WHITE PAPER

The ML Data Processing Maturity Model

Assessing and Advancing Your Organisation's Data Pipeline Capabilities

Published by Code-B | 2026 Edition

code-b.dev | ML Engineering & Data Infrastructure

Executive Summary

Most organisations that invest in machine learning are investing in the wrong place. They hire data scientists, choose model architectures, and benchmark algorithms, while the foundational layer that determines whether any of it works in production, the data processing pipeline, receives a fraction of the attention and almost none of the rigour.

The result is predictable and well-documented: models that perform beautifully in a notebook degrade silently in production, ML projects that cannot be reproduced six months after the data scientist who built them left the team, and data quality failures that manifest as model failures, with no clear way to diagnose which is which.

This white paper introduces the ML Data Processing Maturity Model: a structured framework for assessing where your organisation's data pipeline capabilities currently stand, understanding what that level means for your business risk, and identifying the specific investments needed to reach the next level.

The model defines five maturity levels, from Level 1 (Ad Hoc) to Level 5 (Optimised), each describing a distinct state of data processing capability with observable characteristics, business implications, and a clear transition pathway. A practical self-assessment questionnaire, included in full, allows any technical leader or ML practitioner to score their organisation across five capability domains and receive a concrete maturity profile.

The core finding of this framework, validated across engagements with dozens of engineering teams, is this: organisations rarely fail at machine learning because of their models. They fail at the data infrastructure that feeds them. Advancing your data processing maturity is the highest-leverage investment most ML teams can make in 2026.

What You Will Learn in This White Paper

- The five levels of ML data processing maturity and what each looks like in practice
- A 25-question self-assessment to score your organisation across five capability domains
- The specific transitions required to move from one maturity level to the next
- Industry benchmarks: where fintech, healthtech, e-commerce, and manufacturing typically land
- An anonymised case study: moving from Level 2 to Level 4 in six months

Section 1: Why Most Organisations Overestimate Their Data Pipeline Maturity

The Infrastructure No One Talks About

There is a well-known pattern in machine learning project failures. A team of talented data scientists trains a model that achieves excellent performance on a held-out test set. The model is celebrated internally, presented to leadership, and moved toward production deployment. Then something goes wrong. The model does not perform as expected on real production data. Predictions are inconsistent. The team cannot reproduce the original results. A bug is found in the preprocessing logic. Or, worst of all, the model appears to work but delivers quietly degraded predictions for months before anyone notices.

In the majority of these cases, the failure is not the model. It is the data processing infrastructure underneath it.

Data processing, the collection of decisions, transformations, and engineering practices that determine what data enters a model, in what form, and with what consistency, is the foundational layer of every ML system. It is also the layer that receives the least systematic attention. Model architecture is debated and benchmarked. Feature engineering is iterated on carefully. But the preprocessing pipeline is often assembled quickly, documented poorly, and validated almost not at all.

The Gap Between Perceived and Actual Maturity

When engineering teams are asked to describe their data processing pipeline, the language is often confident: 'We use Scikit-learn Pipelines.' 'We have a Spark cluster.'We run Great Expectations on our ingestion layer.' These statements may all be true, and they tell you almost nothing about whether the pipeline is actually fit for production ML.

The presence of a tool is not evidence of mature usage. A Scikit-learn pipeline that fits the scaler before the train/test split is not safer than a pipeline with no framework at all, it is specifically dangerous because it appears correct. A Spark cluster that processes data without schema validation on ingestion will faithfully distribute corrupt data across every worker node at scale. Great Expectations suites that were set up once and never updated will pass on data that has quietly shifted in distribution.

Maturity in data processing is not measured by the tools used. It is measured by the decisions made, the tests written, the monitoring configured, and the practices embedded in how the team works day to day.

What It Costs to Get This Wrong

The business consequences of immature data processing are real and quantifiable, even if they are rarely attributed correctly to their root cause:

Failure Mode	How It Appears	Real Root Cause
Model performance degrades after launch	Blamed on model staleness or the algorithm	Input feature distribution shifted; no monitoring was in place
Model performs well in testing, fails in production	Blamed on insufficient training data	Training-serving skew different preprocessing in training vs deployment
The ML project cannot be reproduced or handed over	Blamed on team knowledge loss / attrition	No versioning of data, code, or pipeline configuration
The model produces discriminatory or biased outputs	Blamed on training data bias	Encoding choices or imputation strategies introduced bias during preprocessing
Retraining a model takes weeks	Blamed on compute costs or team capacity	The preprocessing pipeline is manual, undocumented, and not automated

None of the failure modes above requires an incompetent team to occur. They are structural consequences of building ML systems on immature data infrastructure — and they happen routinely in organisations that consider themselves sophisticated ML practitioners.

> **The Core Thesis**
>
> Organisations rarely fail at machine learning because of their models. They fail at the data infrastructure that feeds them. The ML Data Processing Maturity Model exists to make that infrastructure legible, assessable, and improvable in a structured way.

Section 2: The Five Levels of ML Data Processing Maturity

The ML Data Processing Maturity Model defines five discrete levels. Each level represents a qualitatively distinct state of capability, not a point on a linear scale, but a coherent description of

how an organisation processes data for machine learning. The levels are cumulative: each one builds on the foundations of the previous, and skipping levels leads predictably to failure.

For each level, this section describes the following: what the pipeline looks like in practice, the observable symptoms and indicators at that level, the business risks it creates, and the specific gaps that must be addressed to advance.

