Model Monitoring

Robert Clements

MSDS Program

University of San Francisco

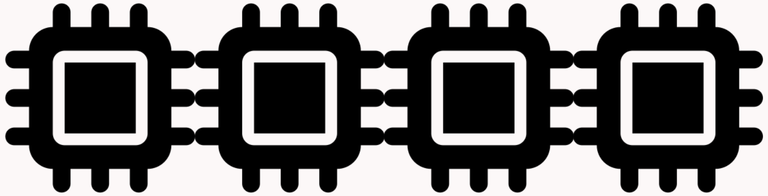
What to Expect

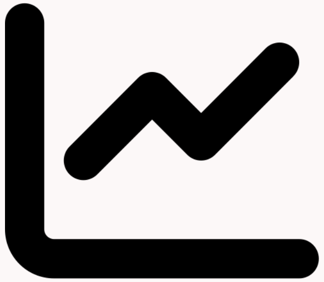
- Goal: to understand the importance of monitoring model data and performance, and what to do when model degradation happens.

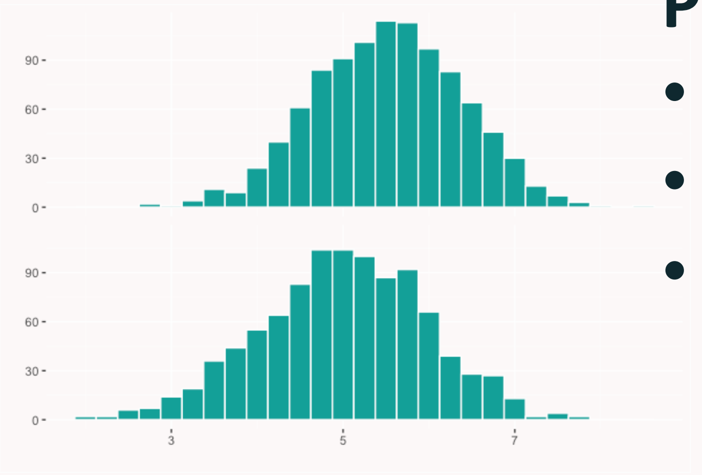
- How: we will explore the different ways of tracking model performance and practice with a dataset.

Operational and Model Performance

Operational:
- Scalability
- Latency
- Failures





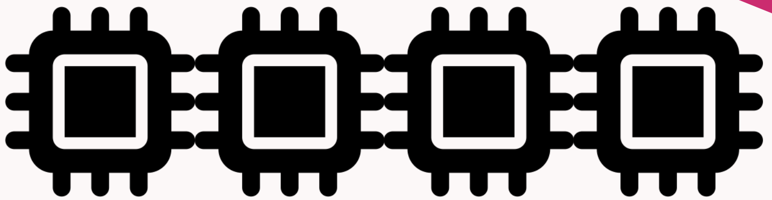


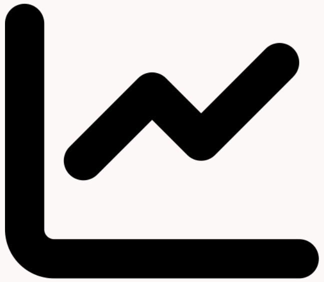
Performance:
- Degradation
- Data/Concept Drift
- Metrics

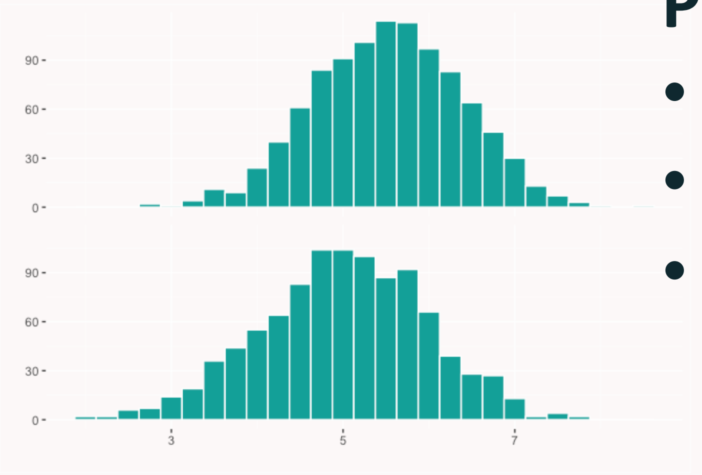
Operational and Model Performance

THIS IS USUALLY PROVIDED BY EXISTING DASHBOARDS

Operational:
- Scalability
- Latency
- Failure



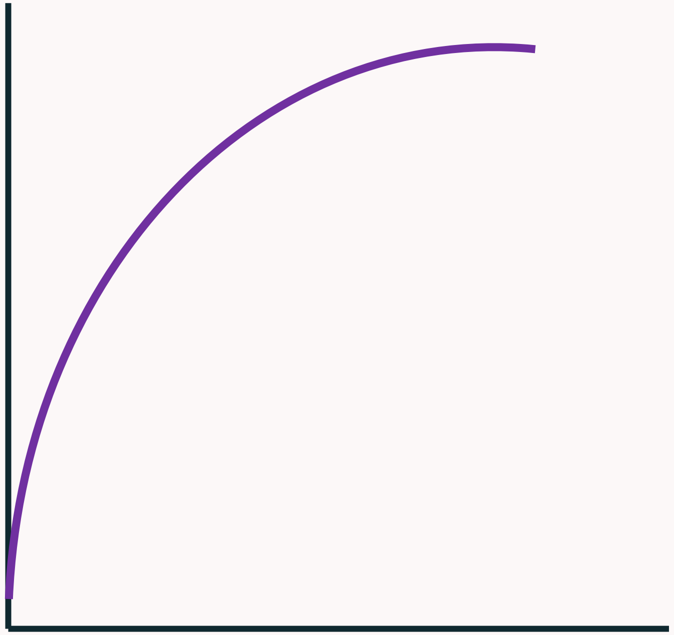




Performance:
- Degradation
- Data/Concept Drift
- Metrics

Model Performance

Metrics, metrics, metrics!!

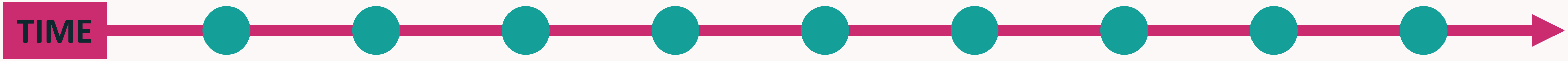


AUC
AUPRC
F1
Accuracy
MSE
RMSE
R-squared
etc.

- Metrics should be context-specific, something the business owner cares about
- Model performance requires labels, which may not be available quickly enough
    - Drift detection can give you a signal that performance is about to drop

Two Important Decisions

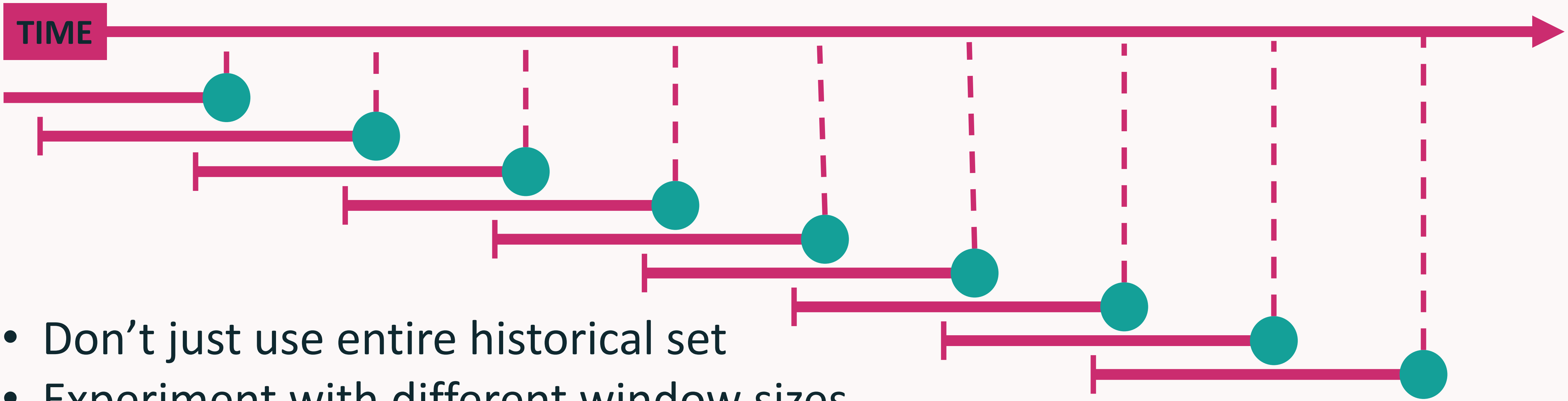
How often should we measure our model's performance?



- Do we have labels?
- Is our model **online** or **offline**?

Two Important Decisions

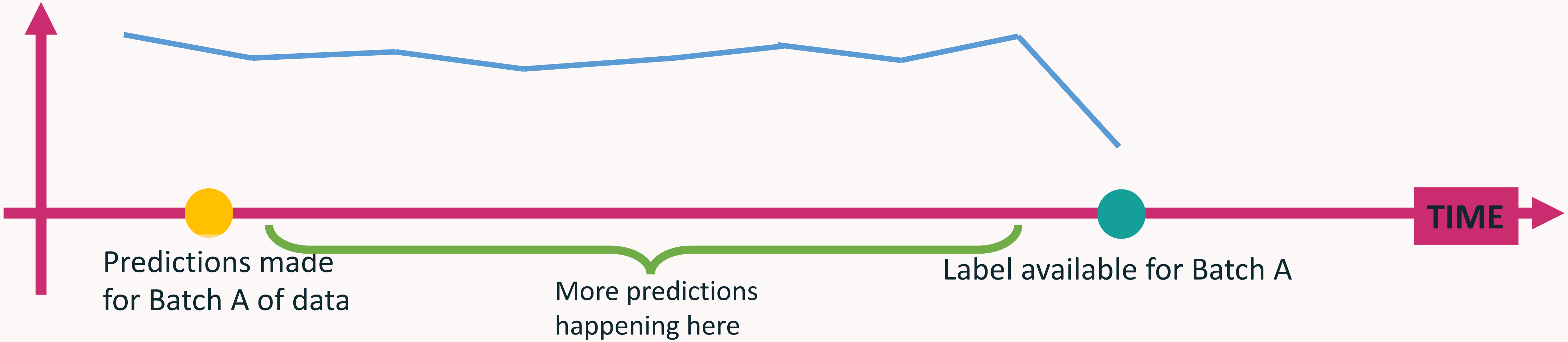
How much historical data should we use for our metrics?



- Don’t just use entire historical set
- Experiment with different window sizes
  - Maybe even monitor at multiple sizes

Delayed Labels

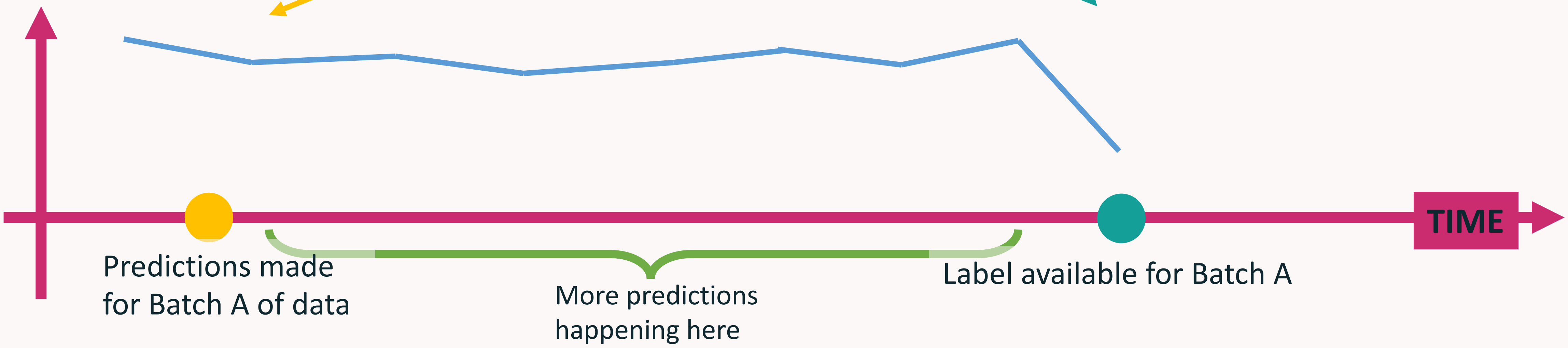
- In many situations we may need to wait a long time for labels
- We cannot rely on model performance metrics alone
- Identifying degradation too late means we're already losing value
- Might be able to "approximate" performance
- Better to simply use a more robust monitoring system and identify cause of degradation before it happens



Delayed Labels

- In many situations we may need to wait a long time for labels
- We cannot rely on model performance metrics alone
- Identifying degradation too late means we're already losing value
- Might be able to "approximate" performance
- Better to simply use a more robust monitoring system and identify cause of degradation before it happens

Degradation ID'd **here** happened back **here**



Model Decay

Over time, model performance will **decay** – sometimes quickly, sometimes slowly (over years). Models decay for two main reasons:

