

Troubleshooting Common Data Science Problems



Troubleshooting is a core competency, not an afterthought



Most effort goes into diagnosing and fixing silent failures



In 2025–2026, the model is rarely the blocker



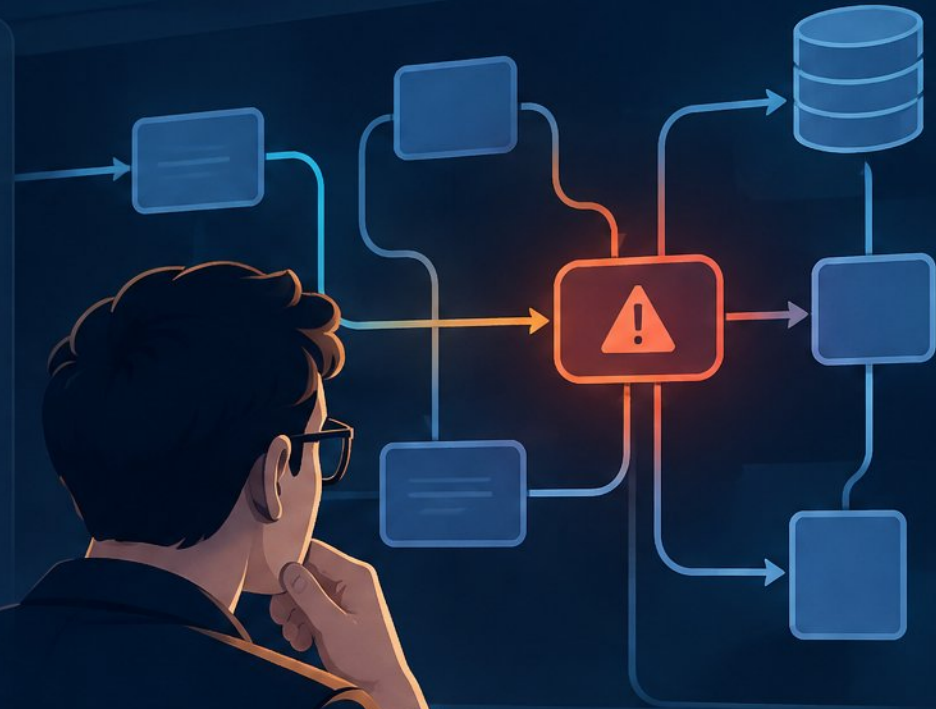
Real failures cluster in data, metrics, workflow, and monitoring



Shared vocabulary: symptom, root cause, mitigation, prevention



Six failure zones: data, leakage, training, reproducibility, production, process



The Data Science Lifecycle and Where It Breaks

From idea to impact — and the real fault lines along the way.



Lifecycle in practice:
problem → data →
cleaning → EDA →
features → training →
evaluation →
deployment →
monitoring



**Failure rarely at
model training;**
it clusters in data
understanding,
preparation, and
post-deployment



Failure propagates:
a cleaning fault
resurfaces weeks later
as an unexplained
accuracy drop



Fault sources:
data-related,
framework/tooling-
related, conceptual
and resource-related



**Practitioners need
fault localization;**
managers need
ownership and
process gates

Data Quality and Integrity Issues

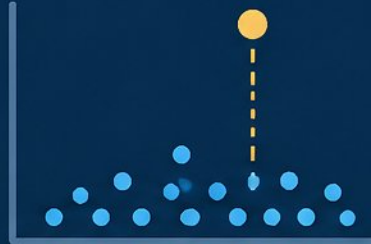
Clean, consistent, and trustworthy data enables reliable models.

BETTER DATA
STRONGER OUTCOMES



Missing values

Missing values: distinguish MCAR, MAR, MNAR before imputing, dropping, or flagging



Outliers

Outliers: combine rules with statistical/ML detection, then cap, correct, or keep



Duplicates and entity resolution

Duplicates and entity resolution: one customer under many IDs distorts training and evaluation



Label noise

Label noise: mislabeled targets and inconsistent guidelines mimic architecture failures



Taxonomy of 23 quality problems, 4 detection strategies

Taxonomy of 23 quality problems, 4 detection strategies: rules, statistics, reference data, humans



rules



statistics



reference
data



humans

Treating Data Quality as First-Class Engineering

RELIABLE DATA • TRUSTED MODELS • STRONGER OUTCOMES



Data tests are the unit tests of the pipeline: validate every ingest and transform