Level	Name	Core Characteristic	Typical Team	Primary Risk
1	Ad Hoc	Manual, notebook-based, unrepeatable	Early-stage / single data scientist	Cannot reproduce the result
2	Scripted	Code in scripts, no tests, no versioning	Small team, first ML project	Breaks when data changes
3	Structured	Proper pipelines, no leakage, version-controlled	Growing ML team	No visibility into production behaviour
4	Monitored	Full MLOps, drift detection, automated alerts	Mature ML team	High infrastructure complexity
5	Optimised	Feature stores: real-time, auditable, self-healing	Enterprise ML platform team	Maintenance cost at scale

Level 1 - Ad Hoc: The Notebook Era

At Level 1, data processing happens in Jupyter notebooks, is different every time, and leaves no durable record of what was done. The data scientist who built the pipeline is the only person who can run it, because it exists primarily in their local environment and their memory.

What It Looks Like

- Preprocessing is performed interactively in Jupyter notebooks, with cells executed in an order that cannot be guaranteed
- Cleaning decisions, such as, which outliers to remove, how to handle missing

values, and what to do with unexpected categories, are made ad hoc and undocumented
- There is no separation between exploratory analysis and production preprocessing; the same notebook does both
- Data is stored in local CSV files or shared folders, with no version tracking
- Reproducing a model requires the original data file, the original notebook, and the original environment, and typically still requires the original author

Observable Business Symptoms

- We cannot retrain the model without [person X] they know how the data was cleaned'
- Model retraining takes a week because the preprocessing has to be redone each time manually
- Different team members produce different predictions from the same raw data
- A model's test performance cannot be reproduced three months after it was built

Business Risk at This Level

Level 1 organisations have no durable ML asset. They have a result, a set of predictions and a trained model, but not the infrastructure to maintain, update, or audit it. Every model is a one-time artefact. This is acceptable for pure research or proof-of-concept work; it is unacceptable for any ML system that needs to be maintained in production.

> **Key Gap to Address at Level 1**
>
> The primary gap is reproducibility. Before improving any other aspect of data processing, teams must move preprocessing code out of notebooks and into versioned, parameterised scripts. The goal is not perfect code, it is code that someone other than the original author can run and get the same result.

Level 2 - Scripted: Code Without Safety Nets

At Level 2, the notebook has been converted into Python scripts. The preprocessing steps are encoded, somewhat repeatable, and can in principle be run by someone other than the original author. However, the code has no tests, the data has no versioning, and there is no separation of concerns between the preprocessing that is applied during training and what will need to happen in production.

What It Looks Like

- Preprocessing exists as Python scripts (often a single preprocessing.py or data_cleaning.py)
- Scripts can be run end-to-end, but they have no unit tests and no input validation

- Scalers and encoders are fit without clear discipline about what data they see; data leakage is common and often undetected
- The train/test split is sometimes performed after transformation, meaning test data statistics influence training preprocessing
- No feature store or pipeline object; the model and its preprocessing steps are not saved together
- Data is in a shared location but not versioned; nobody knows which version of the data produced which model

Observable Business Symptoms

- A schema change in the upstream data source (a new column name or a new category value) causes the preprocessing script to fail silently or with a cryptic error
- Model performance looks excellent in testing, but unexpectedly degrades in production, and the team cannot pinpoint why
- Retraining produces a slightly different model each time, even with the same data
- New team members need weeks of onboarding to understand the preprocessing logic because it is not documented

Business Risk at This Level

Level 2 is the most dangerous maturity level because it creates an illusion of engineering rigour. The code exists, it runs, and it produces results. But without tests, data versioning, or leakage prevention, the results are unreliable and the pipeline is brittle. Many organisations deploy ML models built on Level 2 pipelines and discover the problems only after those models have been in production for months.

> **Key Gap to Address at Level 2**
>
> The two most critical gaps are data leakage prevention (the scaler must be fit on training data only and applied consistently to new data) and saving the preprocessing pipeline as a fitted object alongside the model. Both can be addressed in a single week by adopting Scikit-learn pipelines or equivalents. Without these two changes, no amount of model tuning will produce reliable production performance.

Level 3 - Structured: Engineering Discipline Applied

Level 3 represents the first maturity level at which an organisation has real engineering discipline around data processing. Pipelines are built using frameworks designed for ML reproducibility. The train/test split happens before any transformations are applied. Preprocessing steps are saved as fitted objects together with the model. Code is version-controlled and, in many cases, begins to be tested.

What It Looks Like

- Scikit-learn Pipeline or equivalent is used to chain preprocessing steps, preventing data leakage between fitting and transformation
- Scalers, encoders, and imputers are fit exclusively on training data and serialised alongside the trained model
- Code is in version control (Git); the team can check out any historical commit and reproduce the preprocessing logic from that point in time
- Data is versioned using a tool such as DVC or is stored in a data warehouse with clear snapshotting
- Missing value strategies are deliberate and documented per column.
- Class imbalance is identified and addressed with a documented strategy
- Some unit tests exist for the most critical transformation functions, though coverage is incomplete

What Is Still Missing

- There is no monitoring of input data quality or feature distributions in production; the team has no visibility into whether the data the model receives in production looks like the data it was trained on
- There are no automated alerts if preprocessing fails or if data quality degrades
- Retraining is still a largely manual process that requires human initiation and oversight
- Documentation exists in code, but not as a systematic pipeline specification

Business Risk at This Level

Level 3 pipelines perform well at launch and degrade quietly over time. The model was built correctly on well-understood data. But the world changes, user behaviour shifts, upstream data systems change their formats, and a new data source is added with subtly different semantics. Because there is no monitoring, these changes are invisible until they manifest as model performance problems. The delay between a data distribution shift and detection can be months.