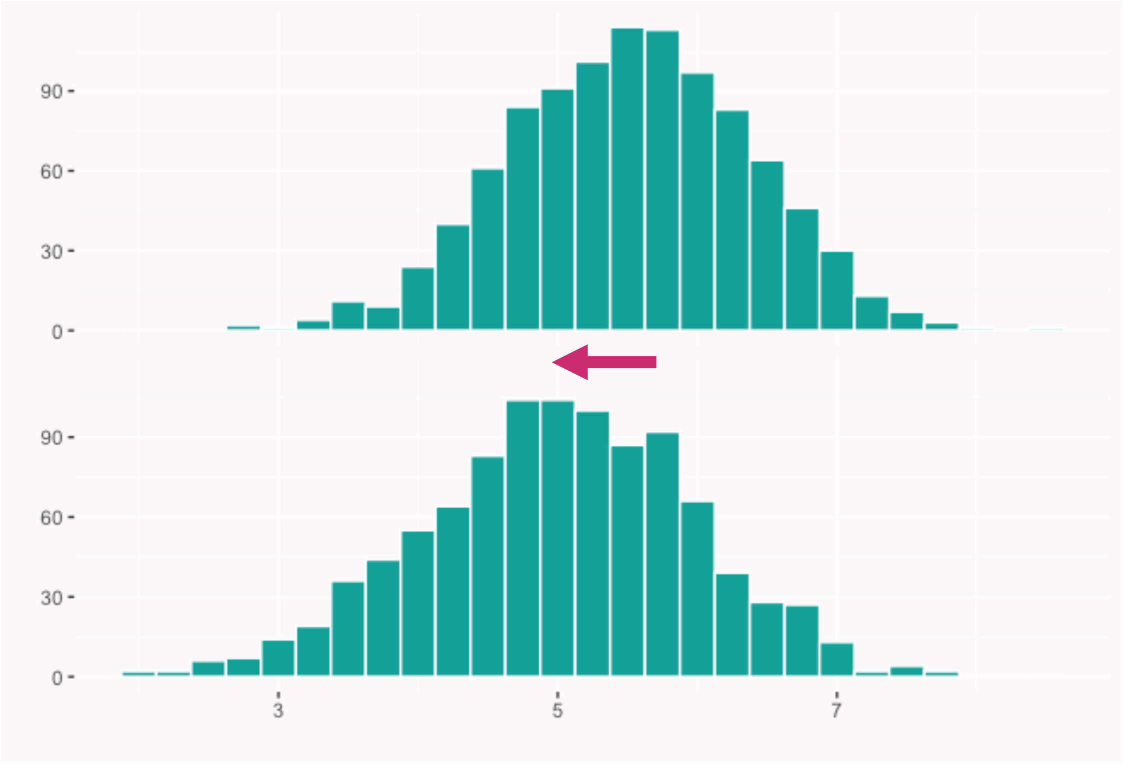
- Data Drift
  - Changes in input data – erroneous or real
- Concept Drift
  - Changes in the relationship between inputs and outputs (the target)

First: Data Integrity and Quality

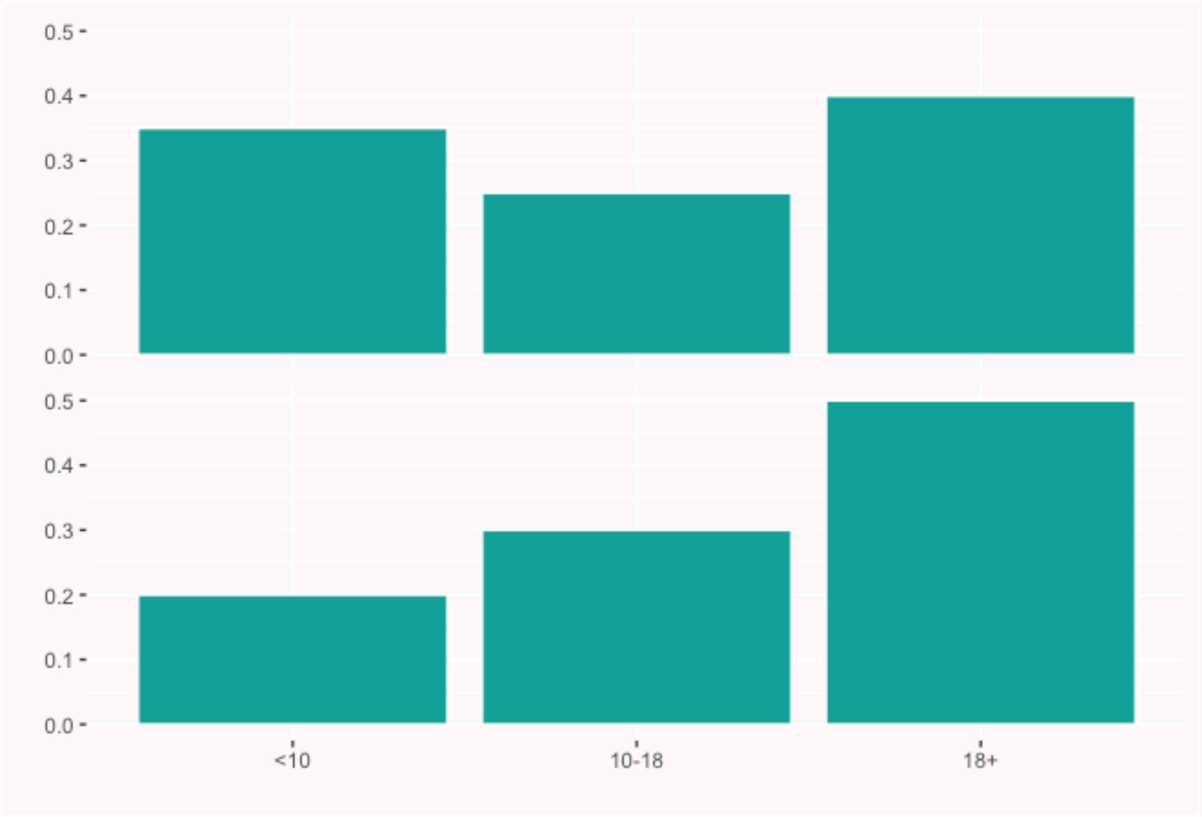
- Rule-based
- Think Great Expectations
- What do we expect our data to look like?
  - Missing values = pipeline problem
  - Values outside prespecified range = out of scope, pipeline problem
  - Column of wrong type = change in schema
  - Entire rows or entire columns missing = pipeline problem
  - Duplicate rows = data loading issue

Data Drift

Univariate distribution change



Continuous



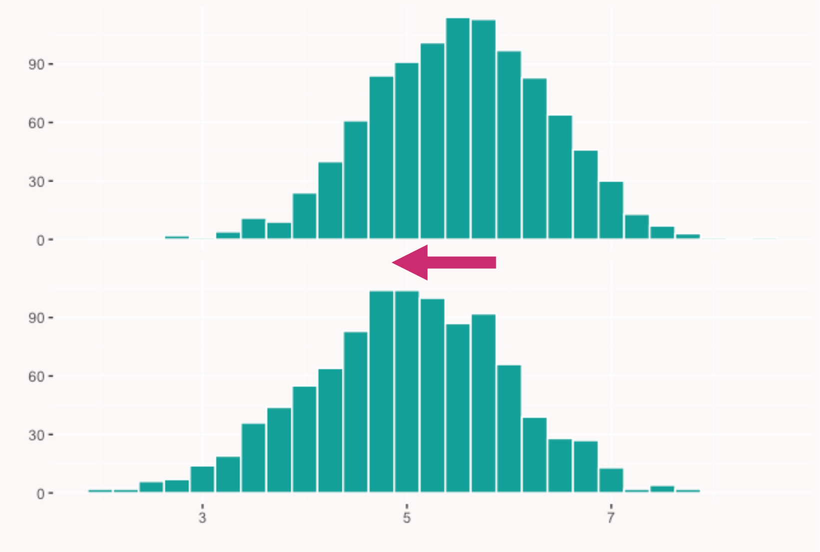
Categorical

Target Drift

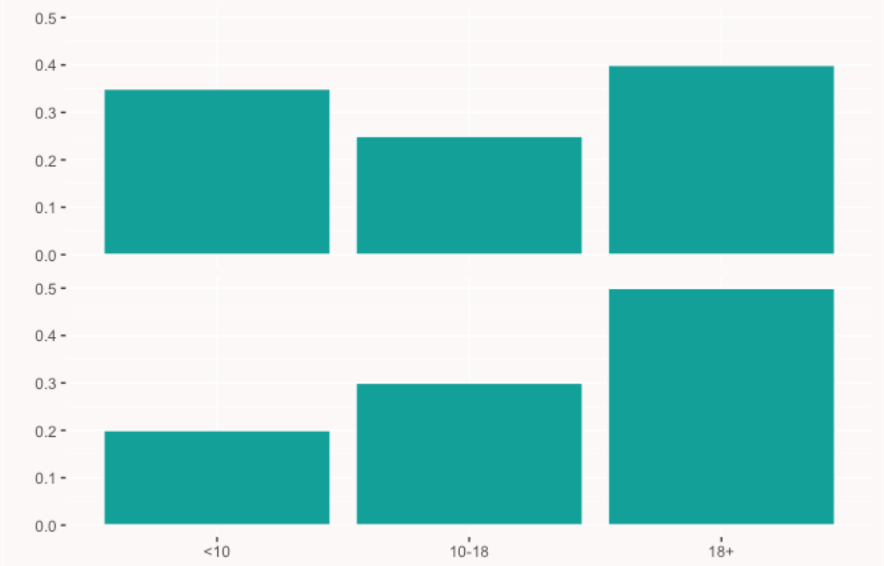
Univariate distribution change

Target drift happens if:

- Distribution changes
- Categories are added or removed



Continuous



Categorical

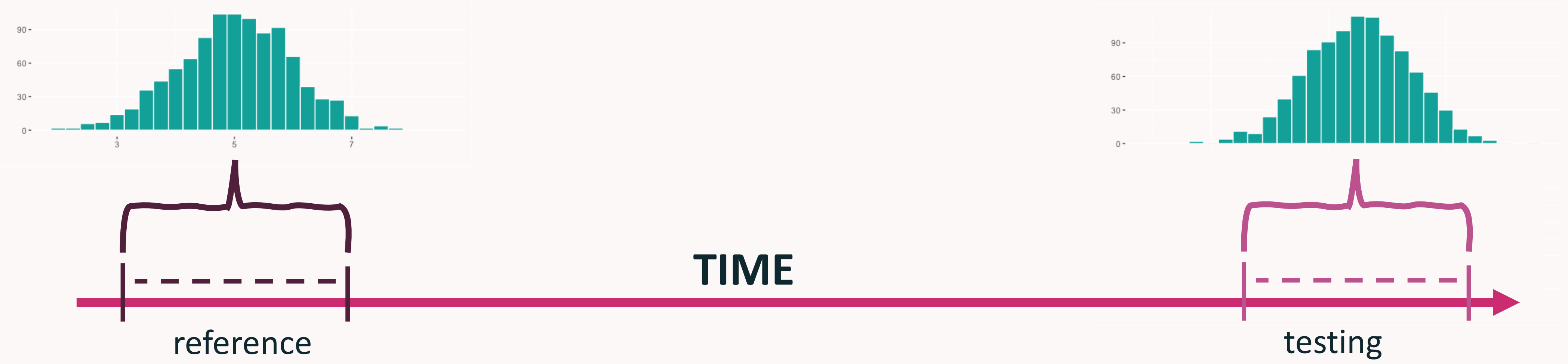
Concept Drift

$$Y \not\mapsto f(X)$$

Example:
Recommender systems - people's buying patterns change over time due to lifestyle changes. COVID-19 was a major cause of a lot of changes that models needed to adjust to.

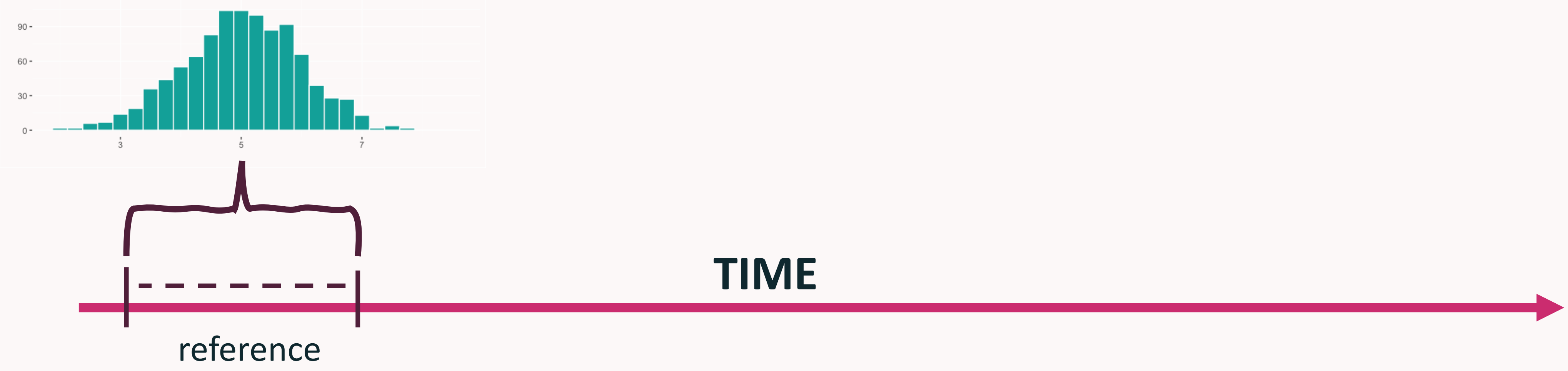
Detecting Data Drift - General Concept

- Step 1: choose a reference period (e.g. features and scores data from 2 weeks ago; subset of training data)
- Step 2: define a testing period (e.g. most recent 2 weeks of features and scores data; most recent 1 day)
- Step 3: compare difference between testing and reference data distributions



