when there's a real problem, regardless of whether it's a trend or drift detector to detect slowly changing feature distributions, or a seasonality detector to detect unexpected behavior according to the season. With smart alerts, teams can extrapolate on the data and aggregate metrics together to understand if, when combined, metrics result in a high alert, or if it just represents a normal deviation.

# 4 Avoid overflow and detect events automatically

To detect the anomalies below, one requires different statistical measurements. One can see in the graph here how, by leveraging smart mechanisms, they can be caught automatically. The blue lines in the images show the changes over time.

**1. "A blip"** - A sudden anomaly that returned to normal after a single day, and that causes an unintentional bug in the ETL.

**2. A change detection** - Something in the outside world changed, causing a feature to change its normal behaviour. While data science teams want to know about it, this may be the new standard. Ideally, the system should be able to adapt itself to this new normal after some time.

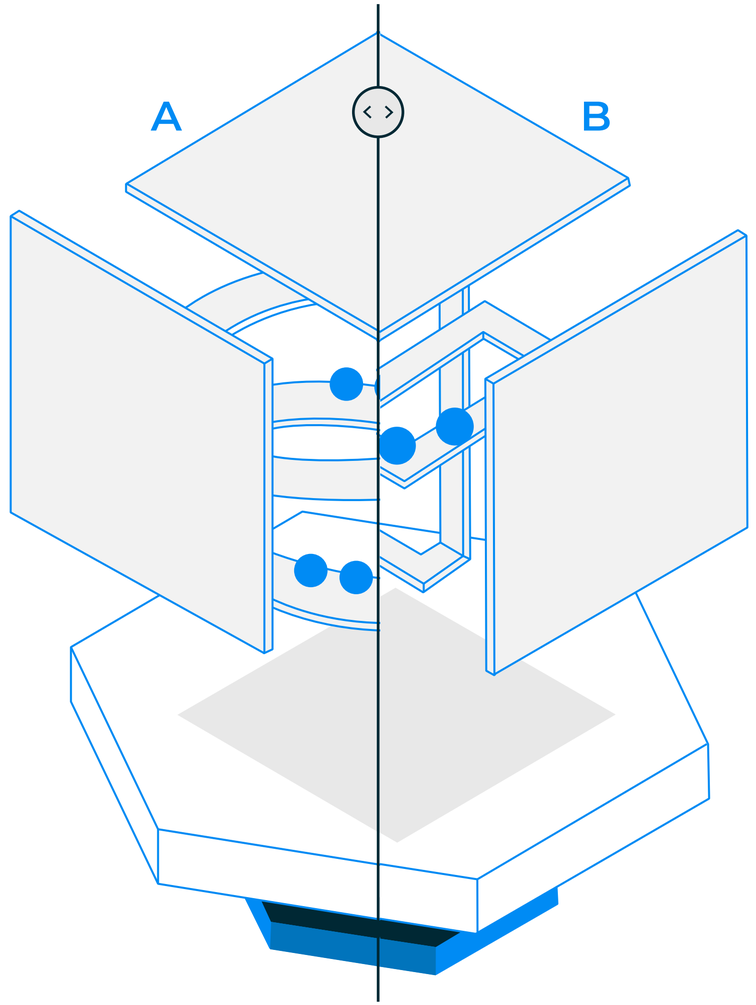
**3. Out of seasonality** - The level of change is within the normal boundaries of change but not in the right season.

# 5 Comparing different versions in parallel

Monitoring more than a single model which run in parallel to solve the same prediction problem may be necessary to validate new model versions. The most common practices involve: using a shadow model, using A/B testing, or Multi-Armed Bandit (MAB).

In cases where the model changes reality, usually when its goal is to affect user behavior, like in recommendation systems, it is impossible to evaluate the model offline. As such, online tests must be performed, in which a portion of the predictions are sent to the new model. In this scenario, two (or more) models must be tested on the same prediction problem, and organizations must be able to monitor and compare the two models: both their inference values and the performance.