ISO/IEC 5259 series gives multidisciplinary teams a shared quality vocabulary and governance



ML-specific dimensions: annotation quality, fairness, privacy, version control, synthetic data provenance



Temporal pitfalls: gaps, irregular sampling, timezone and daylight-saving errors, duplicated rows



Ship a small versioned validation suite first: schema, null, range, and distribution checks

HIGHER QUALITY DATA
BRIGHTER POSSIBILITIES

Data Leakage: The Silent Accuracy Inflator

When tomorrow's information leaks into today's training data...



...evaluation scores look better, but fail in deployment.



Symptom

- Leakage uses outside information, giving scores that fail in deployment
- Four common forms: train/test overlap, preprocessing, multi-test, and target leakage



Root Cause

- Classic offenders: feature engineering before splitting, full-dataset target encoding, outcome features like "days until payment"



Mitigation

- Static tools catch it early: NBlyzer found 25 true leakages across 2,111 Kaggle notebooks at ~93% precision
- LeakageDetector IDE plugins flag Overlap, Preprocessing, and Multi-test issues with automatic quick fixes



Prevention

- Prevention habit: split first, fit transformers on training data only, drop features unavailable at prediction time

Splitting, Cross-Validation, and Metric Pitfalls

GET REALISTIC EVALUATION. BUILD MODELS THAT GENERALIZE.



Random Split

Can mix past and future data, leading to information leakage.

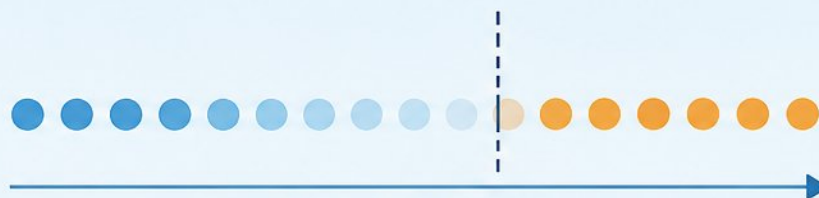


Time-ordered data



Temporal Split

Keeps the natural time order, training on the past and evaluating on the future.



Time-ordered data



Match the split to deployment: random, temporal, or group-aware



Never shuffle time series — random splits leak the future and inflate scores



Reserve a validation set for tuning; touch the test set once



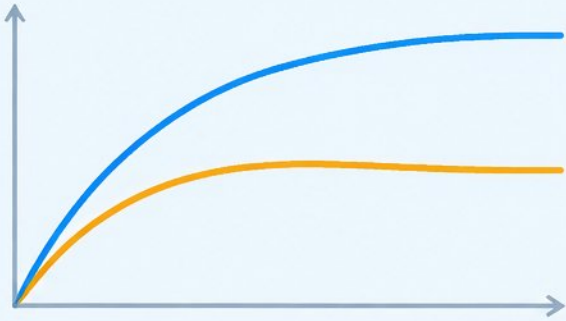
Accuracy misleads under imbalance — use precision, recall, PR-AUC, calibration



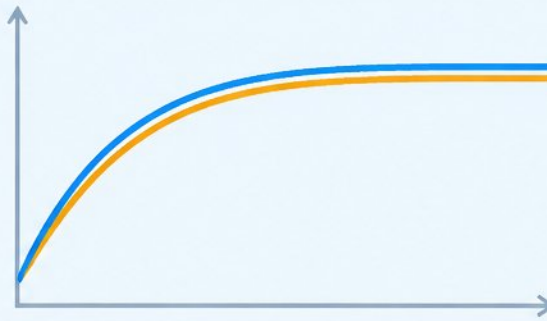
Version datasets, record the split strategy, flag benchmark contamination in LLMs

Diagnosing Model Performance Problems

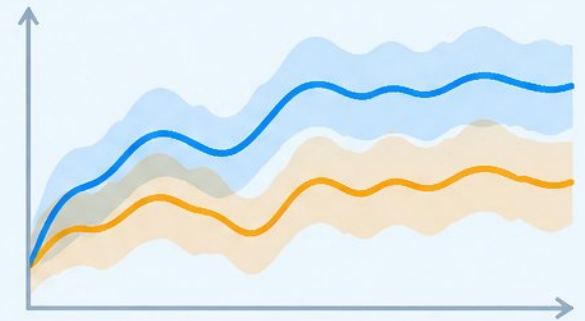
Overfitting



Underfitting



Unstable Across Runs



Learning curves are the primary diagnostic: diverging train/validation means overfitting