> **Key Gap to Address at Level 3**
>
> The critical gap is observability. A Level 3 team knows what their data looked like when they trained the model; they do not know what it looks like today. Adding input data validation (Great Expectations or Evidently AI) and basic feature distribution monitoring closes this gap and is the primary qualification for Level 4.

Level 4 - Monitored: Full MLOps Discipline

At Level 4, data processing is an engineered, observable, and maintainable system. Input data is validated before it reaches the preprocessing pipeline. Feature distributions are monitored

against training baselines. Alerts fire when distributions drift beyond defined thresholds. Retraining is partially or fully automated. The pipeline is tested, and failures produce clear, actionable error messages rather than silent incorrect outputs.

What It Looks Like

- Data validation gates are implemented at pipeline entry: incoming data is checked against a defined schema (column names, types, allowed value ranges, expected missingness rates) before any transformation is applied
- Feature distribution monitoring compares the distribution of each input feature at inference time against the baseline established during training; a standard deviation or Jensen-Shannon divergence threshold triggers an alert
- Model prediction distribution is also monitored; shifts in output distribution often signal input data problems before performance degradation is measurable
- Pipeline code has comprehensive test coverage: unit tests for transformation functions, integration tests for the full pipeline on representative data, and regression tests that prevent re-introducing previously fixed bugs
- Retraining is triggered automatically (by a drift alert, a schedule, or both) and executes with minimal human intervention
- The full pipeline, preprocessing steps, fitted transformers, model weights, evaluation metrics, and the data version used for training, are logged in an experiment tracking system such as MLflow or Weights and Biases
- Documentation is machine-readable: the pipeline configuration is a versioned artefact, not a narrative document

What Is Still Missing

- Training-serving consistency is still not guaranteed at the infrastructure level. The preprocessing applied during training may differ from what runs in production if the two environments diverge
- Real-time feature computation at low latency (milliseconds) is not supported; the pipeline is designed for batch or near-real-time use cases
- There is no centralised feature registry; features computed for one model are not reusable by other models without duplicating the computation logic

Business Risk at This Level

Level 4 is where most mature ML teams land and where many choose to stop. For the majority of business use cases, daily or weekly model runs, batch scoring, and non-latency-critical predictions – Level 4 is sufficient and appropriate. The risk is not immediate model failure; it is that the infrastructure costs of Level 4 (testing infrastructure, monitoring tooling, and alert management) can become significant without the organisational discipline to maintain them systematically.

Key Gap to Address at Level 4

The gaps that separate Level 4 from Level 5 are infrastructure-level training-serving consistency (a feature store) and real-time pipeline capability. Teams should only invest in closing these gaps if their business use cases genuinely require them: fraud detection, real-time personalisation, dynamic pricing, and live recommendation systems all require Level 5; weekly churn prediction models do not.

Level 5 - Optimised: Data Processing as a Competitive Asset

Level 5 is characterised by the industrialisation of data processing. The pipeline is not a one-time engineering project, it is a continuously maintained, actively monitored, and strategically governed business asset. Feature computation is centralised and reusable across models. Training and serving environments are guaranteed to produce identical preprocessing outputs. Real-time pipelines support millisecond-latency use cases. Processing decisions are auditable and explainable.

What It Looks Like

- A feature store (Feast, Tecton, Vertex AI Feature Store) serves as the single source of truth for feature computation, guaranteeing that features used during training are identical to those served at inference time, eliminating training-serving skew at the infrastructure level
- Streaming pipelines (Apache Kafka, Apache Flink) compute features in real time as events arrive, enabling low-latency model serving for fraud detection, personalisation, and dynamic pricing
- All preprocessing decisions, including why each transformation was chosen, when it was last validated, and what data it was calibrated on, are documented in a machine-readable format and reviewable by compliance and governance teams
- Synthetic data generation (CTGAN, diffusion models) is used strategically to address class imbalance and data scarcity without compromising privacy
- Federated processing is implemented where data cannot be centralised due to privacy regulations; preprocessing happens at the data source, and only weight updates or aggregated statistics are shared centrally
- The pipeline has a defined SLA: ingestion latency, processing throughput, and data freshness at the feature store are all contractual targets with automated breach detection
- AutoML preprocessing tools are used deliberately for experimentation and baseline-setting, with human-reviewed outputs before production promotion

Observable Business Characteristics

- New ML models can be built and deployed faster because feature computation is reusable across projects
- Model performance is predictable and stable because the data it receives in production is guaranteed to match the training data processing
- Regulatory audits of AI systems can be satisfied with documented preprocessing logs, not retrospective reconstruction
- The team can support multiple concurrent ML products without exponential growth in data engineering headcount

A Realistic Note on Level 5

Level 5 is the right target for organisations with high-volume, real-time ML use cases and regulatory accountability requirements. It is not the right target for every team. A 10-person fintech that runs a single fraud model does not need a feature store. A SaaS company scoring customer health weekly does not need Kafka. Level 5 infrastructure carries real maintenance costs and organisational complexity. The goal is not to reach Level 5 as a badge of sophistication, it is to reach the level that your business use cases and risk profile require and to maintain it with discipline.