Even when looking at models that don't change reality, and simply for the goal of ensuring the safe roll-out of a new model or versions, it is best practice to deploy the new model as a 'shadow model, before using it in production. In this scenario, all predictions should be sent to both models: the real production model and the shadow one. While the system uses the production model, the monitoring service should persist both predictions, including the model version that each prediction belongs to.

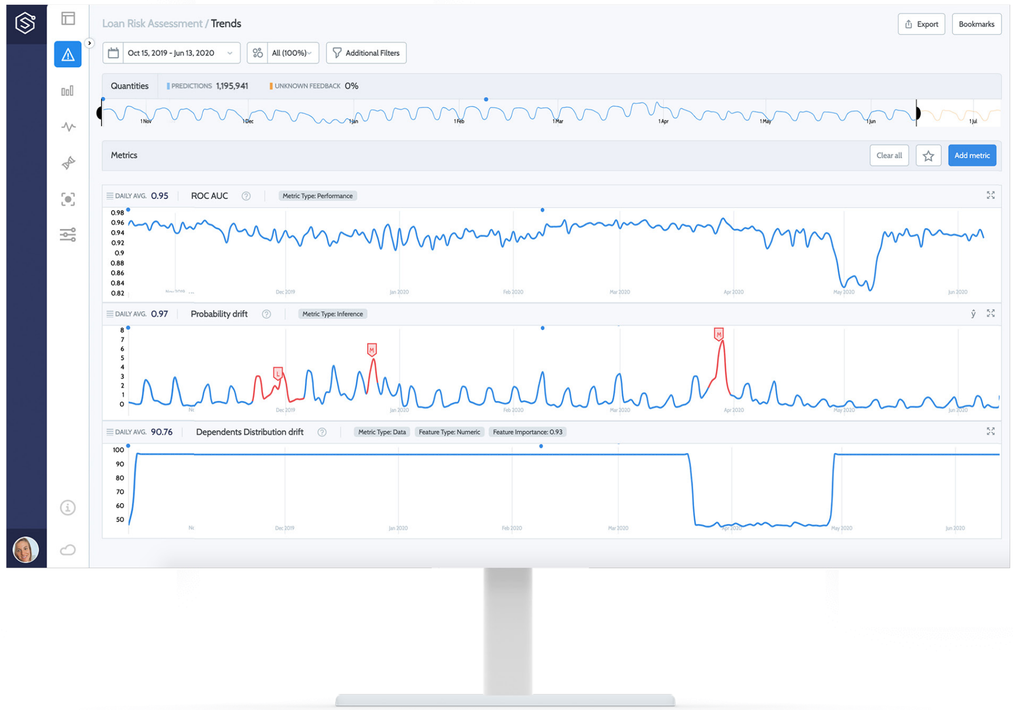


# 5 Comparing different versions in parallel

This way, one can test how the new model actually performs on live production data. Additionally, since the same cases on both models are used, it's easy and useful to see where both models agree and disagree which is a good indicator for some unexpected issues. For example, when the new model contains only minor incremental improvements, perhaps just retraining with more recent data, and there is a high level of disagreement between the models.

On the other hand, using a new model with major changes, a high level of disagreement can be expected, but the cases of disagreement can be compared and validated to ensure that the new model performs better than the old one.

Below you can see the safe delivery of a new version (V2), yielding an improved ROC AUC performance in production.