Detecting Data Drift - General Concept

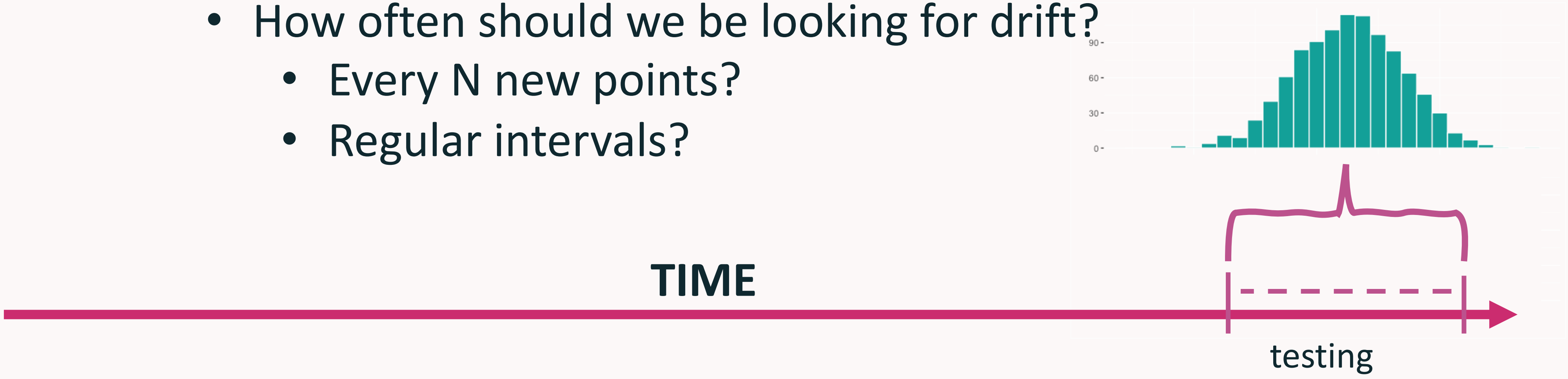
- Step 1: choose a reference period (e.g. features and scores data from 2 weeks ago; subset of training data)
  - Should it be from the training set?
  - Should it be static, or should it be a sliding window?
  - How big should the window be?
    - Big enough such that the distribution is “stable”



Detecting Data Drift - General Concept

- Step 2: define a testing period (e.g. most recent 2 weeks of features and scores data; most recent 1 day)

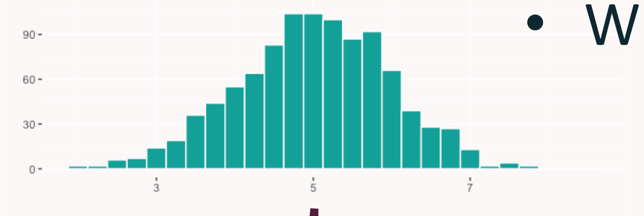
    - How big should the window be?
        - Small enough to detect drift quickly.
        - Big enough that we aren't measuring random variation.
        - We should probably use several window sizes.
    - How often should we be looking for drift?
        - Every N new points?
        - Regular intervals?

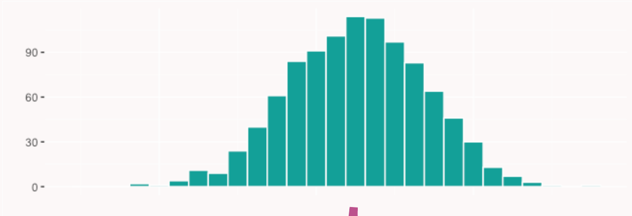


Detecting Data Drift - General Concept

- Step 3: compare difference between testing and reference data distributions

- What and how many metrics should we use?
- Should we use statistical tests or raw statistics?
    - When are p-values irrelevant?





Detecting Data Drift - Specifics

Are the distributions different?

Kolmogorov-Smirnov test

Wasserstein distance

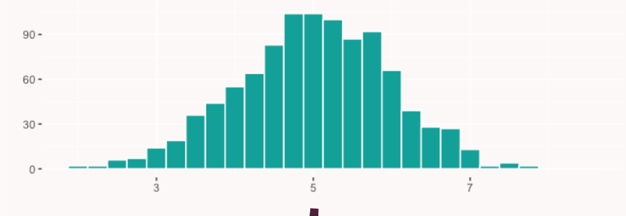
Jensen-Shannon divergence

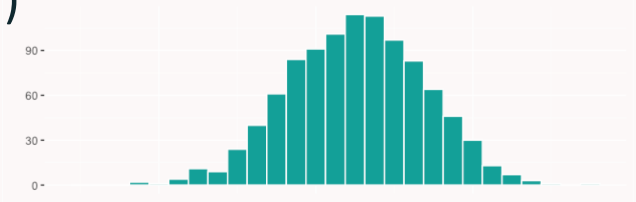
Kullback-Leibler divergence

Population Stability Index

Chi-squared test (categorical only)

Many others…





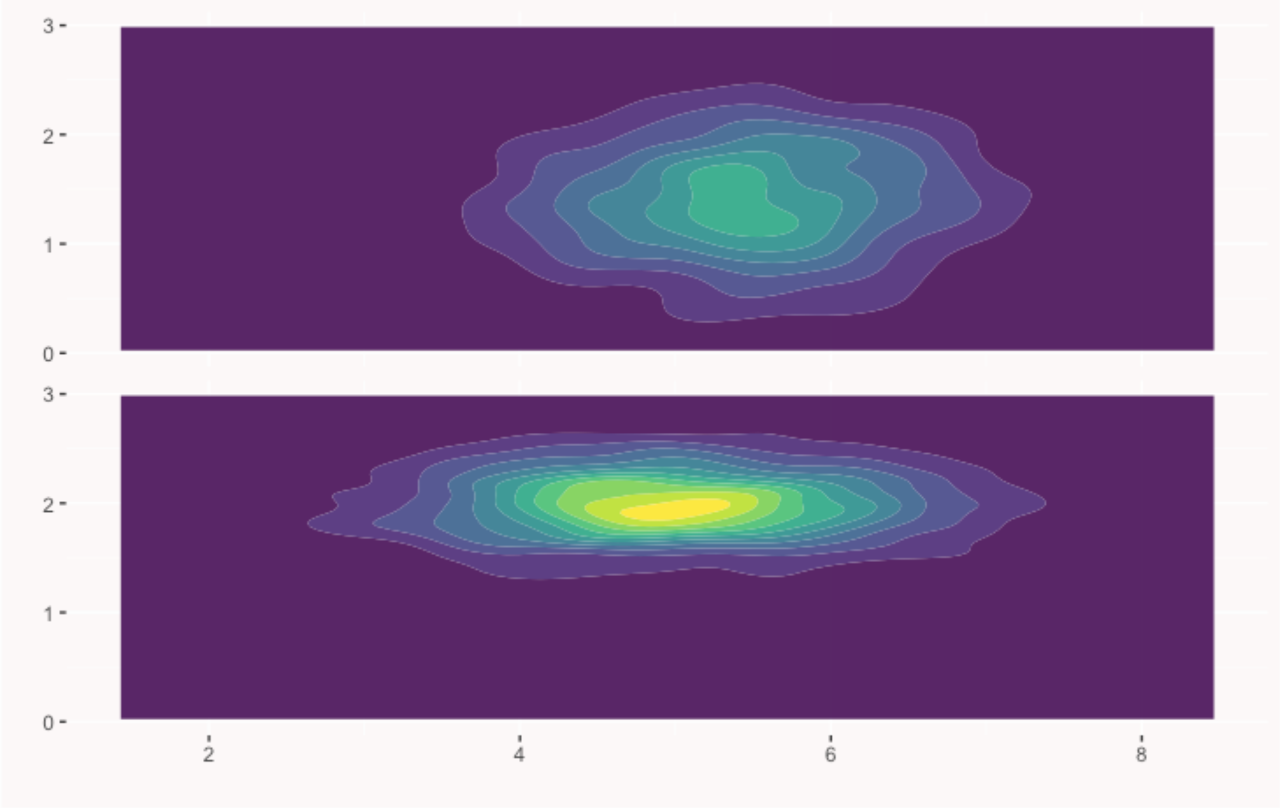
Kolmogorov-Smirnov Test (2 Sample)

Wasserstein Distance

Kullback-Leibler Divergence

Chi-Squared Test

Detecting Multidimensional Data Drift



How to detect drift that happens multidimensionally, rather than univariately?

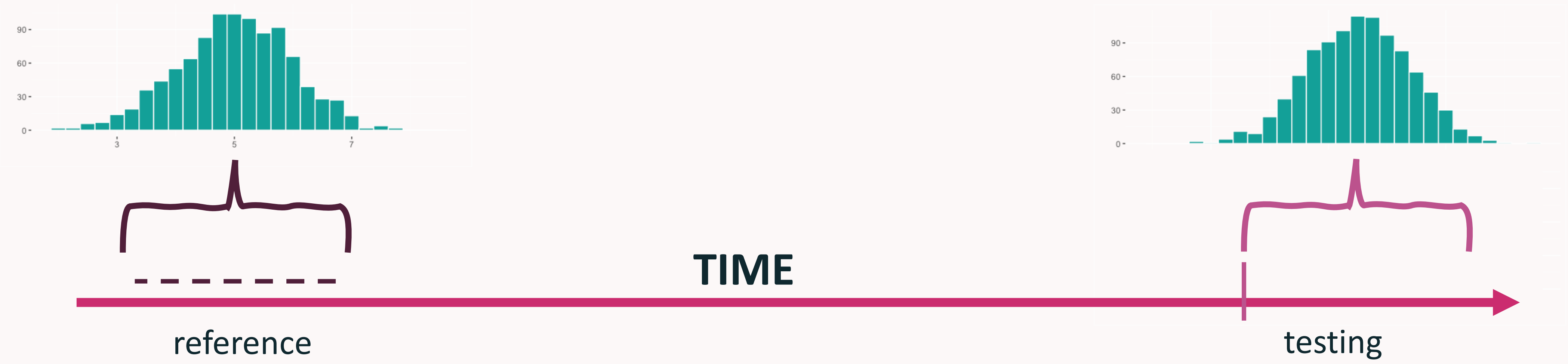
[Adversarial Validation]

Adversarial Validation

Text Data

- Use summary statistics of each string
  - Length
  - % of non-letter characters
  - % of out-of-vocab words

- Compare distributions of summary stats as normal



Embeddings

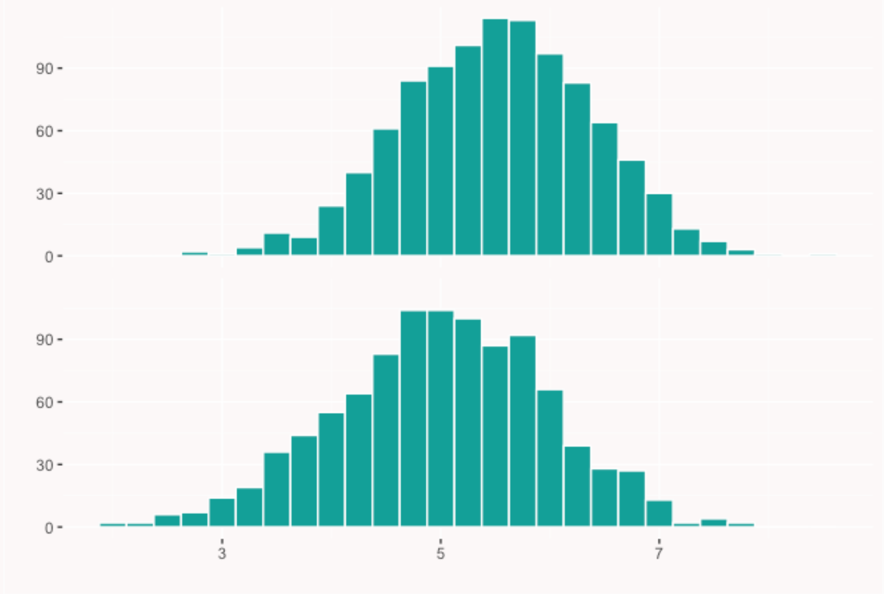
- Track the raw categorical features if possible
  - Use chi-squared

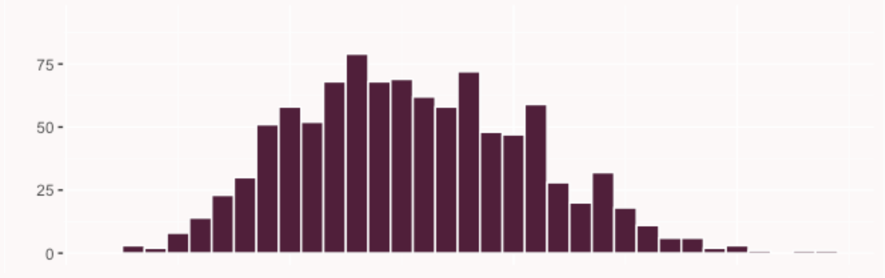
Animal	Pig	Dog	Cat	Cow	…
Pig	1	0	0	0	0
Dog	0	1	0	0	0
Cat	0	0	1	0	0
Dog	0	1	0	0	0

- Track the embeddings themselves
  - Use adversarial validation

1	2	3	4	5	…
0.2	0.15	0.21	0.29	0.35	…
0.1	0.08	0.12	0.19	0.20	…
0.25	0.02	0.21	0.20	0.17	…
…	…	…	…	…	…