Both curves high and flat signal underfitting; add capacity or features



Unstable across runs? Average results over multiple random seeds



Class imbalance: resampling, weights, and thresholds are tools, not defaults



Error analysis first: inspect misclassified, high-loss examples and the labels themselves

Inspect Misclassified Examples

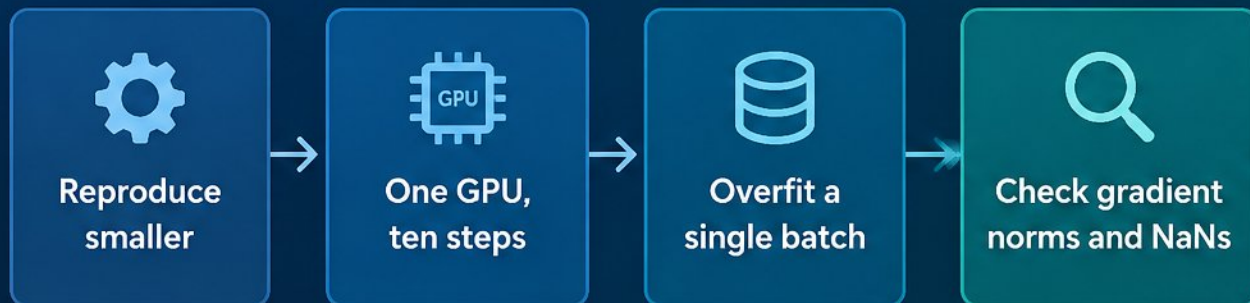


Debugging Training Loops and Silent Training Errors

FASTER INSIGHTS
MORE RELIABLE MODELS

A systematic approach to find and prevent silent training errors.

Debugging Workflow



Silent Error Traps



Missing `zero_grad()`



Bad gradient clipping



Softmax instead of logits



Symptom

- Loss and accuracy are noisy, periodic, and miss silent errors
- TrainCheck caught 18 of 20 real silent errors within one iteration



Root Cause

- Root causes split ~32% user code, ~32% framework bugs
- Classic bugs: missing `zero_grad()`, bad gradient clipping, softmax instead of logits



Mitigation

- Monitor per-layer gradient norms; use adaptive thresholds for vanishing/exploding



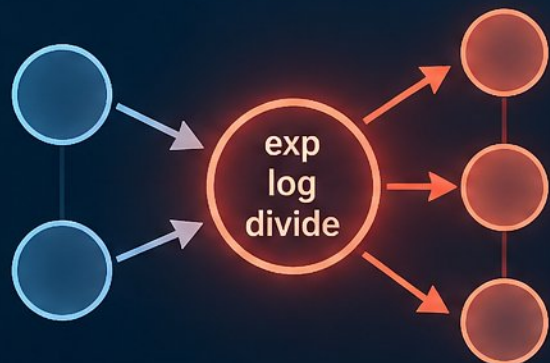
Prevention

- Discipline: smallest config, one GPU, ten steps, overfit a single batch

Numerical Stability, OOM, and Large-Model Failures

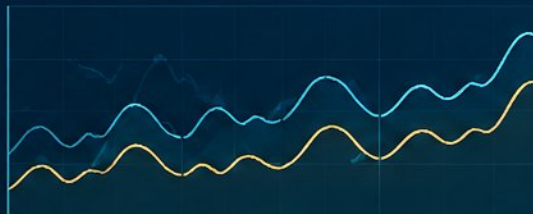
TRAIN > MONITOR > DEBUG > SCALE

NaN/Inf Propagation



NaN/Inf
spreads
through
the graph

Gradient Norms (Instrument)



Gradient norms



Activation stats



Weight update ratios

Memory Usage



OOM



NaN/Inf start in `exp`, `log`, or `divide` ops; they spread through the whole graph



Instrument, don't guess: log gradient norms, activation stats, weight update ratios



OOM causes: oversized batches, duplicated ops, model leaks, full dataset in memory



Large-scale modes: loss stalls, scheduler failures, comms overhead, mixed-precision errors



Never debug a 64-GPU run before a 10-step, 1-GPU sanity check

Reproducibility and Environment Failures

Same code. Different environments. Different outcomes.