Section 3: Self-Assessment Questionnaire

The following 25 questions assess your organisation's data processing maturity across five domains. For each question, select the answer that most accurately describes your current state, not your intended state or your best-case scenario. Honest assessment is the only useful input.

Score each answer: A = 1 point, B = 2 points, C = 3 points, D = 4 points, E = 5 points. Sum your scores per domain. Domain scores and the interpretation guide follow the questionnaire.

Domain 1: Data Quality and Validation (Questions 1–5)

Q1. How is incoming data quality verified before it reaches your preprocessing pipeline?

A) It is not verified. We trust that upstream sources are correct.

B) A team member manually checks a sample of the data before each run.

C) Basic assertions are hardcoded in the script (e.g., checking if row count or column names exist).

D) A data validation framework (e.g., Great Expectations, Soda) defines and enforces expectations automatically.

E) Automated schema and statistical validation gates run at ingestion; failures stop the pipeline and trigger alerts.

Q2. When an upstream data source changes its schema (a column is renamed or a new value appears in a categorical feature), what happens?

A) We find out when the model starts producing errors or wrong predictions — sometimes weeks later.

B) The script crashes with an unhelpful error message, and a team member investigates.

C) The script fails with a descriptive error that identifies the problem, but the response is still manual.

D) An automated alert fires and the pipeline halts, preventing corrupt data from reaching the model.

E) Schema changes are detected at ingestion, and the pipeline either self-heals (for expected changes) or alerts with full diagnostic context.

Q3. How do you handle missing values in your training data?

A) We drop all rows with any missing value or fill everything with zero.

B) We use mean or median imputation globally, applied the same way to all columns.

C) We choose imputation strategies per-column based on data type and distribution, but the strategy is not systematically documented.

D) Imputation strategies are chosen per-column, documented, and implemented consistently in a pipeline object that applies the same strategy at inference time.

E) Imputation strategies are chosen per-column with documented reasoning, tested for impact on model performance, and monitored for assumption validity in production data.

Q4. How do you detect and handle outliers in your features?

A) We do not explicitly handle outliers.

B) We remove rows where values fall outside a fixed threshold, decided informally.

C) We use IQR or z-score methods to identify outliers and apply a documented strategy (clip, remove, or flag) per feature.

D) Outlier handling is implemented in a versioned pipeline, applied consistently to training and inference data.

E) Outlier handling is adaptive: thresholds are calibrated on training data, monitored in production, and reviewed when distributions shift.

Q5. How do you document the provenance of your training data (which version of which source, extracted at what time)?

A) We do not document data provenance systematically.

B) We note the date and source informally in a README or comment.

C) Data is stored with a timestamp and source label, but this is not tied to specific model versions.

D) Data versions are tracked with a tool (DVC, Delta Lake, data warehouse snapshots) and each model training run records which data version it used.

E) Complete data lineage is tracked from source to model: every transformation, join, filter, and version is logged and queryable.

Domain 2: Preprocessing Code Quality (Questions 6–10)

Q6. Where does your preprocessing code live?

I. Primarily in Jupyter notebooks, with cells executed in an ad hoc order.
II. In Python scripts, but not systematically organised or version-controlled.
III. In version-controlled Python scripts or modules, runnable end-to-end.
IV. In a framework-based pipeline (Scikit-learn Pipeline, Spark ML Pipeline, or equivalent) that is version-controlled and tested.
V. In a modular, tested, documented pipeline with defined interfaces, treatable as a software product, not a script.

Q7. What happens to your preprocessing code when it is passed to a new team member?

I. The new team member cannot run it without significant help from the original author.
II. They can eventually run it, but it requires hours of environment setup and debugging.
III. They can run it within an hour with the README instructions, though some edge cases may require clarification.
IV. They can run it immediately: the environment is containerised (Docker), dependencies are pinned, and the README is comprehensive.
V. They can run, modify, test, and deploy it without consulting the original author: the code is self-documenting, and the test suite provides a safety net.

Q8. How is data leakage prevented in your preprocessing pipeline?

I. We are not certain what data leakage is or whether it is present.
II. We are aware of the concept but have not audited our pipeline for it.
III. We perform the train/test split before applying transformations, but this is done by convention rather than enforced by the framework.
IV. A pipeline or ColumnTransformer object ensures that fitting happens on training data only and the fitted state is applied consistently to new data.
V. Leakage prevention is enforced by the pipeline framework, audited at code review, and tested by a specific test case that would catch common leakage patterns.

Q9. What level of test coverage exists for your preprocessing code?

I. No tests exist.
II. Manual spot-checks are run before major runs, but there are no automated tests.
III. Some unit tests exist for the most critical transformation functions, but coverage is incomplete.
IV. Unit tests cover all transformation functions, and integration tests validate the full pipeline on representative data.
V. Full test suite including unit, integration, and regression tests; edge cases (nulls, empty strings, single-value columns, and and out-of-range inputs) are explicitly tested; tests run in CI/CD.

Q10. How are your preprocessing parameters (e.g., scaler means/stds, encoding maps, imputation values) stored and managed?

I. They are recomputed from scratch each time the pipeline runs.
II. They are stored in the script as hardcoded values (which become stale over time).
III. They are saved as separate files (pickle and JSON) alongside the model, but not managed systematically.

IV. They are serialised as part of the Pipeline object and stored with the model in a model registry.
V. They are versioned artefacts in a model registry with full metadata (which training run produced them, which data version they were computed from, and when they were last validated).

Domain 3: Reproducibility and Versioning (Questions 11–15)

Q11. Can you reproduce any model your team has trained in the last 12 months, bit-for-bit?