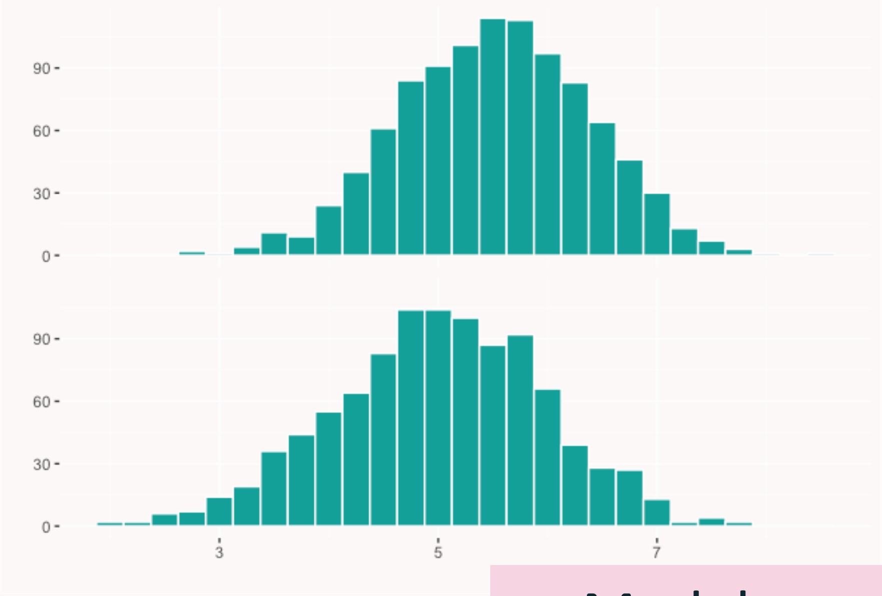
Scenario 1: Input data drifts, output data does not

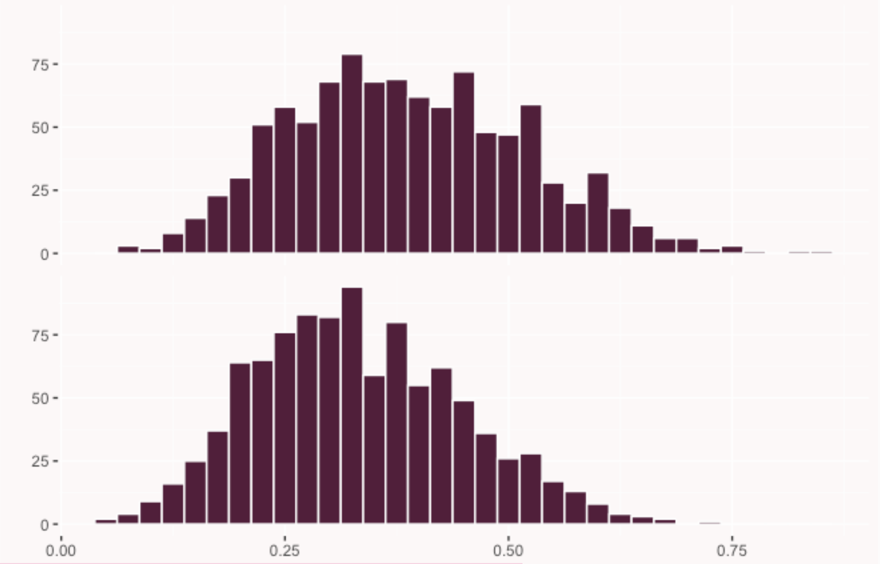




- Model is robust to slight drift, or
- Only unimportant features are drifting, or
- There was significant drift, and model couldn’t extrapolate properly.

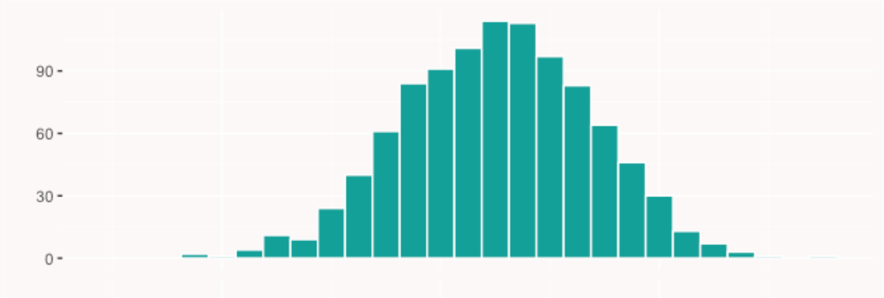
Scenario 2: Input data drifts, output data drifts

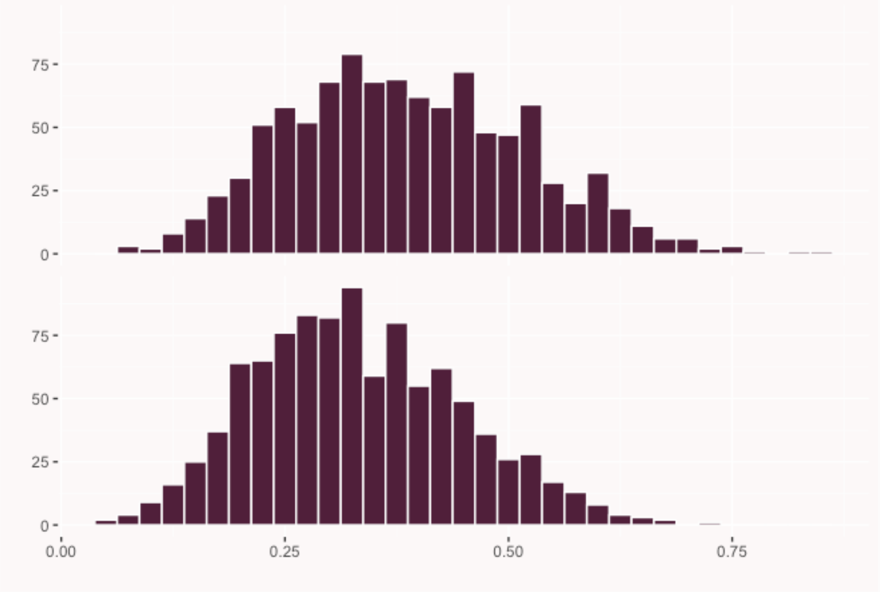




- Model reacted as expected to drifting data
- Model performance will likely suffer and retraining may be necessary.

Scenario 3: Output data drifts, input data does not

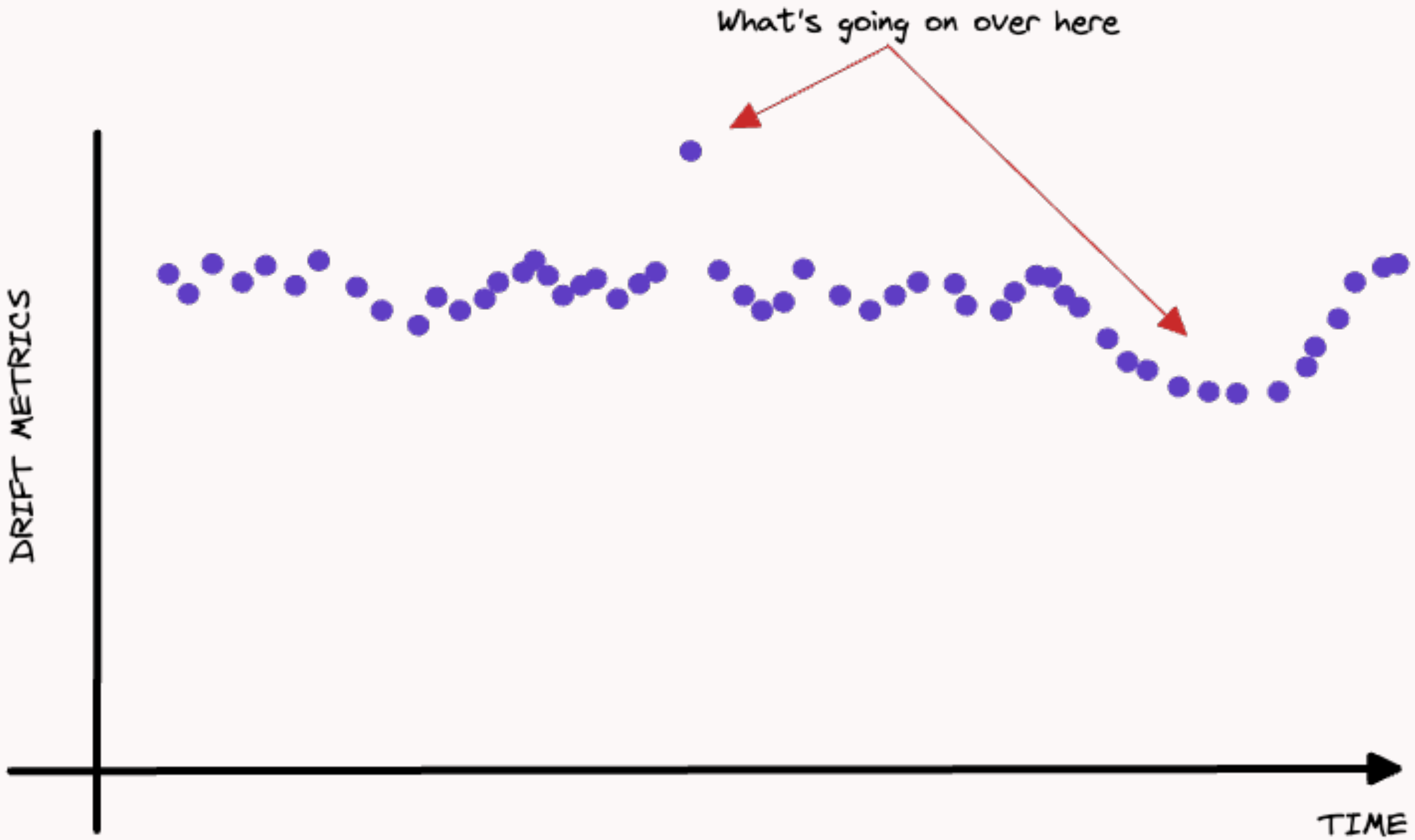




- There’s a bug in the drift detection, or
- There’s a bug in the model scoring code, or
- The data pipeline broke (e.g. not all scores being loaded into DB).

