

baswe.Ai Engineer Accelerator™

ML Foundations Checkpoint

Format:

Build + Loom Walkthrough + Technical Exam

Time investment:

10–14 hours

Stack:

Python · scikit-learn · MLflow · DVC · Great Expectations · GitHub Actions



baswe's

AiEngineerAccelerator

Pre-checkpoint checklist

Before you start, confirm all of the following.

Accounts & access

- GitHub account
- Loom account (free tier is fine for a single recording).
- Optional: a free DagsHub or self-hosted MLflow setup for the experiment tracking server. Local MLflow is also acceptable.

Local environment

- Python 3.10 or later.
- Git, configured with your name and email.
- A code editor with Python tooling.
- Terminal comfort with venv / uv / poetry.
- Docker is optional but useful for reproducibility.

Stack

- scikit-learn (latest stable) — model training and evaluation.
- pandas — data manipulation.
- MLflow — experiment tracking.
- DVC — data versioning.
- Great Expectations — data quality validation.
- GitHub Actions — CI/CD pipeline.
- pytest — test suite.
- Open within constraints: you may choose any scikit-learn-compatible model class, any serving layer (FastAPI, Flask, BentoML), and any monitoring instrumentation approach. Justify your choices in the README.

Part 1 The Build

Scenario

You have been hired as an Applied AI/ML engineer at Meridian Health, the regional health insurance company you assessed for ClaimsAssist in Module 4. Meridian's retention team has come to your team with a new problem: ~24% of members do not renew at the end of their plan year, and the retention team currently has no way to identify at-risk members early enough to intervene.

Your task: build a member churn prediction pipeline that the retention team can run monthly to identify the top ~5–10% of members most at risk of non-renewal. The pipeline must be reproducible, monitored, and deployable. This should not be just a notebook, it should be a production-shaped system that another engineer could pick up and run tomorrow.

Module 4 governance carries forward

Meridian's AI Safety & Governance team (you, last module) has documented constraints on what features can be used. The provided data dictionary will flag features that are sensitive or off-limits. This continuity is intentional: your governance work doesn't end when you start engineering.

Provided materials

- Meridian member dataset (link in The Edge: ML Foundations -> Checkpoint -> START): train.csv (40K rows, 2024 renewal decisions), test.csv (15K rows, Jan–Jun 2025).
- Data dictionary describing every feature, its source, and any governance flags.
- Project brief: business context, success metrics, deployment expectations.

Required architecture

Data validation	Great Expectations suite that validates raw inputs against schema, range, and missingness expectations. Validation runs as a pre-training step. Validation failure aborts the pipeline with a clear error.
Data versioning	DVC tracks the dataset. Different model runs reference specific data versions. README documents how to reproduce a specific run.
Feature engineering	Reproducible feature pipeline (sklearn Pipeline or equivalent) that handles missing values, encoding, and scaling. Same pipeline applies at training and inference.
Model training	scikit-learn classifier. You choose the model family and justify it. Hyperparameter tuning via cross-validation. Final model is fit on training data only.
Experiment tracking	MLflow tracks every training run: parameters, metrics, artifacts, model registry. At least 3 distinct experiments comparing

	meaningfully different approaches (not just hyperparameter sweeps of one model).
Evaluation	Honest evaluation on test.csv using metrics appropriate to the business problem. Includes both aggregate and subgroup performance (by plan_type, by age bucket). Calibration curve included.
Serving	A simple inference interface (FastAPI endpoint, CLI, or batch script) that loads the registered model and produces predictions for a CSV input. You choose the serving pattern; justify it.
Monitoring	Instrumentation that logs prediction-time inputs, predictions, and (when available) outcomes. Documented monitoring plan: which signals you would track, what thresholds you would alert on, and why.
CI/CD	GitHub Actions workflow that runs on every PR: lint, tests, data validation against a tiny sample, training smoke test (no full retrain), model registry sanity check.
Tests	pytest suite covering ingestion, feature pipeline, training, evaluation, and serving. Mocks are fine where external services are required.

Required repo structure

```

churn-pipeline/
├── README.md                # Architecture decisions + how to run
├── pyproject.toml           # Or requirements.txt - pinned versions
├── .env.example             # Document required env vars
├── .dvc/                   # DVC config
├── .github/workflows/
│   └── ci.yml              # GitHub Actions pipeline
├── data/
│   ├── raw/               # DVC-tracked
│   ├── processed/         # DVC-tracked
│   └── expectations/       # Great Expectations suites
├── src/
│   ├── ingest.py          # Reads + validates raw data
│   ├── features.py        # Reproducible feature pipeline
│   ├── train.py           # Training entry point
│   ├── evaluate.py        # Test-set evaluation
│   ├── serve.py           # Inference interface
│   └── monitor.py         # Monitoring instrumentation
├── mlruns/                 # MLflow tracking (gitignored)
├── tests/
│   ├── test_ingest.py
│   ├── test_features.py
│   ├── test_train.py
│   ├── test_evaluate.py
│   └── test_serve.py
├── notebooks/
│   └── exploration.ipynb   # Optional - your scratch work
└── dvc.yaml                # Pipeline definition

```

Required technical checks

1. `python -m pytest tests/` exits 0.
2. `python -m src.ingest --raw data/raw/train.csv` runs end-to-end and produces validated output.
3. `python -m src.train` runs end-to-end and produces an MLflow run with logged parameters, metrics, and a registered model artifact.
4. `python -m src.evaluate --model <run_id> --data data/raw/test.csv` produces a metrics report including aggregate AND subgroup performance.
5. `python -m src.serve` accepts a CSV input and produces a predictions CSV.
6. GitHub Actions CI workflow runs successfully on at least one PR in your repo history.