1. No. Key preprocessing steps, data versions, or environment configurations have not been preserved.
2. Probably, but only with significant effort to reconstruct the original environment.
3. Yes, if we have the original data file and can recreate the environment from the requirements file.
4. Yes. Data version, code version, environment, random seeds, and hyperparameters are all logged per run.
5. Yes. We can reproduce any run in under 30 minutes using our experiment tracking system, with automated environment reconstruction.

Q12. How are random seeds managed in your preprocessing and model training?

1. They are not managed; results may vary between runs.
2. They are set manually in the script but not logged.
3. They are set and logged, but not systematically managed across all sources of randomness (train/test split, data shuffling, model initialisation).
4. All sources of randomness are seeded, and the seed is logged as part of the experiment record.
5. Seeds are managed, logged, and enforced across the entire pipeline, including distributed components.

Q13. When you retrain a model, how long does the end-to-end process take (data to deployed model)?

1. Weeks. The preprocessing has to be redone each time manually.
2. Several days. There are manual steps, but the process is documented.
3. One to two days. The pipeline runs automatically but requires manual initiation and some oversight.
4. Hours. The pipeline is automated end-to-end and runs with a single command or trigger.

5. Under an hour. The pipeline is fully automated, containerised, and optimised for speed.

Q14. How are feature engineering transformations documented?

1) They are not documented; the code is the only record.
2) Documented informally in comments or a README that may be outdated.
3) Documented in a structured format (e.g., a feature catalogue or data dictionary) that is maintained alongside the code.
4) Documented with business definitions, technical transformations, and expected value ranges, documentation is version-controlled with the code.
5) Machine-readable documentation that is automatically generated from the pipeline definition; queryable and referenceable by other teams and tools.

Q15. If your primary data engineer left tomorrow, how long would it take the remaining team to fully understand and operate the data processing pipeline?

1) Months. Critical knowledge is not documented and lives only in that person's head.
2) Several weeks. The code exists but requires extensive reverse-engineering.
3) One to two weeks. The code and documentation are sufficient, but onboarding requires effort.
4) A few days. The code, documentation, tests, and README provide sufficient context.
5) Hours. The pipeline is self-documenting, tested, and designed to be operated by any qualified engineer.

Domain 4: Production Monitoring and Drift Detection (Questions 16–20)

Q16. How do you know when the data entering your deployed model has changed significantly from the data it was trained on?

I. We do not have a systematic way to detect this.
II. We check manually by running summary statistics on a sample of recent production dataperiodically.
III. We track a small set of high-priority features manually and review them monthly.
IV. Automated monitoring compares feature distributions in production against training baselines; a threshold breach triggers an alert.
V. Real-time drift monitoring covers all features, uses statistical tests appropriate to each feature type, and triggers automated investigation workflows.

Q17. How do you monitor the health of your preprocessing pipeline in production?

I. We are notified when users report problems.
II. We check pipeline logs manually and periodically.
III. Basic success/failure logging is in place; failures send an email notification.
IV. Pipeline latency, throughput, data volume, and failure rate are monitored with dashboards and alerting thresholds.
V. A full observability stack (metrics, logging, tracing) covers every stage of the pipeline; anomaly detection fires before user impact is likely.

Q18. When your model's prediction quality degrades in production, how quickly is this detected?

I. It is detected when customers complain or a business metric visibly declines, potentially months later.
II. It is detected during a periodic manual model review, typically monthly or quarterly.
III. It is detected when automated performance metrics (where ground truth is available) fall below a threshold.
IV. It is detected within days, using a combination of performance monitoring and proxy drift metrics.
V. It is detected within hours, using real-time monitoring of input distributions, prediction distributions, and performance metrics; automated triage starts immediately.

Q19. How is model retraining triggered?

I. Manually, when someone decides the model seems stale.
II. On a fixed schedule (e.g., monthly), regardless of whether retraining is needed.
III. On a fixed schedule, with a manual review of whether performance metrics warrant it.
IV. Triggered automatically by performance degradation signals or a data drift alert, in addition to a scheduled fallback.
V. Triggered by a combination of drift signals, performance metrics, and business KPIs, the retraining pipeline executes fully automatically and deploys the new model if validation criteria are met.

Q20. How are production preprocessing failures handled?

I. The pipeline fails silently; incorrect predictions are served without awareness.
II. The pipeline crashes; serving stops until a team member investigates and restarts it.
III. The pipeline fails with an error log; a team member is notified and manually initiates recovery.
IV. Failure triggers an automated alert, the pipeline falls back to a safe state (e.g., the last valid model), and a resolution ticket is created automatically.

V. Failure triggers automated recovery (retry, fallback, and circuit-breaker), with full diagnostic context surfaced to the on-call engineer; SLA tracking records time to resolution.

Domain 5: Team Practices and Governance (Questions 21–25)

Q21. How are data preprocessing decisions reviewed before a model goes to production?

A) They are not formally reviewed; the data scientist's judgment is trusted.

B) The model is reviewed, but the preprocessing is not explicitly part of the review.

C) Preprocessing is reviewed informally by a senior team member before production deployment.

D) A structured code review covers preprocessing logic, with specific checks for leakage, encoding correctness, and documentation.

E) A formal preprocessing review checklist is used; sign-off is required from both a data science and a data engineering reviewer before production deployment.

Q22. How does your team handle imbalanced datasets (where one class is significantly underrepresented)?