# 5 Comparing different versions in parallel

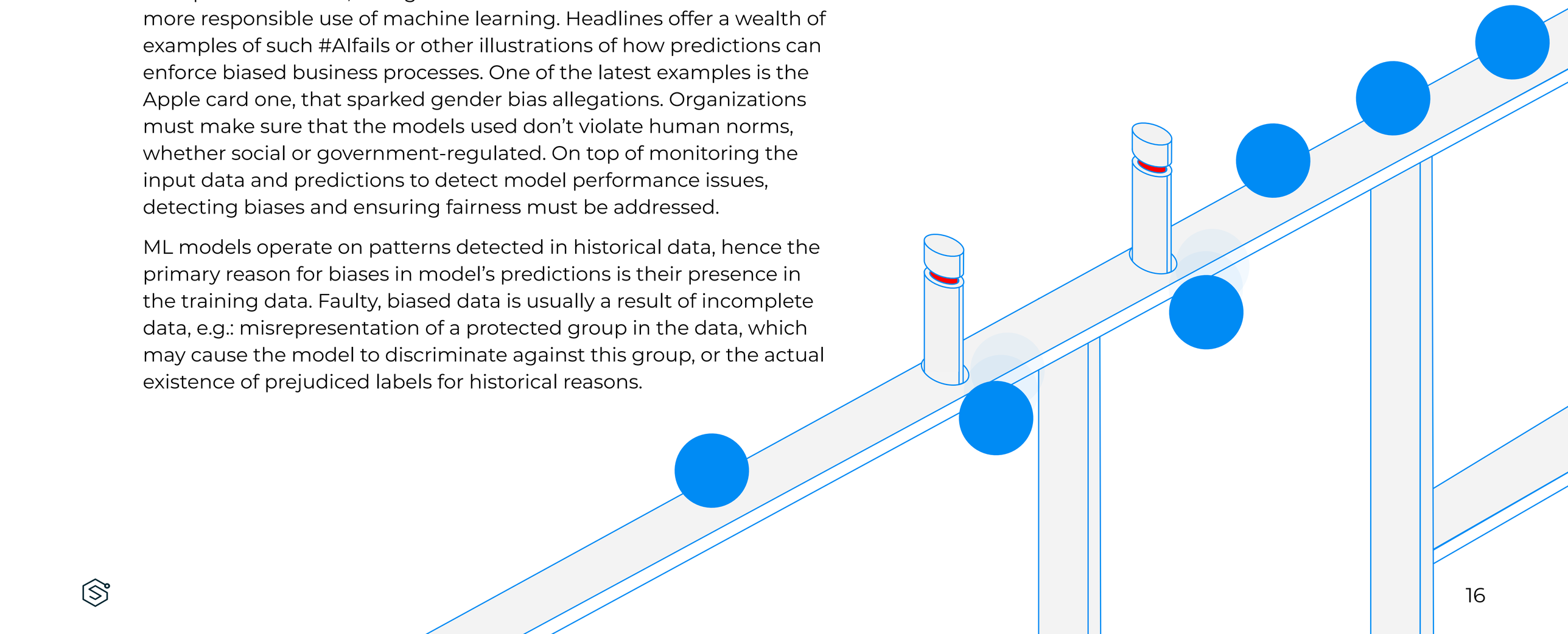
Leverage shadow models, A/B testing or multi-armed bandit to ensure the safe roll out of your models or a mix of a few of them together depending on your use cases.

The monitoring service needs to be able to collect and analyze the status and KPIs of the new rollout version, and enable advanced testing for various use cases. Even if you set aside the complexities related to the measurements of two models at once and their comparisons, to assure safe(r) model roll outs, the monitoring service should be able to measure the distribution of the decision making and its confidence levels, on top of the performance KPIs to detect the specific level of bias metrics, and to be able to do so across the dataset and for specific high-resolution segments.

# 6 Monitor protected features as proxies to ensure fairness

With a host of regulations arising that aim to ensure a fairer and more transparent use of AI, all organizations should seek to establish a more responsible use of machine learning. Headlines offer a wealth of examples of such #AIfails or other illustrations of how predictions can enforce biased business processes. One of the latest examples is the Apple card one, that sparked gender bias allegations. Organizations must make sure that the models used don't violate human norms, whether social or government-regulated. On top of monitoring the input data and predictions to detect model performance issues, detecting biases and ensuring fairness must be addressed.

ML models operate on patterns detected in historical data, hence the primary reason for biases in model's predictions is their presence in the training data. Faulty, biased data is usually a result of incomplete data, e.g.: misrepresentation of a protected group in the data, which may cause the model to discriminate against this group, or the actual existence of prejudiced labels for historical reasons.