REPRODUCE
UNDERSTAND
BUILD TRUST



"Works on my machine": dependency drift, OS/architecture gaps, unseeded randomness, undeclared data versions



Version everything, not just code: datasets (DVC-style), models in registries, experiment tracking



2026 practice: lockfiles, containers, GPU-aware managers, layered venvs (venvstacks) to avoid duplicating PyTorch



Determinism costs speed; caching and per-run receipts trade overhead for verifiability



Handoff checklist: seed all, pin deps, snapshot data, record environment per release

A Reproducibility Kit

Capture what matters. Avoid painful surprises.



Lockfile

Prevents
dependency
drift



Container

Prevents
OS/architecture
gaps



Seed list

Prevents
unseeded
randomness



Data snapshot

Prevents
undeclared
data versions



Model registry

Prevents
missing
model versions

Unreliable Environment



VS



Reproducible Environment



Production Failures: Skew, Drift, and Silent Decay



Training-serving skew: schema skew (format/distribution) and feature skew (different engineering logic) match production bug damage



Drift types differ: covariate shift, concept shift, and benign drift need distinct responses — retraining isn't always the answer



Silent failure is the dangerous pattern: models stay confident while quality decays



Up to 20% of failures reach customers before internal detection

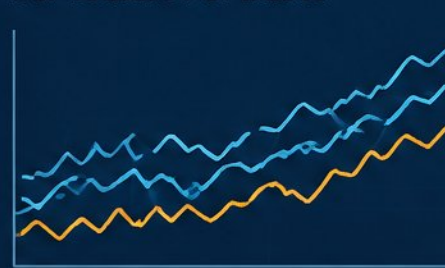


Detection without labels: PSI, MMD joint-distribution tests, output drift, conformal coverage, CUSUM

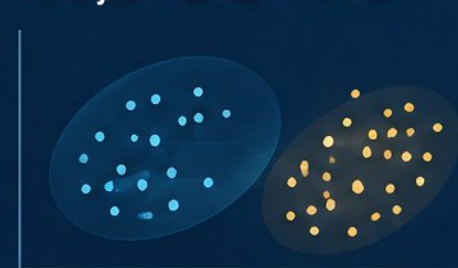


Caution: confidence degradation does not reliably precede drift; track it, don't rely on it as an early warning