A) We do not adjust for class imbalance; we train on the data as-is.

B) We address it by oversampling the minority class using random replication.

C) We use SMOTE or class weights and document the strategy, but do not systematically evaluate its impact.

D) The strategy is chosen based on the specific imbalance severity and model type, documented, and evaluated for impact on precision and recall for the minority class.

E) Multiple strategies are evaluated and compared; the choice is documented with justification; the impact is monitored in production.

Q23. How do you manage preprocessing for privacy-sensitive data fields (PII, health data, and financial data)?

A) Privacy-sensitive fields are not systematically identified or handled differently.

B) We know which fields are sensitive, but handling is ad hoc and person-dependent.

C) Sensitive fields are identified and masked or excluded from training data as a standard practice.

D) Sensitive fields are identified, classified by sensitivity level, and handled according to a defined policy (masking, anonymisation, differential privacy) enforced in the pipeline.

E) Full privacy engineering is embedded in the pipeline: automated PII detection, differential privacy where required, audit logging of all data access, and compliance documentation for GDPR/HIPAA/CCPA requirements.

Q24. How are preprocessing decisions made when a new ML project begins?

A) Each project starts from scratch; there is no organisational memory of what has worked before.

B) The data scientist consults previous projects informally but has no structured way to reuse prior work.

C) A shared code library of common preprocessing functions exists and is consulted, though adoption is not enforced.

D) A documented set of preprocessing standards and a shared feature library are used as the default starting point for every project.

E) A feature store and preprocessing template library provide ready-made, tested, production-grade components; new projects begin by selecting from existing features and extending where needed.

Q25. How does your organisation evaluate whether a preprocessing strategy was the right choice after the model is in production?

A) We do not evaluate preprocessing decisions post-deployment.

B) If the model performs poorly, we may revisit preprocessing as a possible cause.

C) Periodic model reviews include a retrospective on whether preprocessing choices contributed to performance issues.

D) Preprocessing decisions are documented at deployment and reviewed against production metrics during model health reviews.

E) A closed-loop evaluation process systematically attributes production model behaviour to specific preprocessing choices; learnings are incorporated into the team's standards and the feature library.

Interpreting Your Score

Sum your scores for each domain (minimum 5, maximum 25 per domain). Use the table below to interpret each domain score and your overall total.

Score Range	Domain Level	Overall Score	Overall Level	Interpretation
5 – 9	**Level 1**	25 – 45	**Level 1**	Significant foundational work needed before any production ML investment
10 – 14	**Level 2**	46 – 65	**Level 2**	Engineering foundation exists, but safety and reproducibility gaps create real production risk
15 – 18	**Level 3**	66 – 82	**Level 3**	Solid engineering discipline in place; monitoring is the critical next investment
19 – 22	**Level 4**	83 – 98	**Level 4**	Production-grade pipeline; evaluate real-time and feature store needs based on use cases
23 – 25	**Level 5**	99 – 125	**Level 5**	Optimised pipeline; focus on maintaining standards and extending to new use cases efficiently

Note on Domain Imbalances

An organisation may score at different levels across different domains. A team that scores Level 4 on Code Quality but Level 1 on monitoring is at an overall Level 2 because the weakest domain determines operational risk. Pay particular attention to your two lowest-scoring domains; these represent your highest-priority improvement areas.

Section 4: What It Takes to Move Up One Level, A Practical Roadmap

Organisations frequently attempt to skip maturity levels, investing in Level 5 tooling (feature stores, streaming pipelines, AutoML platforms) before establishing Level 3 fundamentals. This approach consistently fails: sophisticated infrastructure built on an immature foundation inherits and amplifies the weaknesses of that foundation. A feature store populated by leaky, inconsistently processed features is worse than no feature store at all.

Each transition described below represents the minimum viable set of changes needed to move from one level to the next. The timelines given are based on observed real-world implementations across teams of two to ten data engineers and data scientists.

Transition: Level 1 to Level 2

Estimated Timeline: 2–4 weeks

Minimum Investment: Data engineer time; no new tooling required

What Must Change

Move all preprocessing logic out of notebooks and into Python scripts that can be executed from the command line with a single command
Establish a Git repository for all ML code; adopt a convention for branching and commit messages
Document the preprocessing steps in a README sufficient for a new team member to understand what each script does and in what order to run them
Establish a shared, accessible location for data (a shared drive, S3 bucket, or database) rather than local files
Identify and document the expected schema of input data: column names, data types, expected ranges, and known edge cases

What to Avoid

Do not invest in a monitoring framework, a feature store, or an AutoML platform until Level 3 is established, the investment will not survive contact with the underlying fragility of the pipeline
Do not reorganise the team or hire before the technical foundations are in place

Transition: Level 2 to Level 3

Estimated Timeline: 3–6 weeks

Minimum Investment: Senior data engineer time; Scikit-learn or equivalent already available

What Must Change

Implement the train/test split as the first operation in the pipeline, before any transformations are applied this single change eliminates the most common source of data leakage.

Replace manual scaler and encoder code with a Scikit-learn Pipeline or ColumnTransformer that wraps all preprocessing steps in a single object

Serialise the fitted Pipeline alongside the trained model; deploying the model without its fitted preprocessing steps is not an option at Level 3 or above

Write unit tests for the three to five most critical transformation functions: the ones where an error would most directly harm model quality or produce incorrect predictions

Adopt DVC or equivalent for data versioning; each model training run should reference a specific, retrievable data snapshot

Document imputation and encoding strategies per-column in a data dictionary

The Single Most Impactful Change at This Transition

Of all the changes above, moving to a Scikit-learn Pipeline with correct train/test split ordering has the highest immediate impact on model reliability. In our experience across client engagements, this change alone when implemented correctly has materially improved deployed model performance in cases where the team believed their performance problems were algorithmic. The true cause was data leakage in the previous pipeline.