# 6 Monitor protected features as proxies to ensure fairness

Take the loan approval case where it’s important to make sure there is no discrimination against a certain minority group, associated with a protected feature. Organizations try to avoid disparate treatment, i.e. provide different output for people in the minority group, compared to similar people in the majority group. As a first step, the protected feature that is associated with this minority group (e.g. gender, race, etc.) must be removed from the training data. Yet, and given that there may be other features as proxies which can be used by the model to understand the belonging to the minority group, this may not be enough.

The first step to preventing biases in models is to look for them in the training data in the research phase, before training and deploying to production. However, no matter how careful the data scientist is and even if the model was developed and evaluated to ensure it’s free of biases, the data in production continuously changes. There is always a risk that a bias which hasn't been seen so far may show up and increase, or even be amplified by, the model itself. Therefore, it is important to continuously test your model for  biases with the live production data.

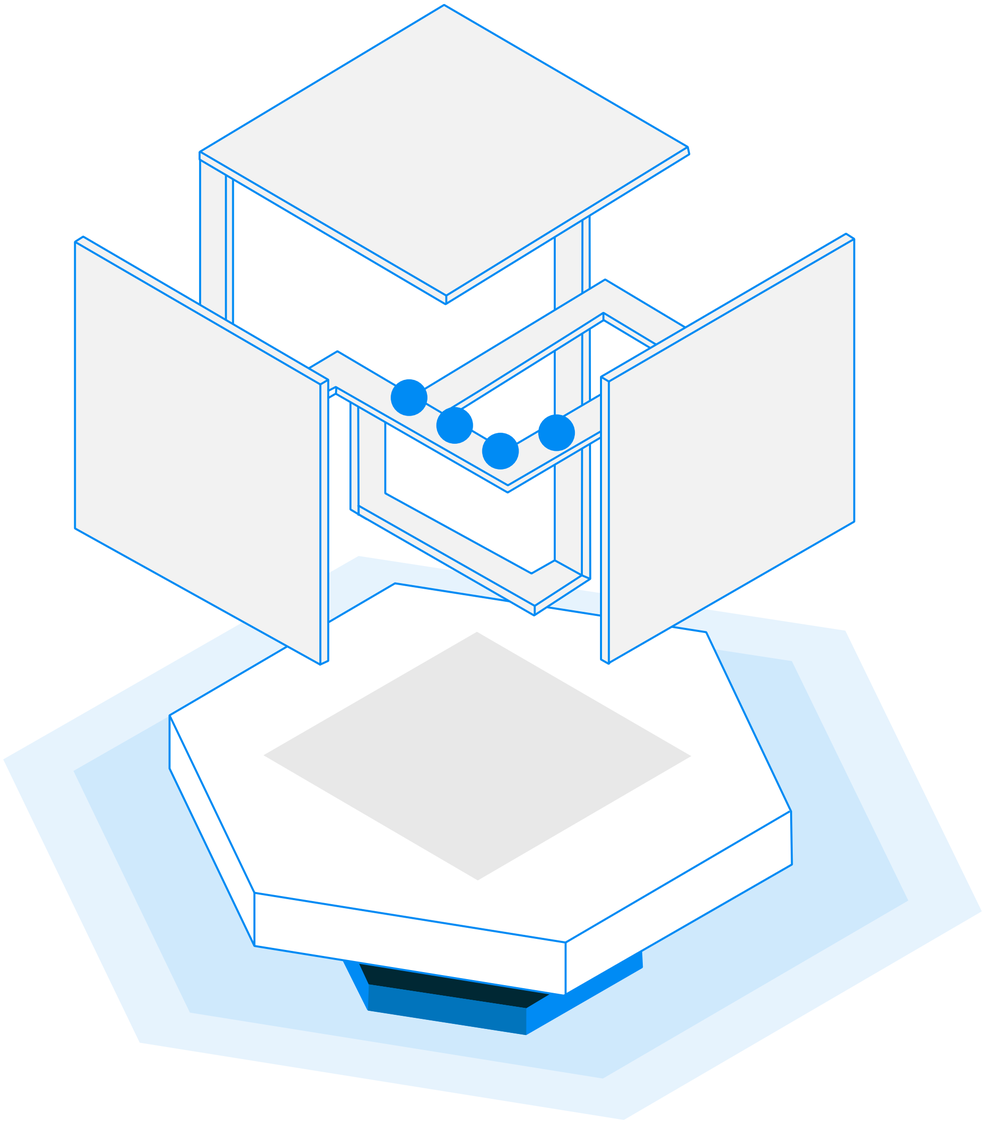
# 6 Monitor protected features as proxies to ensure fairness