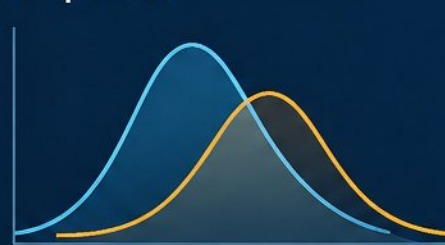
Per-feature PSI traces



MMD joint-distribution tests



Output drift



CUSUM



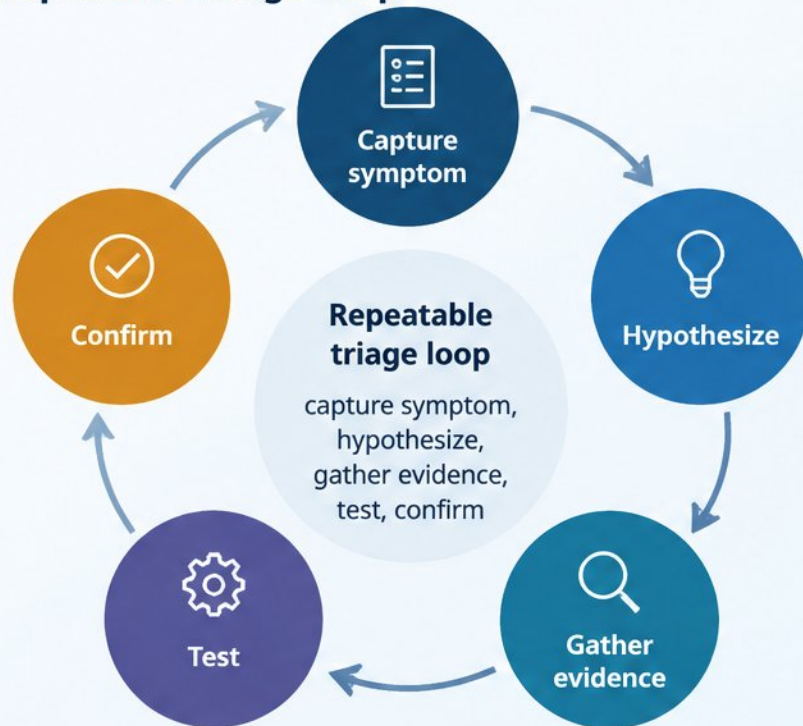
High confidence, declining quality



Diagnostic Workflow and Root Cause Analysis

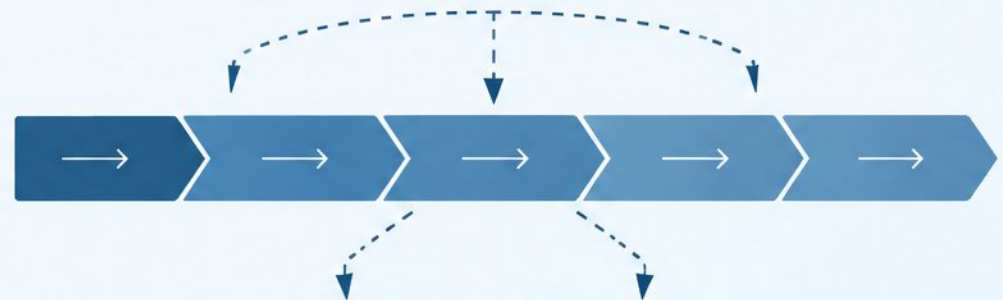
FIND
ISOLATE
UNDERSTAND
RESOLVE

Repeatable Triage Loop



Binary-Search Across Pipeline Steps

Narrow the fault fast



Fault-Zone Map (Pipeline Stages)

Localize before theorizing: isolate the stage, reproduce on a sample



Key Points

-  Repeatable triage loop: capture symptom, hypothesize, gather evidence, test, confirm
-  Localize before theorizing: isolate the stage, reproduce on a sample
-  Binary-search across pipeline steps to narrow the fault fast

-  ML root cause differs from service RCA: causal graph often unknown
-  Logs, traces, and data/model tests form the evidence base
-  Escalate with symptom, best hypothesis, ruled-out causes, and impact

Case Studies and Hands-On Scenarios

Case 1 — churn model
scores too good: total-charges
plus pre-split preprocessing
leak the target



Case 2 — silent pipeline failure:
upstream field meaning changed;
single-feature drift, no parent drift



Case 3 — handoff crisis:
no data snapshots, seeds, or
locked environment, so retraining
is impossible



Case 4 — training divergence:
gradient-clipping logic bug with
no exception and no metric anomaly



Lab (60–90 min): diagnose a
broken notebook, fix issues,
present evidence-backed root cause



Prevention Checklist and Team Practices

Turn best practices into everyday habits.



Pre-flight gate

- data validation
- split correctness
- leakage tests
- reproducibility
- documented baseline



Monitoring baseline

- data quality checks
- drift thresholds
- pipeline SLAs
- named on-call owner



Document for audit

- model cards
- datasheets
- evaluation reports
- deployment specs
- lifecycle plans



One owner per concern

- data quality
- model performance
- production reliability



Learning loop

- blameless postmortems
- shared playbooks
- scheduled threshold reviews

FROM PREVENTION TO A MORE RELIABLE TOMORROW

SAME
DISCIPLINE
BRIGHTER
POSSIBILITIES

Wrap-Up, Resources, and Your Next Steps

TURN INSIGHTS INTO ACTION



Key Takeaways



Highest-impact habits: debug data first, verify splits, reproduce small, instrument for silent failure



Mindset beats tooling: more tools won't fix unclear metrics or unowned pipelines



Run one project through the pre-flight checklist; log top three gaps



Close with a short quiz, a reflection prompt, and role-specific follow-ups



Your Next Steps and Resources



Practitioners

Apply the checklist, identify your top three gaps, and build your plan.



Engineers

Instrument for silent failure and make reproduction easy.



Managers

Create clarity on metrics and ownership, and support follow-ups.



Understand



Build



Validate



Deploy



Next Steps

PersonWise.

PersonWise • AI digital-human course platform



Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/d8c9275d/public