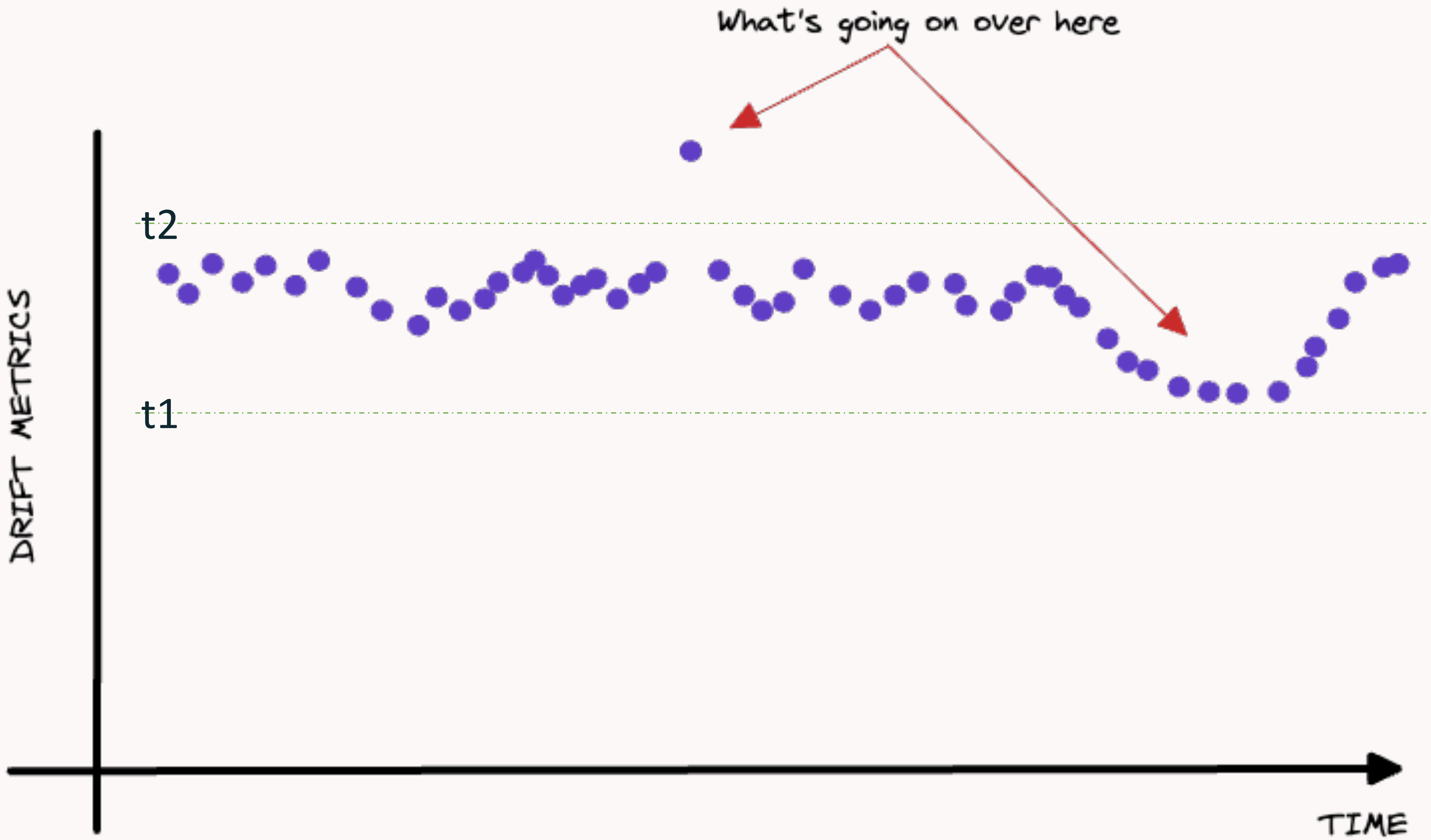
Outlier Detection

- Statistical tests may not be enough



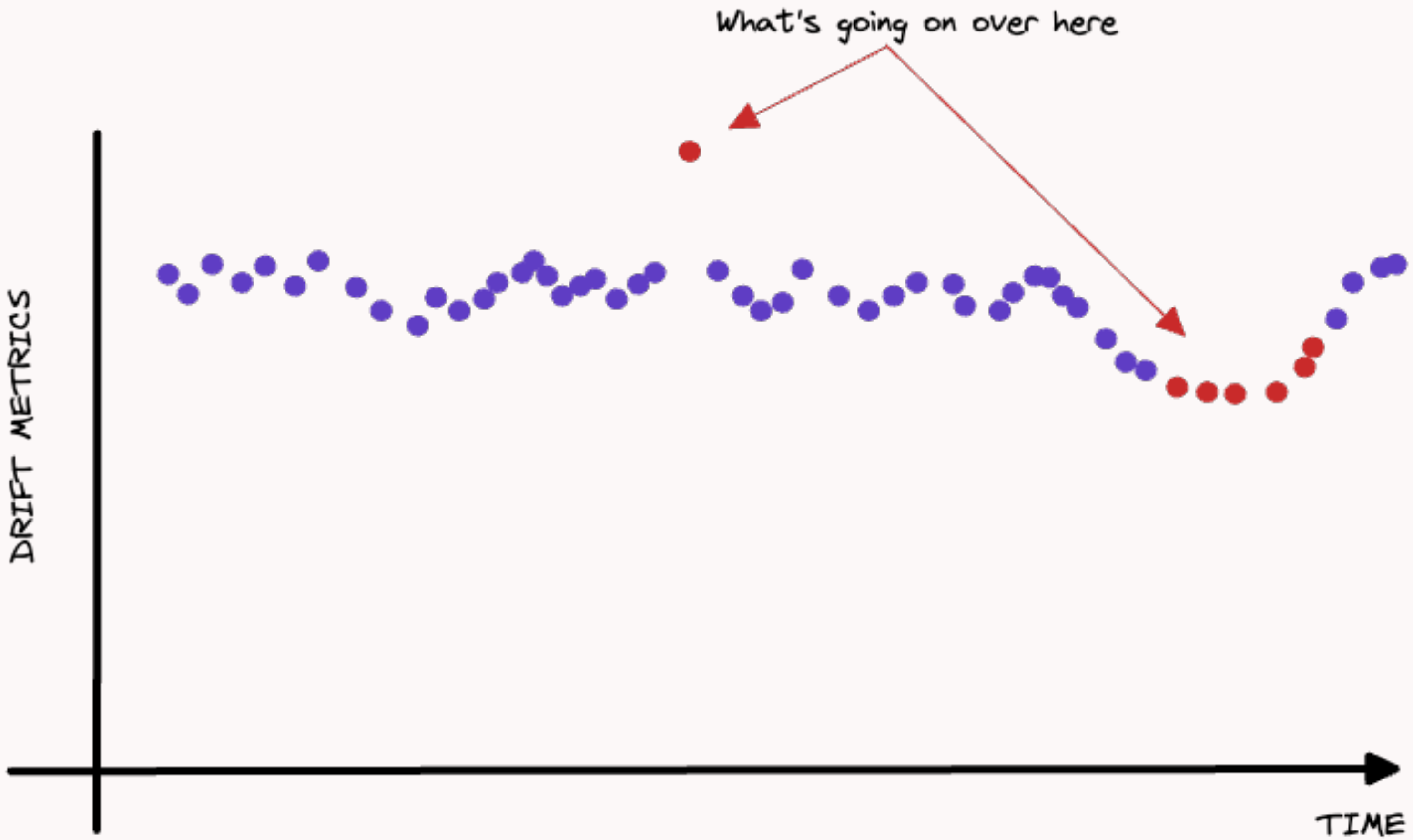
Outlier Detection

- Statistical tests may not be enough
- Can use hard-coded thresholds:
  - If metric > t2 or metric < t1 => drift!
  - Might miss collective anomalies
  - How do you choose thresholds?



Statistical or ML-Based Outlier Detection

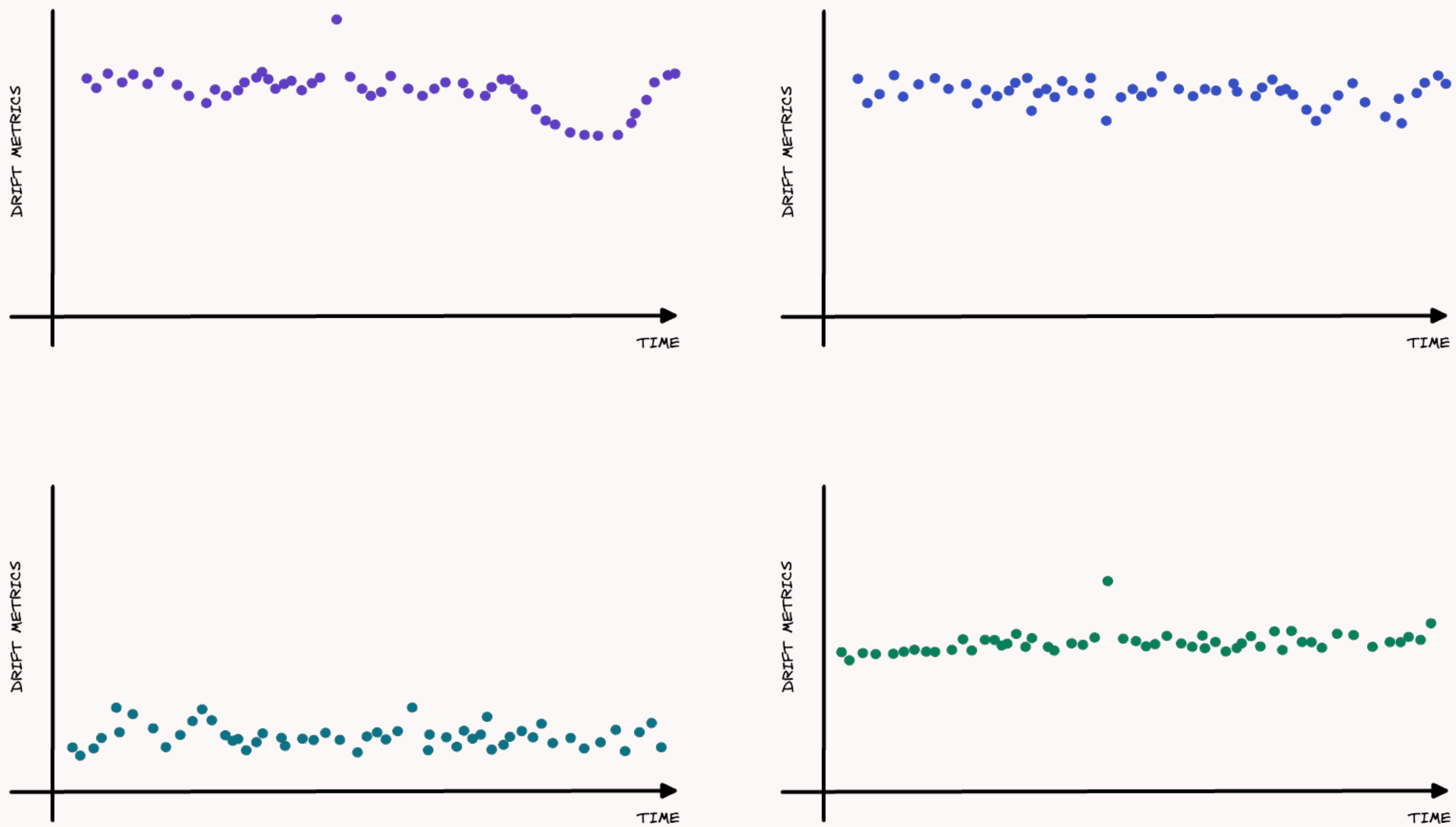
- Use detection methods for finding global and collective anomalies



Statistical or ML-Based Outlier Detection

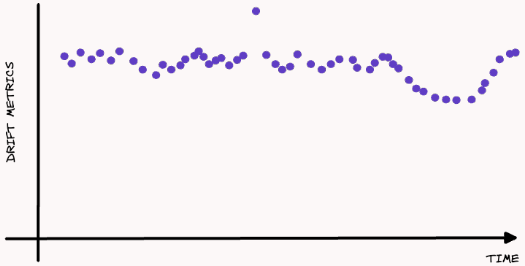
- Use detection methods for finding global and collective anomalies

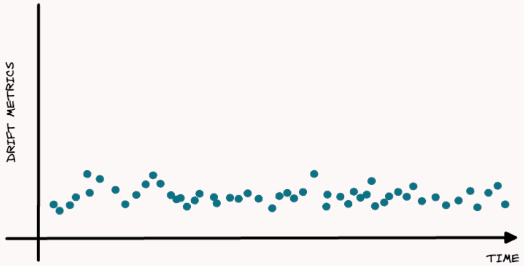
- Lots of metrics = lots of potential false positives

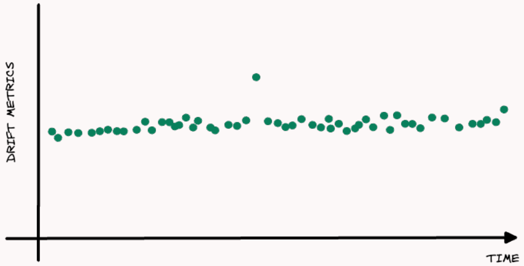


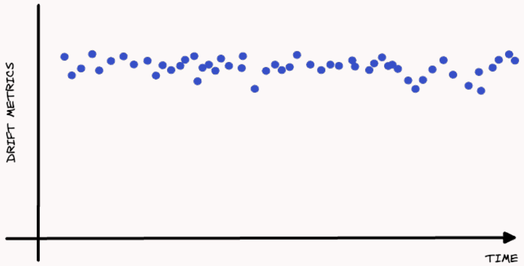
Statistical or ML-Based Outlier Detection

- Flag only most ***significant*** outliers
- Send alert only if > ***N*** flags
- Label alerts: 0 = False alarm; 1 = good alarm
- After enough labels are collected, train a classifier using original metrics as features
- Don’t overcomplicate things!!!









Plan of Action

- Alerting
  - Trigger alert via email, Slack/Teams message
  - Avoid alerting fatigue using significance thresholds
  - Provide all relevant info in the alert: feature that drifted; datetime of testing data; metric that detected drift

Plan of Action

- Investigation
  - Perform a root cause analysis
  - Check data schemas
  - Check for messages about broken data pipelines or data schema changes (these happen more often than you might think)
  - Get data scientists involved to look at the data distributions directly
    - Consider looking at different groupings or slices of data using the categorical variables

Plan of Action

- Action Items
  - Fix schemas
  - Rollback unnecessary changes to pipelines
  - Yell at the data team
  - Or, if nothing is broken upstream, this means your model may now be broken. Go to next section.

Tools

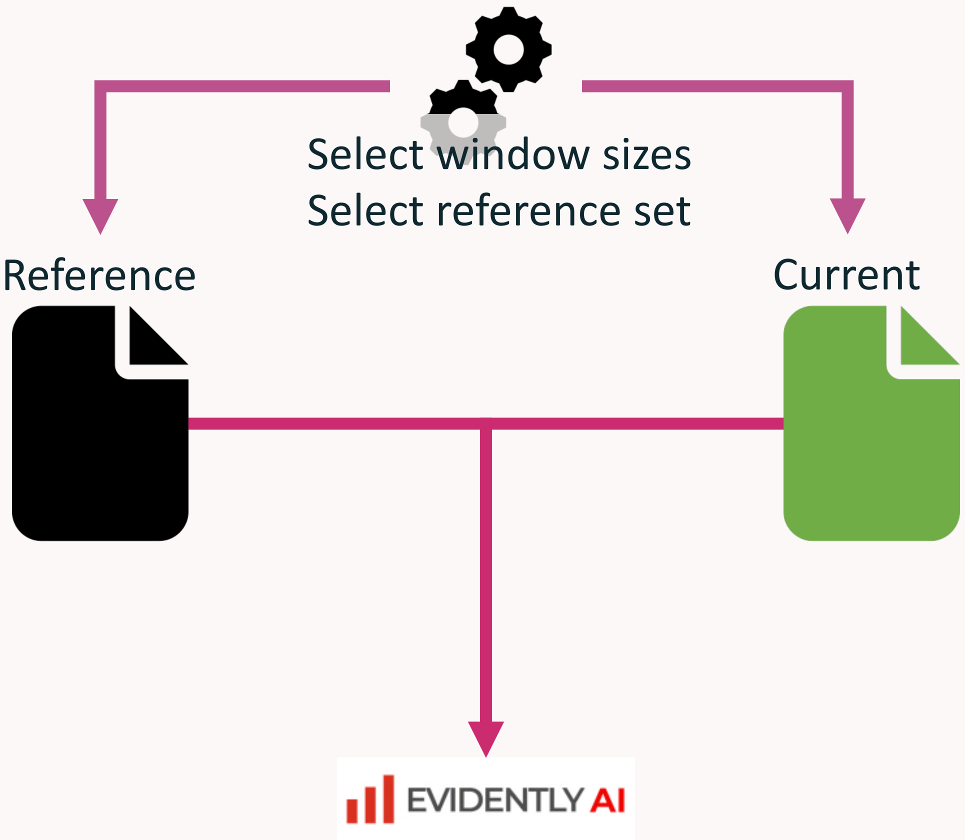
- Evidently AI: creates drift, performance, and integrity metrics
- Seldon – Alibi Detect: includes outlier detection
- Prometheus + Grafana: you probably will need to code your own metrics
- MLRun: tries to do a lot of things
- Scikit-Multiflow: for streaming data
- And others…

