Atidot AI Engineer - Home Assignment

Goal

Ship a tiny **Decision Assistant** that:

1. trains a small churn-style classifier on **synthetic monthly panel** data (temporal split, no leakage) that separates **lapsing insurance policies** (the positive class) from **active insurance policies** (the negative class),
2. produces **lapse-prevention strategies** via **RAG**, **injecting the predicted lapse probability** for the customer into the prompt,
3. produces **lead-conversion strategies** via **RAG** for 3 synthetic leads (unrelated to the classifier),
4. runs via **one command** and finishes in **<5 minutes** on a typical laptop.

Solving the assignment using LLMs (e.g., ChatGPT) is **allowed and encouraged**. The RAG documents should be **simple and LLM-generated** (ChatGPT text pasted locally is fine).

The submission should be a Github repo.

Hard constraints

- Build time target: **≤2 hours** (on your side).
- End-to-end runtime: **<5 minutes** on a typical laptop.
- **One entry point:** `python run.py` creates all outputs under `out/`.
- **No external keys required.** If you use an API, include a no-API fallback and run with it by default.

What to build

Generate a synthetic temporal dataset of generic insurance policies. An insurance policy is defined by a coverage (the amount insured, e.g., $10000) and a premium (the price paid periodically, e.g., $100).

- Size: ~2k policies × 12 months.
- Columns (except target): `policy_id, month, age, tenure_m, premium, coverage, region, has_agent, is_smoker, dependents.` You are allowed to interpret those names as you wish, as long as it makes intuitive sense.
- Target: `lapse_next_3m` (lapse during the next 3 months).
- Include a simple drift after a date (e.g., `2023-07`).
- Include a **leakage trap** feature (e.g., `post_event_*`) and exclude it from the model. Document in `DISCUSSION.md`.

- Data split: strict time split (train, val, test).

Model & metrics

- Model: One fast XGBoost (or LightGBM) classifier with early stopping and light tuning via RandomizedSearch or Optuna, ≤30 trials.
- Metrics: AUC-PR (primary), precision@k (k=1%, 5%).
- Save the trained model, `metrics.json`, and a global SHAP bar plot. Discuss it very briefly in `DISCUSSION.md`.

RAG docs (two small corpora, LLM-generated)

- **Lapse corpus** (5–6 markdown docs): grace period, agent outreach, payment plans, loyalty discounts, seasonality/smoker coaching.
- **Lead corpus** (5–6 markdown docs): messaging by segment, touchpoint cadence, objection handling, value props, trial/discount guidelines.
- Use **TF-IDF** retrieval (no heavy deps).

Run

- **Lapse prevention branch:** pick 3 test customers (high/median/low risk), compute predicted lapse probability, inject it into their prompts, generate a short 3-step plan with [Doc#] citations.
- **Lead conversion branch:** define 3 lead profiles based on age, region, channel, needs, and objections, and generate 3-step conversion plans with [Doc#] citations.

Outputs
The command `python run.py` must perform all the above, and produce the required outputs.

Anti-Triviality Requirements

This assignment includes several built-in verification steps. You are expected to reason through and satisfy each of them in your own way.

1. **Data Leakage Safeguard.** You must detect and avoid target leakage.
2. **Temporal Integrity.** Your temporal split must be programmatically validated.
3. **RAG Faithfulness.** Each generated answer should be grounded in its retrieved sources.
4. **Probability-in-Prompt.** In the lapse-prevention branch, the predicted lapse probability for each demo customer must clearly appear inside the RAG reasoning flow.
5. **Determinism and Reproducibility.** Your results must be stable across runs.
6. **Runtime Discipline.** The complete pipeline must execute in under 5 minutes.

Scoring (100)

Engineering (30), ML soundness (25), RAG (25), Communication (15), Runtime (5).