Transition: Level 3 to Level 4

Estimated Timeline: 4–8 weeks

Minimum Investment: Data engineering time + monitoring tooling (Evidently AI, Great Expectations, or similar)

What Must Change

Implement a data validation suite that runs automatically at pipeline entry: define expectations for each input feature and fail loudly when those expectations are violated

Establish training baseline statistics for each input feature (mean, standard deviation, quantiles, missing rate, cardinality for categoricals) and store them as part of the model deployment artefact

Implement feature distribution monitoring that compares live production data against those baselines at a regular interval (daily minimum, hourly for high-volume systems)

Define alerting thresholds: for each feature, what level of distributional shift triggers a human review? What level triggers an automatic pipeline halt?

Adopt an experiment tracking platform (MLflow, Weights and Biases, or equivalent) that logs data version, code version, preprocessing configuration, hyperparameters, and evaluation metrics for every training run

Implement comprehensive test coverage: unit tests for all transformation functions, integration tests for the full pipeline, and at least one test specifically designed to catch data leakage

Set up a CI/CD pipeline for the ML code: tests run automatically on every pull request; deployment to production requires passing tests

Common Mistakes at This Transition

Setting monitoring thresholds too tightcauses alert fatigue that leads the team to ignore alerts calibrate thresholds on historical distribution variance, not on theoretical ideal values

Monitoring only the features the team considers most important rather than all features, distribution shifts often arrive through less-watched features

Implementing monitoring but not connecting it to a retraining or investigation workflow an alert with no action plan degrades to noise

Transition: Level 4 to Level 5

Estimated Timeline: 3–6 months

Minimum Investment: Dedicated data platform engineering time; feature store infrastructure; streaming pipeline infrastructure

What Must Change

Adopt a feature store (Feast for open-source, Tecton or Vertex AI Feature Store for managed) to centralise feature computation and guarantee training-serving consistency

Migrate feature computation logic from model-specific preprocessing scripts into the feature store, defining features as reusable, versioned, tested assets

Build streaming feature computation for any use case requiring freshness under 30 minutes: the pipeline must compute features as events arrive, not on a batch schedule

Establish a formal feature governance process: every feature added to the store requires documentation, ownership assignment, a deprecation policy, and validation criteria

Implement privacy engineering at the pipeline level: automated PII detection, masking, anonymisation, and audit logging for regulated data

Define and enforce SLAs for data freshness, pipeline latency, and feature store availability

When Not to Make This Transition

The Level 4 to Level 5 transition is expensive in engineering time and ongoing maintenance. Before committing to it, answer three questions honestly: Do any of your ML use cases genuinely require sub-minute feature freshness? Do you have multiple ML models that would benefit from shared feature computation? Does your regulatory environment require machine-readable audit trails of data processing decisions? If the answer to all three is no, Level 4 is the right stopping point and that is a valid and mature choice, not a compromise.

Section 5: Industry Benchmarks, Where Does Your Sector Stand?

Based on Code-B's experience working with engineering teams across industries, combined with published industry research on ML production readiness and MLOps adoption rates, the following section provides a realistic picture of where different sectors typically land on the maturity model, and what competitive pressures are driving improvement.

The Regulatory Acceleration Effect

One of the clearest patterns in this data is the relationship between regulatory pressure and maturity advancement. Fintech organisations consistently achieve higher maturity than their

technical sophistication alone would predict, because regulatory requirements for auditability, reproducibility, and model explainability create structural incentives to build Level 3 and Level 4 pipelines.

The EU AI Act, which came into force in stages from 2024 through 2026, is beginning to have the same effect on healthcare and high-risk AI applications more broadly. Organisations required to produce technical documentation of their AI systems' data processing steps are discovering that their current infrastructure cannot satisfy those requirements and that closing the gap requires advancing from Level 2 to Level 3 or Level 4.

For organisations in sectors where regulation is tightening, maturity advancement is not only a technical investment – it is a compliance investment that reduces regulatory risk and the cost of future audits.

Where Competitive Pressure Is Forcing Maturity Advancement

In e-commerce and fintech, the competitive dynamics of recommendation and fraud systems are driving maturity advancement organically. Organisations whose recommendation models return stale or poorly processed features serve worse recommendations, which leads to lower conversion rates, which leads to revenue loss. The feedback loop is rapid and measurable, creating business pressure to invest in pipeline quality even without regulatory mandate.

In B2B SaaS and manufacturing, the feedback loop is longer and the consequences of poor data processing are less immediately visible, making these sectors more likely to remain at Level 2 longer. Teams in these sectors often do not know their models have degraded until the business metric the model was designed to improve (churn rate or equipment downtime) has moved meaningfully in the wrong direction.

Section 6: Case Study, From Level 2 to Level 4 in Six Months

The following case study is based on a real engagement with a financial services client (details anonymised for confidentiality). The organisation was a UK-based lender using ML for credit risk scoring. They had deployed a logistic regression model to production 18 months prior. The model had not been retrained since initial deployment.