Model Monitoring with Evidently

Evidently

- Free, open source python library
- Computes metrics for
  - Drift
  - Integrity
  - Model Performance
- Computes tests for alerts
- Creates exploratory and customizable reports and visualizations
- Integrates with Airflow, Metaflow, MLFlow, Grafana

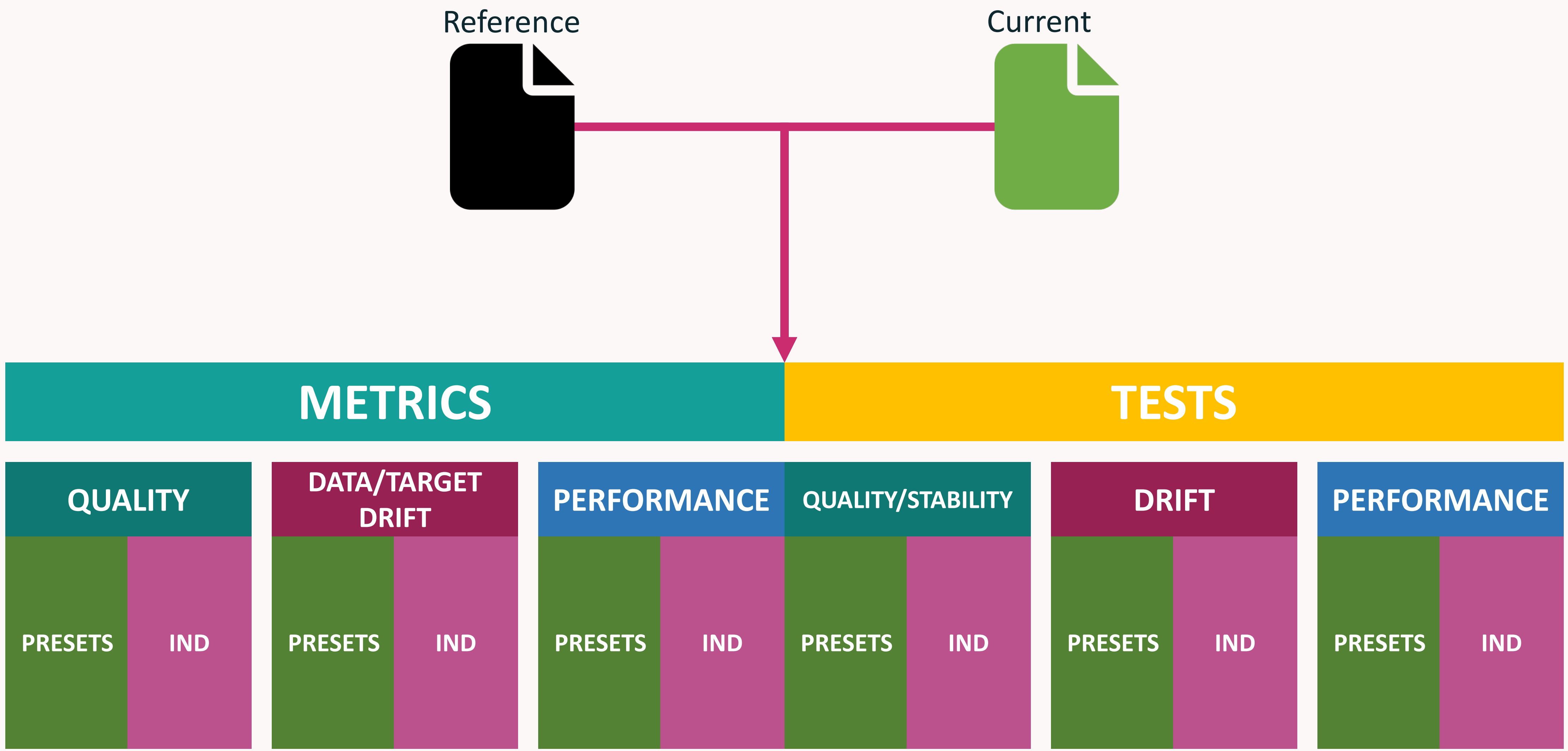
Evidently



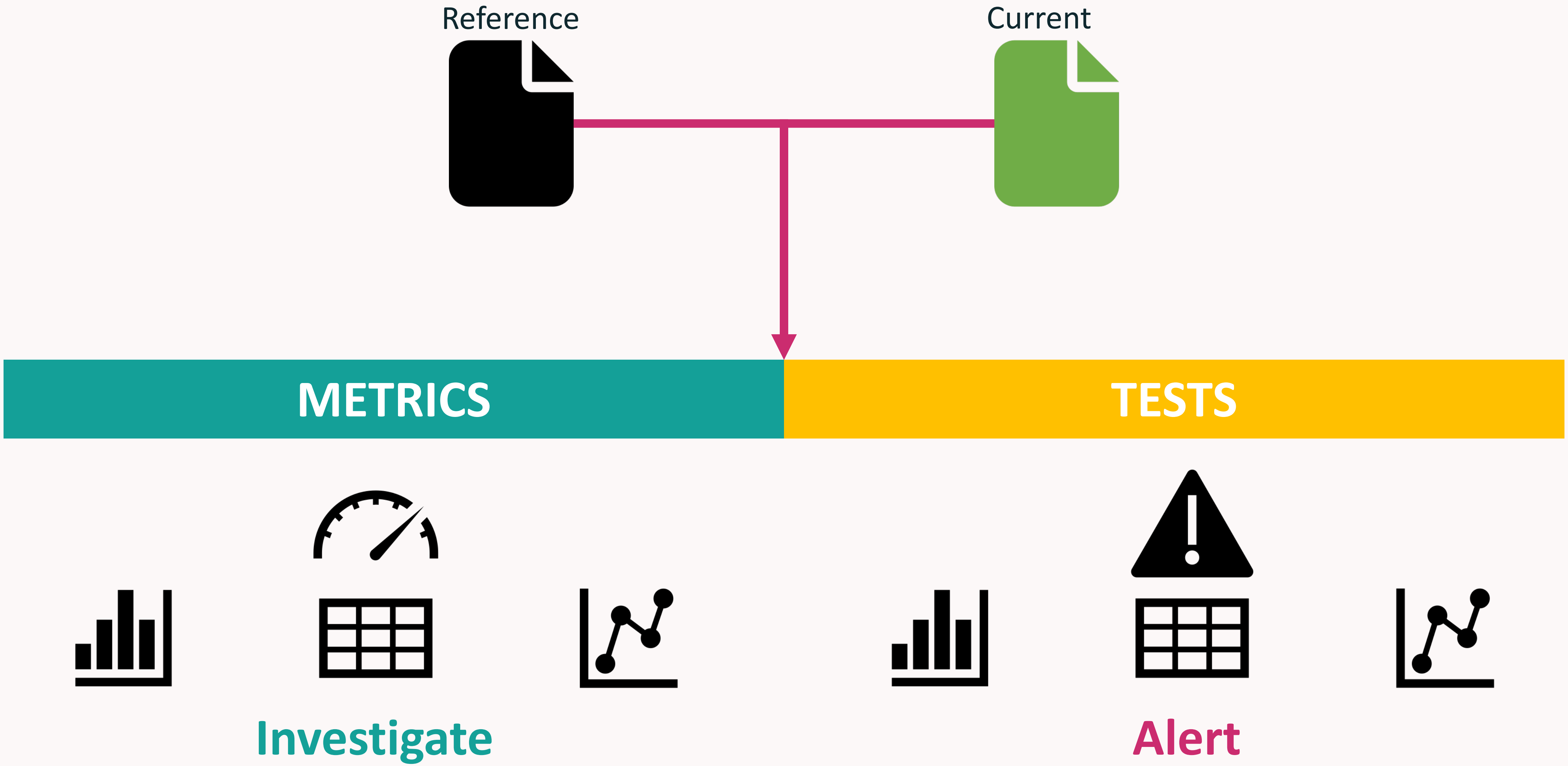
Feed Reference and Current sets to Evidently

Evidently **will not** do the data processing for you!

Evidently


Reference
Current
METRICS
TESTS
QUALITY
DATA/TARGET DRIFT
PERFORMANCE
QUALITY/STABILITY
DRIFT
PERFORMANCE
PRESETS
IND
PRESETS
IND
PRESETS
IND
PRESETS
IND
PRESETS
IND
PRESETS
IND

Evidently



Evidently Visualizations

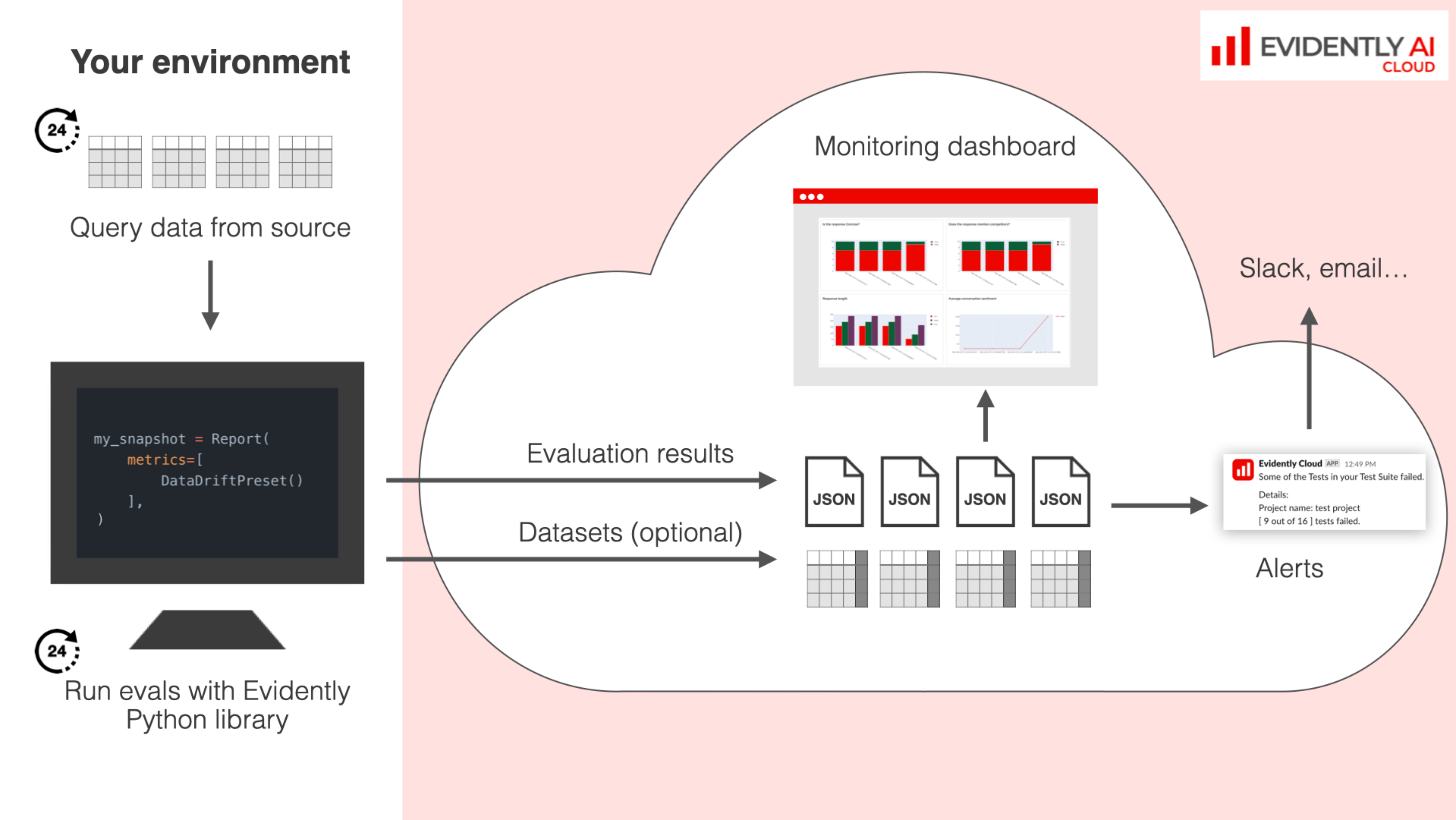
Source: https://docs.evidentlyai.com/readme/core-concepts

- Save as html – open in browser or use within other tools (Streamlit)
- Save as json, python dictionary – build your own visualizations
- View inline in a notebook