A common approach for detecting disparate treatment, both in research and in production, is to check the importance, as a proxy to effect, of the protected feature in the decision making process. If the protected feature has high importance, it means it affects the decisions of the model and so the model may be biased. Note that in this case, since the protected feature was removed from the training data, it is not enough to monitor only the features used by the model but also to analyze and monitor external features not used by the model.

A more complex case of fairness and bias detection is when looking for disparate impact, i.e. the model provides outputs that benefit, or hurt, a group of people sharing a value of a sensitive attribute more frequently than other groups of people. In this case, the labels must be taken into account and the error rates of the different groups must be measured to validate that the protected groups error rates are not higher than other groups.

# 7 Remain platform agnostic

As ML has become ubiquitous, more and more teams within the same organization develop and deploy ML and very often use different platforms (SageMaker, TensorFlow, Pytorch,...). To accomodate this agility, monitoring should be decoupled from the machine learning platform so that it remains agnostic and can offer insights across all the different types of technologies  that models are being deployed with (Python, Java, R,..).



# 8 Empower the different stakeholders

The insights derived from a proper monitoring solution go beyond data scientists or engineers users. Rather, it serves all the model's stakeholders, including everyone whose work is dependent on the predictions. In many companies, the common stakeholders are divided to three groups:

**The data science group**

who research and develop the model. They want to see data distribution & understand what happens with the model and the performance of the model in production.

**The engineering group**

who deploy the model to production & are responsible for the data pipeline. They want to catch potential data integrity issues and ETL problems early on.

**The operational group**

who use the model's predictions & want to make sure that the business is not at risk and have the confidence to run as fast as they can with the predictions.

# 8 Empower the different stakeholders

One of the many pitfalls related to the use of AI at scale is the question of ownership of the models once they are in production. Very often, the prerogatives of assuring the health of the models in production is attributed to the data science and the operational teams who struggle to establish a common language, and who struggle to derive the insights they require.

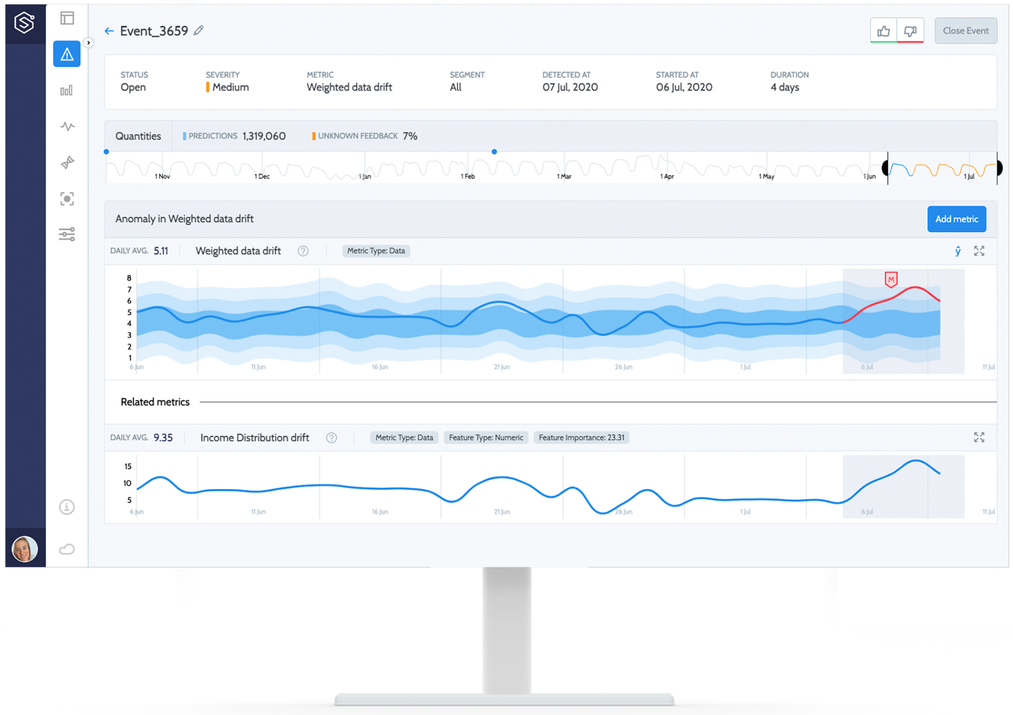
In this sense, the definition of ML monitoring should be expanded to AI assurance so as to empower each stakeholder to have control, and visibility.

