Propose a model monitoring pipeline and describe how you would track model drift in 500 words.

Section 1: What is model monitoring?

Figure 1 shows the typical Machine Learning (ML) model lifecycle and where model monitoring can come in.

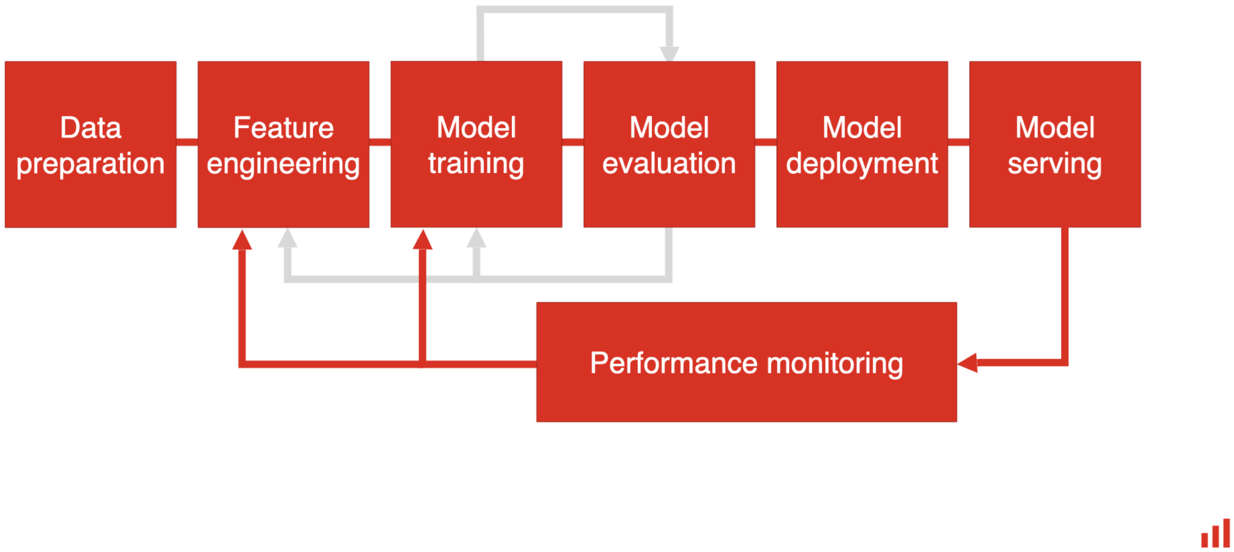

Figure 1: ML model lifecycle and model monitoring (Source: https://www.evidentlyai.com/ml-in-production/model-monitoring#model-monitoring-architectures)

Model monitoring involves the continuous, automated observation of a ML system and its inputs, outputs, operational health and actual performance to detect model drift, training-serving skew and data pipeline problems. These observations are used to calculate metrics for reporting to visualization tools for remediation. Practically, the model monitoring pipeline can be split into a real-time loop for quick remediation and a batch processing loop for actual performance given ground-truths and for driving model retraining/recalibration.

Section 2: Proposed model monitoring pipeline

The real-time monitoring loop usually does not have ground-truths yet.

1. Decide on settings and tune iteratively.
    1. Baselines – from the training distribution or the first stable week (informed by batch loop).
    2. Time intervals – e.g. 15 to 60-minute intervals.
    3. Evaluation metrics – Examples: PSI, KS statistic, JS divergence, Wasserstein distance.[1]
    4. Thresholds – help to decide when alerts are triggered; set judiciously to prevent alert fatigue.
2. Measure data/feature drift and prediction drift – see ***Section 3***.
3. Monitor pipeline or data-quality issues via automated workflow managers like Airflow.
    1. Monitor drops in quantities of successful predictions over time against baseline.
    2. Use of data validation tests against processing pipelines to verify data quality.

---

1 Population Stability Index (PSI), Kolmogorov-Smirnov (KS) statistic, Jensen-Shannon (JS) divergence, and Wasserstein distance are all metrics used for drift.

3. Check that pipelines are updated through context from database schema changes, audit logs.
4. Store above in a stream (Kafka), timeseries (Datadog) and/or archive (S3) for back-testing when ground-truths become available.
5. Alerts triggered based on thresholds, and contains rich event information for effective remediation.

The batch monitoring loop runs on archived predictions once ground-truths are available.

1. Settings
    1. Join keys – how predictions mapped to ground-truths.
    2. Label latency – SLAs for ground-truth arrival.
    3. Time intervals – daily/weekly aggregates can help to account for seasonality effects.
    4. Slices – by geography, customer or data source to surface localized issues.
    5. Also, evaluation metrics and thresholds.
2. Join prediction with labels and compute back-filled performance overall and by slice.
3. Detect concept drift – see ***Section 3***.
4. Run back-tests and correlate with business impact.
    1. Evaluate historical windows with 7 to 15-day rollups to smooth seasonality.
    2. Correlate model metrics with downstream KPIs; capture lead/lag relationships where relevant.
5. Store joined datasets, reports and metrics in a warehouse/lake for traceability and analysis
6. Remediation and continuous learning
    1. If concept drift: retrain models, recalibrate thresholds, or adjust features.
    2. Safe rollout: shadow/canary the challenger; promote only if SLOs hold, else rollback.
    3. Update baselines after a new stable regime.

Section 3: Tracking model drift – 3 categories

1. **Data/feature drift:** compare recent ***input*** distributions to a baseline (training or a first stable prod week; consider multiple/moving, time-aligned baselines for seasonality) using relevant metrics[2], then run per-feature and in aggregate; add simple rule checks and monitor feature-attribution drift[3] to explain changing feature reliance.
2. **Prediction drift:** apply the above tests to ***score/class*** distributions and shape proxies[4], and pair with calibration proxies[5] to reduce false positives on feedback delay.
3. **Concept drift:** when labels arrive, backfill and evaluate[6], then calibrate[7] by slice against baselines; sustained degradation indicates the ***input-to-label relationship*** has changed and should trigger remediation and a baseline refresh.

---

2 PSI, KS, JS, Wasserstein and Chi-square (used for categorical data)

3 SHAP is a metric that can be used and explains a model's prediction by fairly splitting the "credit" among input features (via Shapley values). This makes feature attribution easy: you see how much each feature pushed a prediction up or down for one case, and you can aggregate these contributions to learn which features the model relies on overall.

4 Shape proxies are summary statistics to represent the output distribution, like mean, variance, entropy and percentile bin counts.

5 Calibration proxies are quick, label-free checks (like shifts in average confidence, entropy, or score histograms) that tell you if your model's predicted probabilities are getting unusually "sure" or "unsure."

6 Relevant evaluation metrics here – AUC/PR-AUC/F1 (classification) vs RMSE/MAE (regression).

7 ECE/Brier (calibration for probabilistic classifiers): ECE is the average gap between predicted probabilities and actual frequencies across bins (0 = perfectly calibrated), while the Brier score is the MSE of predicted probabilities vs. outcomes (lower is better) and penalizes both miscalibration and overall inaccuracy.