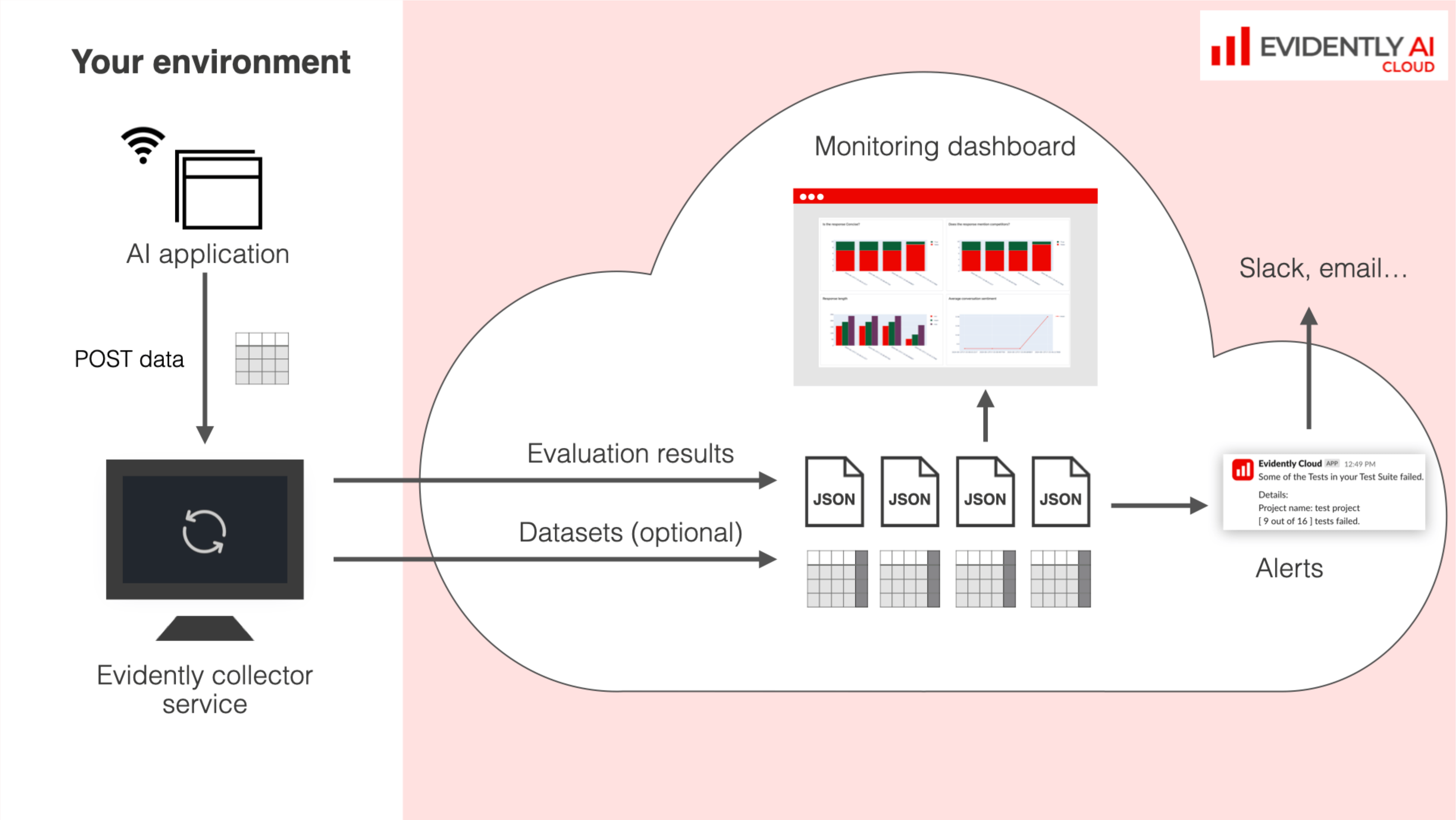
Evidently Data

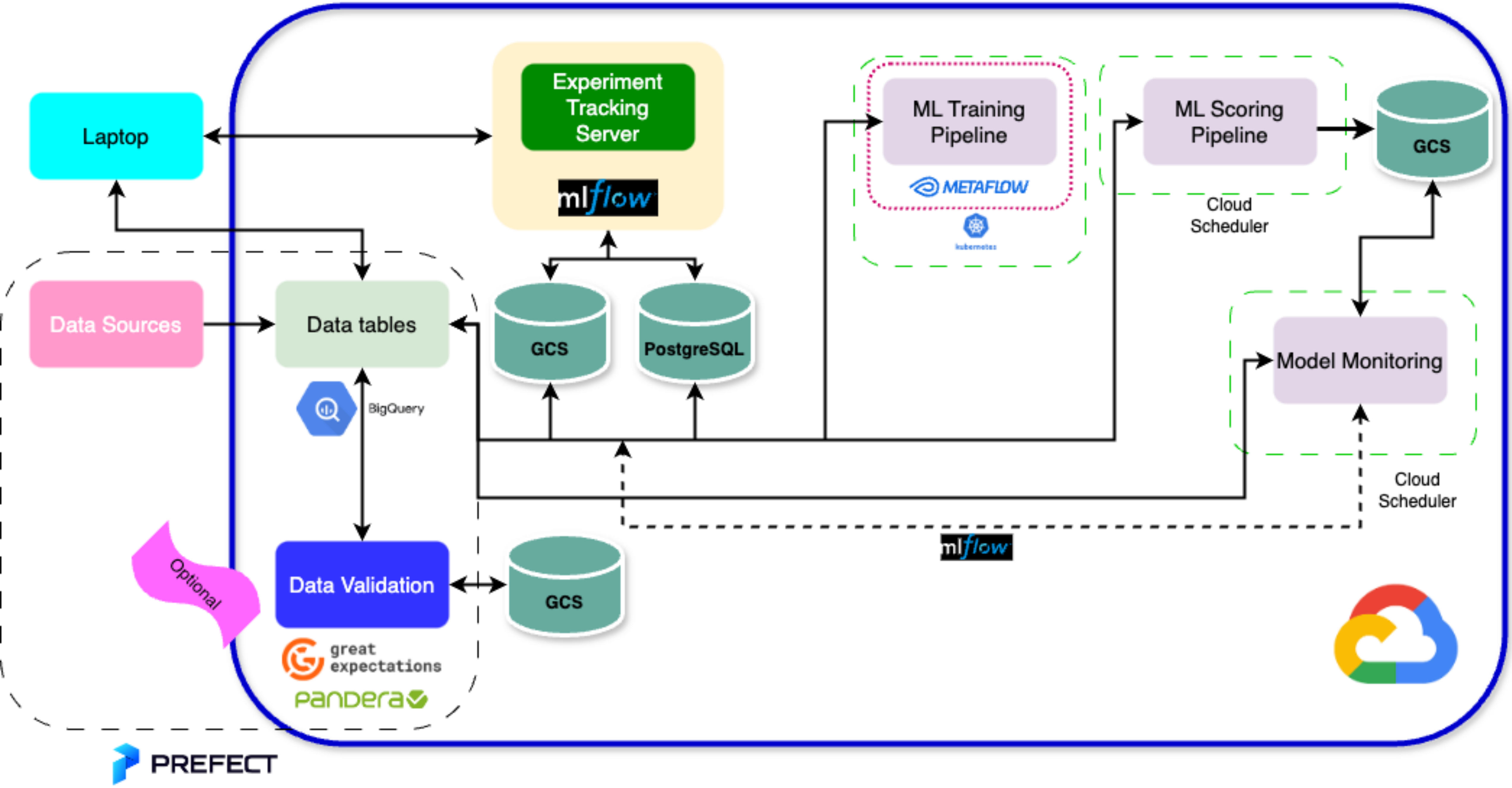
- **id**: ID column
- **datetime**: treated as a datatime column
- **prediction**: predictions column
- **target**: labels column
- None numeric or datetime columns: treated as categorical
- Can map numeric, categorical, text, and embeddings using `ColumnMapping()`

Evidently Batch Workflow



Evidently Near Real-time Workflow





Laptop
Experiment Tracking Server
mlflow
ML Training Pipeline
METAFLOW
kubernetes
ML Scoring Pipeline
Cloud Scheduler
GCS
Data Sources
Data tables
GCS
PostgreSQL
BigQuery
Model Monitoring
Cloud Scheduler
mlflow
Optional
Data Validation
GCS
great expectations
pandera
PREFECT

When Models Fail

Model Retraining

- When performance degrades
  - Manually retrain model with fresh data
  - Consider **continual retraining?**
  - In some cases, a completely new model built from scratch is necessary (new features, new metrics, fresh perspective)

Retrain based on:
- Time (e.g. every two weeks)
- Performance (e.g. AUC degrades by 10%)
- Drift detection

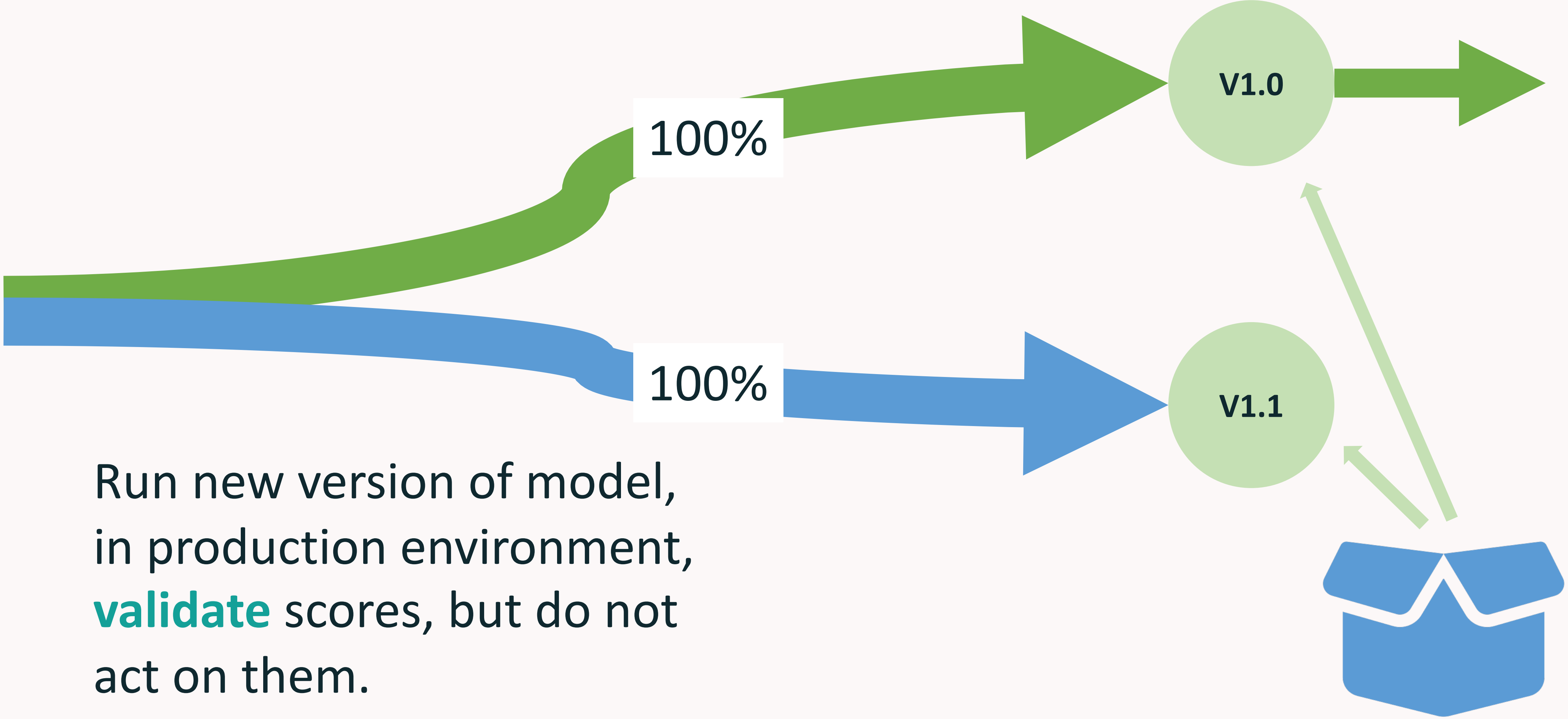
Model Retraining

- When performance degrades
  - Manually retrain model with fresh data
  - Consider **continual retraining?**
  - In some cases, a completely new model built from scratch is necessary (new features, new metrics, fresh perspective)

NOT COMMON AS FAR AS I CAN TELL

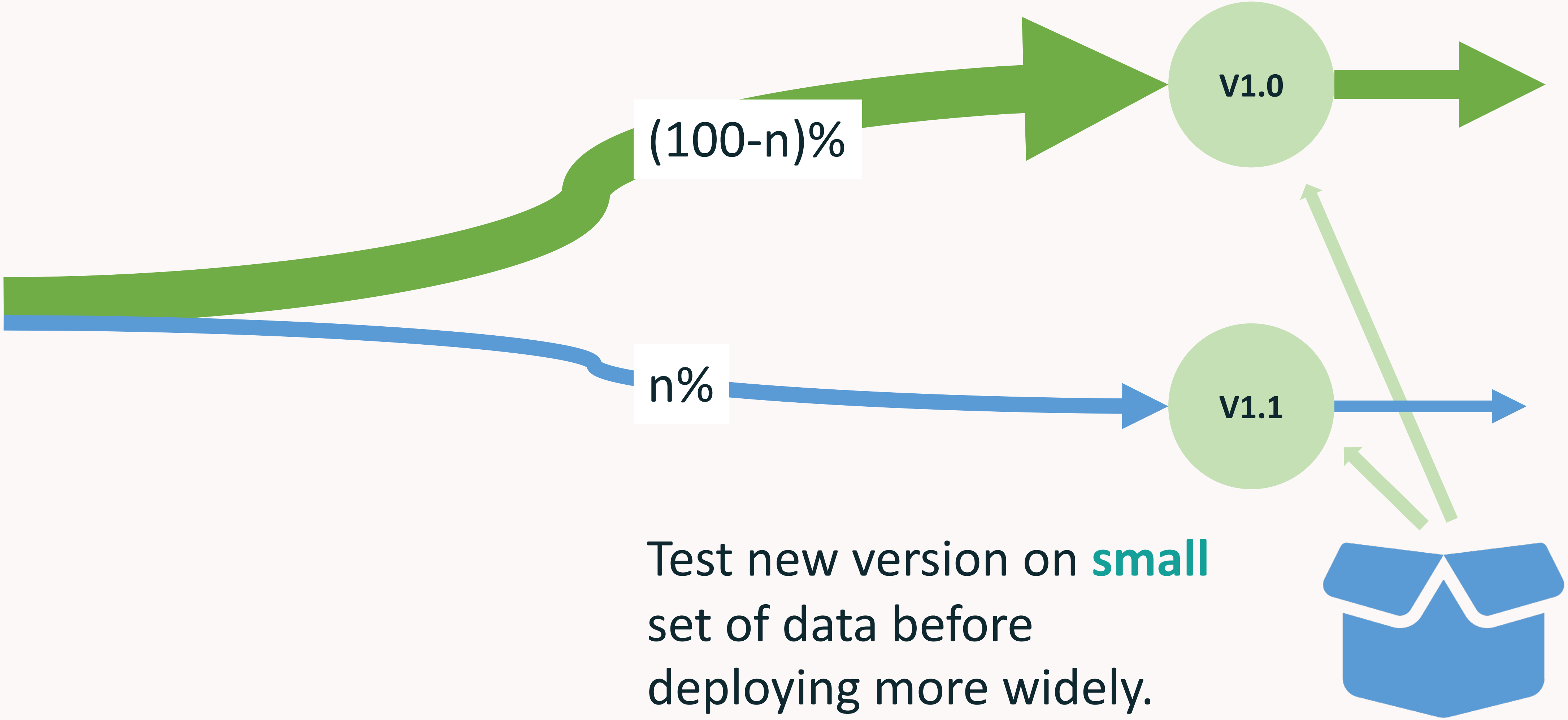
Retrain based on:
- Time (e.g. every two weeks)
- Performance (e.g. AUC degrades by 10%)
- Drift detection