Engineering judgment shows in the metrics you pick

There is no single "right" model class for this problem. There IS a wrong way to report metrics. A 95% accurate model on a 24% churn problem might be useless.

README requirements

- One-paragraph summary of what you built and what it does for the retention team.
- Architecture diagram (Mermaid, draw.io export, or a clear ASCII diagram is fine).
- Data section: which features you used, which you deliberately excluded, why. Reference the data dictionary's governance flags.
- Model section: which model family you picked, what alternatives you compared against in MLflow, why this one won. Include the metrics that mattered to you.
- Validation section: what Great Expectations suite catches, with an example of what "bad data" would trigger which expectation.
- Reproducibility section: how to recreate your best model end-to-end from the repo, including DVC pull and the exact MLflow run.
- Monitoring section: which signals you would track in production, what thresholds, what alerts. Be specific.
- How to run: clone → install → dvc pull → validate → train → evaluate → serve. Six commands or fewer.
- What you'd do with another week: ranked list of next improvements with the reasoning.

Part 2 Loom Walkthrough

Format

- Single Loom recording, 20–35 minutes, screen share with face cam on.
- Public or unlisted Loom link, included in your README

Required structure

1. Architecture overview (4–6 minutes)

- Walk through your architecture diagram.
- Explain ingestion → validation → training → evaluation → serving → monitoring, end to end.
- Call out where you made a non-default decision and why.

2. Data + governance decisions (4–6 minutes)

- Show the data dictionary. Walk through which features you used and which you excluded.
- Name at least one feature you deliberately excluded for governance reasons and explain why.
- Name at least one feature you initially considered including but caught as a leakage risk.

3. Live demo (4–6 minutes)

- Run your pipeline end-to-end live: validation, then training (or load a pre-trained run), then evaluation, then a sample prediction.
- Show the MLflow UI with your tracked runs. Point out the run you registered as production-ready and explain why it beat the alternatives.
- Narrate what's happening so a hiring manager could follow along.

4. Tradeoffs you made (4–6 minutes)

- Walk through 3 specific tradeoffs you made under the time constraint.
- For each: what you chose, what you sacrificed, and what would change with another week.
- This is where engineering judgment shows. Generic answers ('I would add more monitoring') score low. Specific answers ('I would add population stability index alerts at 0.2 on the top-5 features because right now I would not notice drift until the retention team flagged it') score high. Companies love engineers with opinions, it shows proof of tested beliefs.

5. What you would do differently (3–5 minutes)

- Honest reflection: what would you change if you started over?
- What did you not know going in that you know now?

Part 3 Technical Exam

Format

- Open-book. Open-internet.
- Do **NOT** use an LLM to generate your answers. Do **NOT** Google the entire question. Cheating on these only cheats yourself. Time-boxed: 1–2 hours. We do not enforce the timer, but if you spend 4+ hours on this section, that is a signal you might not have understood the concepts deeply enough.
- Written answers in a single Markdown file (exam-answers.md)

Section A: ML Foundations (6 points)

Q1 (3 pts)

Define overfitting, underfitting, and the bias-variance tradeoff. For each, describe one concrete signal you would see in your training metrics or validation curves that would tell you you're in that regime. Use specific terminology from your scikit-learn build where it applies.

Q2 (3 pts)

You're training a binary classifier on a 5% positive class. Walk through the metrics you would prioritize, in order, and explain why accuracy is not on your list. Name a specific scenario where you would optimize for recall over precision, and one where you would optimize for the reverse.

Section B: Data Engineering & Quality (6 points)

Q3 (3 pts)

Compare ETL and ELT. For each: what kind of data infrastructure assumes which pattern, what are the tradeoffs, and where does the line blur for ML workloads specifically? Give a concrete scenario where you would pick each.

Q4 (3 pts)

You receive a daily 200MB CSV from an upstream system. Some days it has missing columns, some days a column changes type, sometimes the file is 50MB and you don't notice. Walk through how you would catch these failures BEFORE they corrupt your training run.

Section C: Experiment Tracking & Versioning (6 points)

Q5 (3 pts)

Why do ML teams version data with DVC instead of just committing it to Git? Explain at least three problems DVC solves. Then describe a specific scenario where DVC alone isn't sufficient and you'd reach for something else (DataLake, Delta Lake, Iceberg, etc.).

Q6 (3 pts)

Walk through what MLflow tracks in a typical run and what it does NOT track that you still need to capture somewhere else. Then describe how you would use the MLflow model registry to manage promotion from staging to production. Be specific about the gates.

Section D: Deployment & CI/CD (6 points)

Q7 (3 pts)

You have a trained scikit-learn model. Compare three deployment patterns: batch inference (run nightly, write to a table), real-time API (FastAPI endpoint behind a load balancer), and embedded (model file shipped with a client app). For each: what is the right business scenario, what are the operational risks, and what monitoring is hardest?

Q8 (3 pts)

Your CI pipeline currently runs unit tests on every PR. What additional checks would you add for an ML project that you wouldn't add for a typical software project? Be specific, not just "run the training" but what artifacts you'd test, what you'd block merges on, and what you'd leave to manual review.

Section E: Monitoring & Production (6 points)

Q9 (3 pts)

Define data drift, concept drift, and prediction drift. For each, give one concrete signal you would compute and the threshold you would alert on. Explain why a system with no drift detection becomes silently broken faster than a typical software system.

Q10 (3 pts)

Your model has been in production for 6 months. Performance has degraded 8 percentage points on AUC and no one knows why. Walk through your debugging process step by step. What do you check first, second, third? When would you retrain vs. roll back vs. take the model down?

AiEngineerAccelerator