The Starting State: Level 2 with Level 1 Characteristics

When Code-B's initial audit was performed, the credit risk scoring pipeline exhibited the following characteristics:

Preprocessing existed as a 600-line Python script with no tests, no version control, and several commented-out code blocks of unclear provenance

The train/test split was performed after StandardScaler was fit meaning test set statistics had influenced the scaler, and the validation performance was optimistically biased by approximately 2–3 percentage points on AUC

The serialised model file did not include the fitted scaler; the production serving code applied a separately maintained scaling configuration that had diverged from the training configuration 8 months after initial deployment

Missing value imputation used the mean of the entire dataset (including test rows) for two features, representing textbook data leakage

No monitoring existed; the team had no visibility into whether the distribution of incoming applications in the current month matched the distribution of the training data from 2022

The data engineer who had written the original pipeline had left the organisation; their tribal knowledge had not been documented

Beyond the maturity scores, the audit identified a specific business risk: the credit risk model's apparent performance at training time (AUC 0.78) was inflated by data leakage. The estimated true AUC, corrected for the leakage, was approximately 0.71. The model had been approving and declining loan applications based on a scoring function that was meaningfully weaker than the team believed and had been doing so for 18 months.

The Intervention: Six Months to Level 4

Months 1–2: Establishing Level 2 and 3 Foundations

All preprocessing logic was moved into a Scikit-learn ColumnTransformer pipeline; the train/test split was corrected to occur before all fitting operations

The data leakage was eliminated: imputation values were computed on training data only, and the fitted imputers were serialised as part of the model artefact

A Git repository was created for the ML codebase; the full preprocessing pipeline, model training script, and evaluation script were committed with documentation

Data was migrated from local CSV files to a versioned S3 bucket; each model training run was logged with the S3 path and timestamp of the data used

A retraining run was performed with the corrected pipeline; the model was re-evaluated and the new, honest AUC of 0.72 was documented — slightly lower than the inflated previous number, but now reliable

Basic unit tests were written for the five most complex transformation functions

Month 3–4: Implementing Monitoring and MLOps

Great Expectations suites were configured for the loan application data: schema validation, range checks on numerical features, and cardinality checks on categorical features

Evidently, AI was deployed to monitor feature distributions in production; baseline statistics were computed from the current training dataset and stored as the monitoring reference point

An early discovery: the distribution of one key income-verification feature had shifted substantially in the 18 months since the model was trained; income bands that were rare in 2022 were now common and vice versa. The model had been making systematically worse decisions for this segment without anyone knowing

MLflow was adopted for experiment tracking; the retraining run logged all parameters, preprocessing configuration, and evaluation metrics

A CI/CD pipeline was configured using GitHub Actions; tests ran on every pull request to the ML codebase

A retraining was triggered based on the discovered distribution shift; model performance improved to AUC 0.76 on current data

Month 5–6: Reaching Level 4

Alerting thresholds were calibrated based on historical distribution variance for each monitored feature

A retraining runbook was documented: what conditions trigger retraining, who approves the new model, and what validation criteria must be met before deployment

Privacy classification was completed for all input features; two features containing quasi-identifiers were masked before the feature engineering step

Test coverage reached 85% for preprocessing functions; integration tests covered the full pipeline from raw data ingestion to prediction output

A preprocessing review checklist was adopted as mandatory for any change to the pipeline

The most significant business outcome was not in the metrics above. The discovery that the model's decision-making had been operating on a shifted data distribution for 18 months and that a specific income-verification feature had created systematic errors for an entire borrower segment was identified by the monitoring infrastructure put in place in Month 3. Without that monitoring, the error would have continued indefinitely. The ability to detect, diagnose, and correct that kind of silent failure is what Level 4 infrastructure makes possible.

Section 7: Getting Started, Code-B's Data Pipeline Audit

The ML Data Processing Maturity Model is a diagnostic framework. The self-assessment tells you where you are. The roadmap tells you what changes are needed. The question that remains is whether those changes are best made by your internal team, by a specialist partner, or by some combination of the two.

For teams that have identified significant gaps in their assessment, particularly in data quality validation, leakage prevention, or production monitoring, the fastest and most reliable path to Level 3 or Level 4 is an engagement with a specialist who has executed that transition before. Not because internal teams lack the capability, but because the transition requires a combination of ML engineering knowledge, software engineering discipline, and familiarity with the tooling landscape that takes time to develop from scratch.

What a Code-B Data Pipeline Audit Covers

Our initial engagement with any client begins with a structured data pipeline audit. Over five to ten working days (depending on pipeline complexity and data volume), we produce a comprehensive assessment of your current preprocessing infrastructure across all five domains of the maturity model.

The audit deliverables include a scored maturity assessment with evidence for each score, an itemised list of the highest-priority gaps ranked by business risk, a recommended transition roadmap with timelines and resource estimates, and identification of any immediate risks (data leakage, training-serving skew, or undetected drift) that require urgent remediation.

The audit is a fixed-scope, fixed-price engagement. Its output is useful regardless of whether you choose to engage Code-B for remediation work — the assessment and roadmap stand on their own as planning documents.

Who This Is For

Engineering leaders at scale-ups who have deployed ML models and are uncertain about their production reliability

CTOs and Heads of Data who are planning a significant ML investment and want to understand their current baseline before committing

ML teams who have experienced unexplained model performance degradation and suspect but cannot confirm, a data processing root cause

Organisations in regulated industries (fintech and health tech), preparing for an AI audit or regulatory review, that need to understand the documentability of their preprocessing decisions

Published 2026. Version 1.0 | code-b.dev |