Shadow Deployments



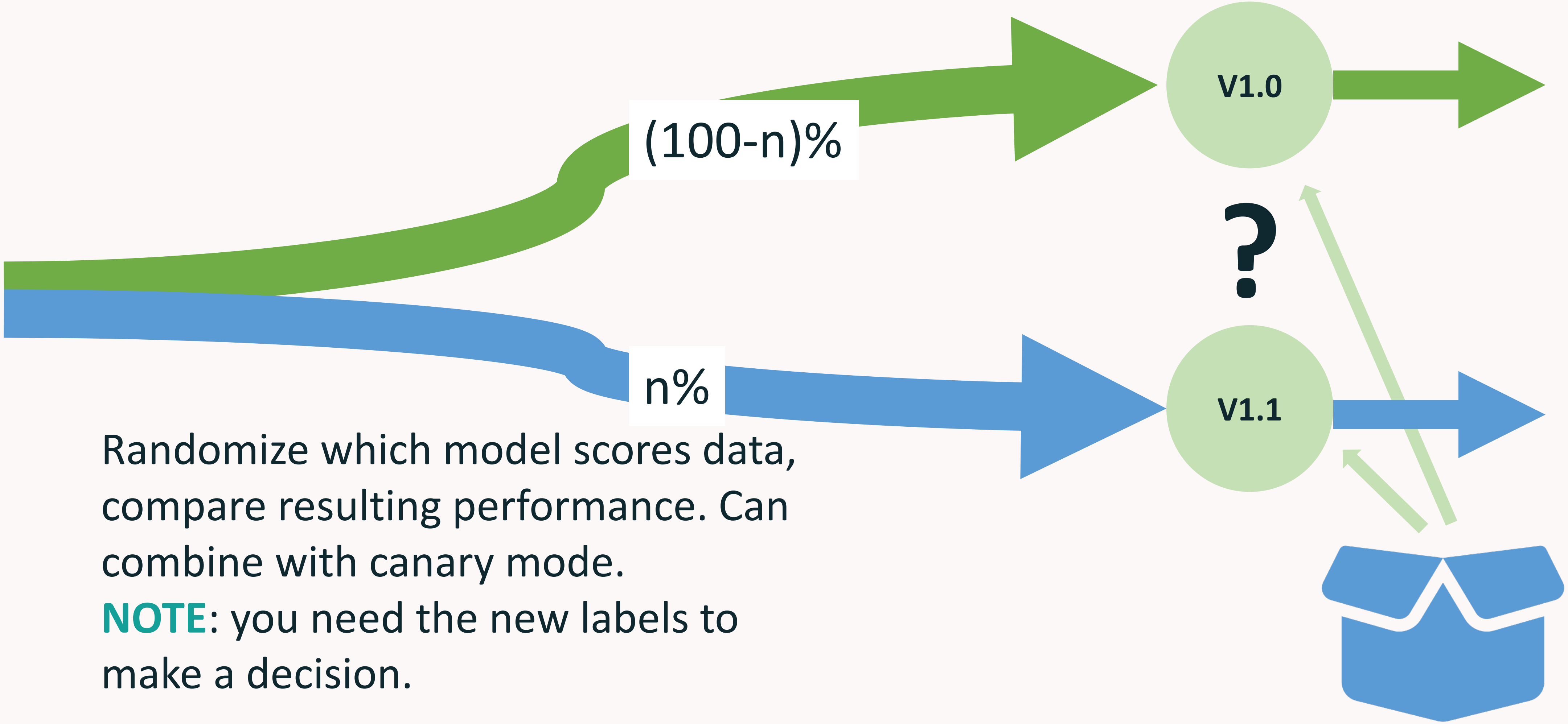
Run new version of model, in production environment, **validate** scores, but do not act on them.

Canary Deployments



Test new version on **small** set of data before deploying more widely.

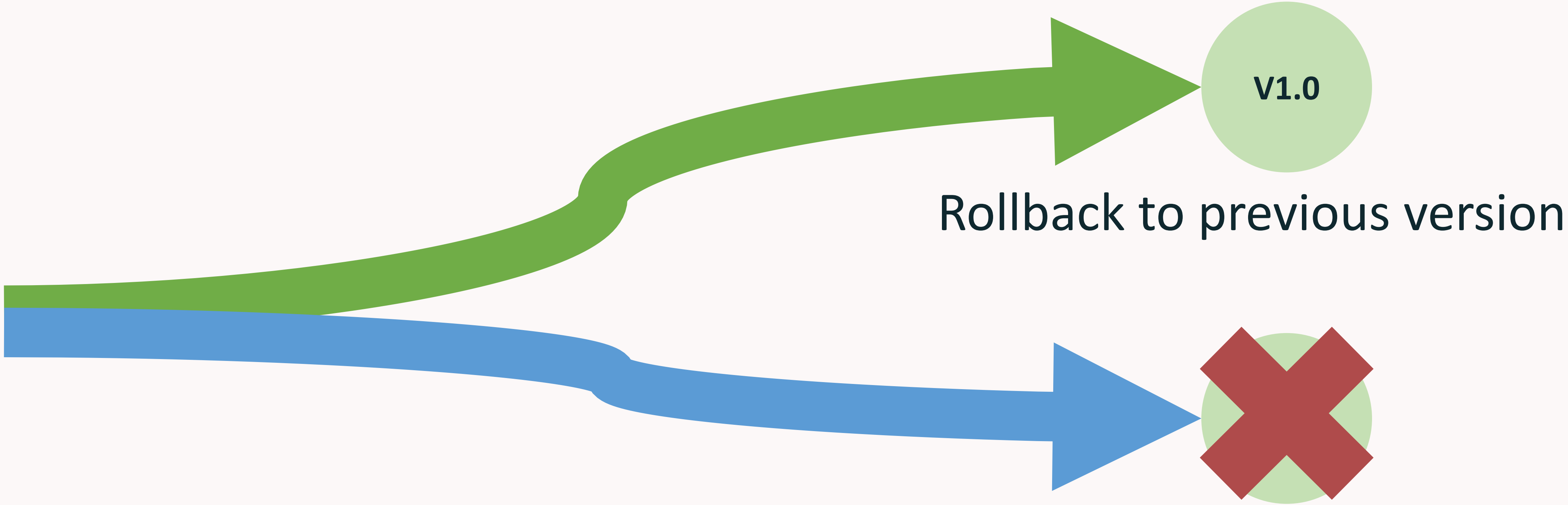
A/B Testing



Randomize which model scores data, compare resulting performance. Can combine with canary mode.

NOTE: you need the new labels to make a decision.

When New Models Fail



Model Monitoring Demo

Model Monitoring Lab

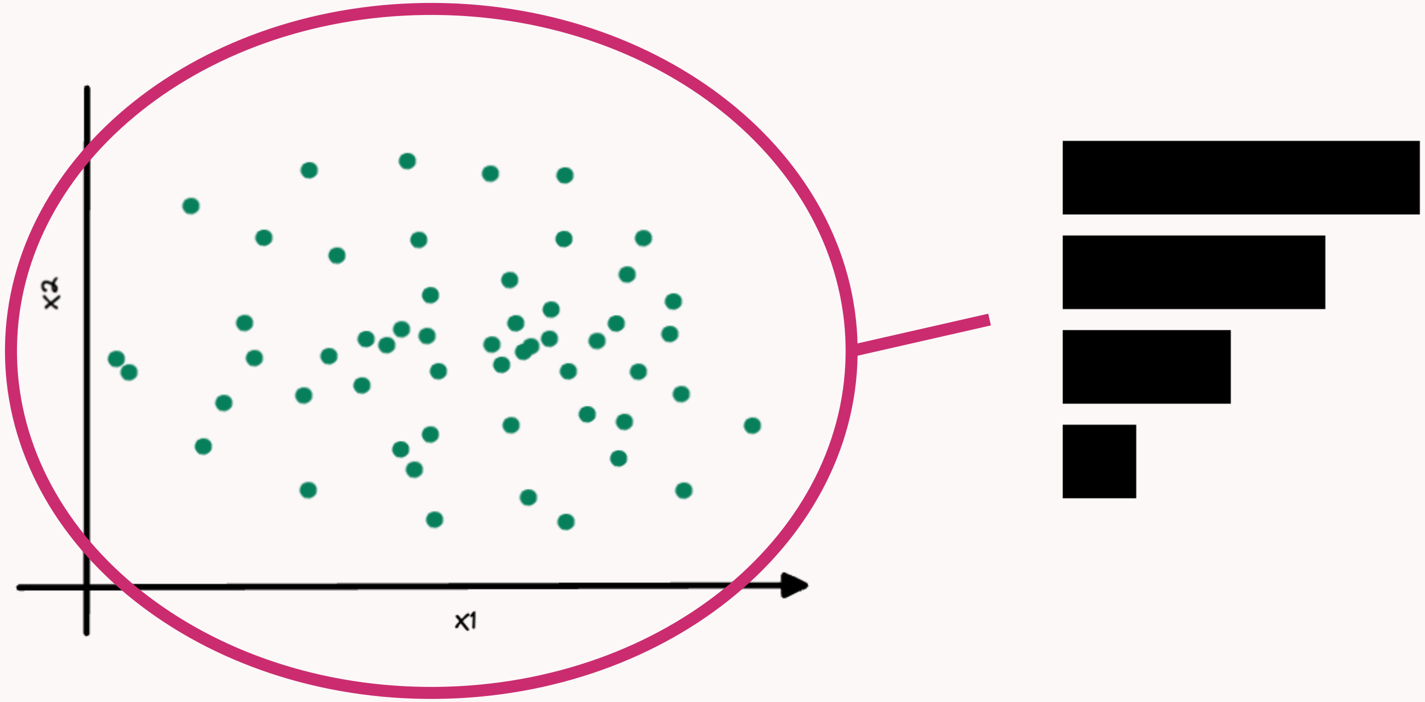
Model Explainability

Why Do We Need Explainability (XAI)?

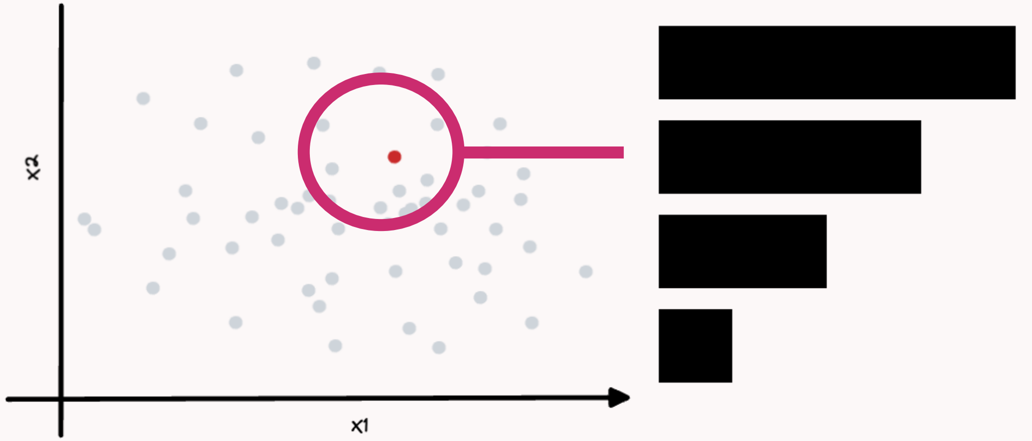
- Regulators may ask you to explain how your model works or how it made a specific prediction
  - Healthcare
  - Finance/Credit
- Customers may ask you to explain the same
  - People don’t trust what they don’t understand
- Identifying bias

Approaches

- Global importance
  - Coefficients
  - Permutation importance
  - Partial dependency plots



- Local importance
  - Local Interpretable Model-agnostic Explanations (LIME)
  - SHapley Additive exPlanations (SHAP)



Considerations

- Neither of these methods explains causality
- Neither of these methods is fool-proof
- They can be computationally expensive, so how should we implement them as part of our monitoring system?
    - Create explanations on-the-fly when needed?
    - Create and store explanations for all predictions made?

LIME

- Read the original paper here
- Can use the lime python library
- Pros:
  - Easy to use
  - Can explain any black-box model, including models for images and text
  - How well the surrogate model fits can tell you how useful explanations are
- Cons:
  - Choice of kernel affects explanations, sometimes dramatically
  - Choice of sample affects explanations, sometimes dramatically

SHAP

- Read the original paper here
- Can use the shap python library
- Pros:
  - Foundations in game theory
  - Specific "fast" implementations for tree-based models and DL
  - Can be local and global
- Cons:
  - Very computationally slow, specifically for kernel SHAP
  - Can still yield misleading results