For example, the engineering group can use the framework to detect data integrity and technical issues, the data science team can use it to detect cases where the model is not optimal and should be retrained (e.g. when there is a significant input drift), and the models users, usually analysts, can use it to monitor sub-populations (e.g. VIP clients) or understand changes in the model behavior by analyzing the changes in the input features.

It is necessary to offer a view of the predictions and data behaviour in production that satisfies the needs of all stakeholders, and that enable them to be more proactive and efficient. Whether via APIs or dashboards, data science, engineering and operational teams should be able to speak a common language and have an objective view of their data's behaviour.

# 8 Empower the different stakeholders

For the operational teams, they need to derive the necessary insights from the model's health to foster trust and step away from the "black box" paradigm of AI implementations. They also need to be independent enough to have access to a view of the predictions behaviour that doesn't require the involvement of the data science team.

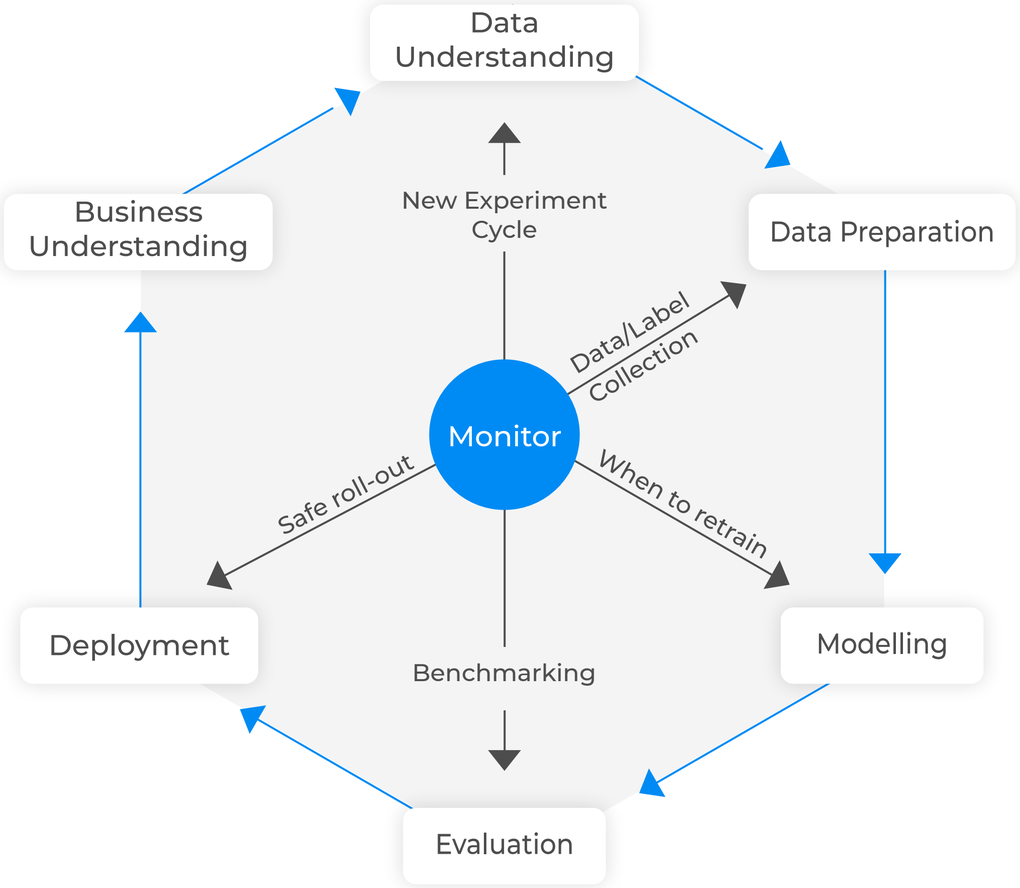



Superwise.ai event investigation view with all correlated metrics

# 9 Use your production insights for other stages of your ML process

Production is not a stand-alone part of the ML process. For example, take a model in production which is automatically retrained every month with the last 3 months of data. If there was a known data integrity issue in several days during this period, these days can be excluded from the data since they do not represent the normal data distribution and might harm the model learning process. In another case, if a concept drift is detected, which means the relationship between the input features and the target variable changed, taking historical data prior to the change will also harm the model learning.

More to the point, to connect the insights from production to your training pipelines or business dashboard, we recommend gaining access to the insights from production automatically via APIs.

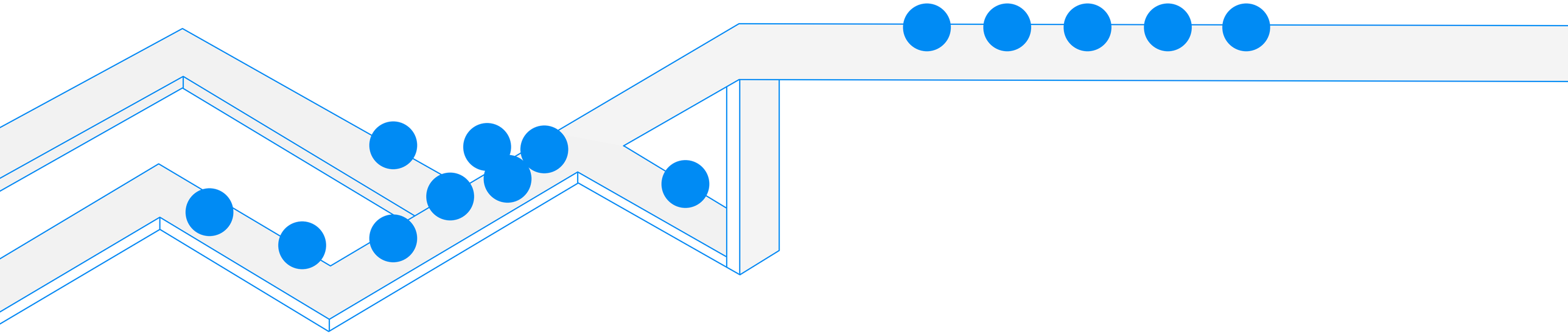



**The insights generated in the production phase should aid in designing better models and drive better business understanding regarding the process it supports.**

# Conclusion

To ensure that your models have the impact they were designed for and to avoid the pitfalls of “Day 2” when your models meet the real world, is directly tied to effective monitoring. This is why ML monitoring needs to become an integral part of the ML infrastructure.

Yet, to implement and benefit from effective monitoring throughout every stage of the ML flow, and to ensure optimal functionality in production one needs time and resources to look beyond simple metrics and manual thresholds. By granting monitoring the proper place in the ML orchestration flow, organizations and users can gain the control, visibility and efficiency necessary to power the use of AI at scale.



# About Superwise

Superwise is the company that  monitors and assures the health of your AI models while alerting you when something goes wrong. At the right time.

At Superwise, we help organizations overcome monitoring challenges to create models that behave as expected production.

Our solution enables data science and business operations teams to reduce the labour intensive efforts invested in the maintenance of AI in production, and shorten the time to detect and fix issues.

We already monitor and assure the health of AI-based models for global players in the area of financial services, fraud prevention, insurances, and marketing.

To find out more about how we do this and brainstorm on ways to improve your organization’s ML monitoring processes, get in touch with us today.

www.superwise.ai