Loan Default Prediction End-to-End ML System

Malyaj Mishra

Problem Statement:

Major Objective:

- Predict the **probability of loan default** for each customer using structured application data.
- Build an **end-to-end ML system**: versioned data, tracked experiments, containerized serving, and drift monitoring.

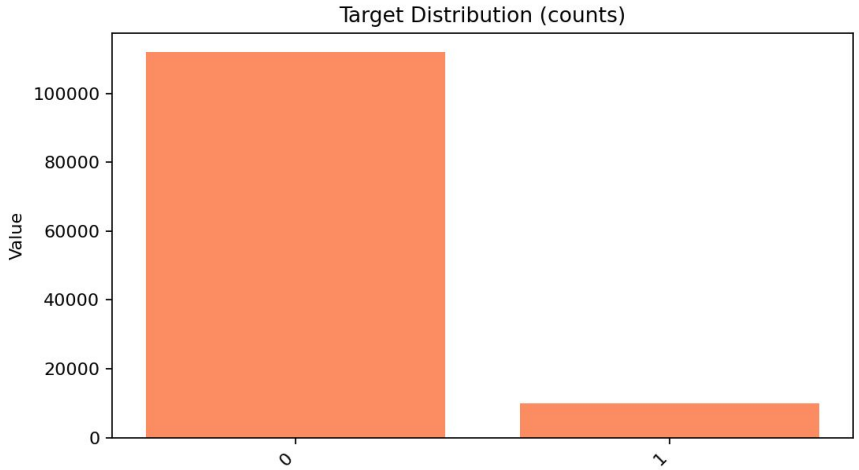
All the paths referenced in the slides are from the github repo here.

Data Overview:

Key insights:

- Dataset: `Dataset.csv` (source folder: `data/raw/`)
- Shape: **121,856 rows × 40 columns**
- Target: **Default** (binary)
- Class balance: **0 → 112,011 (91.92%)**, **1 → 9,845 (8.08%)**
- Feature types: income/credit amounts, ownership flags, credit bureau & score sources, demographics

Full EDA report can be accessed here.

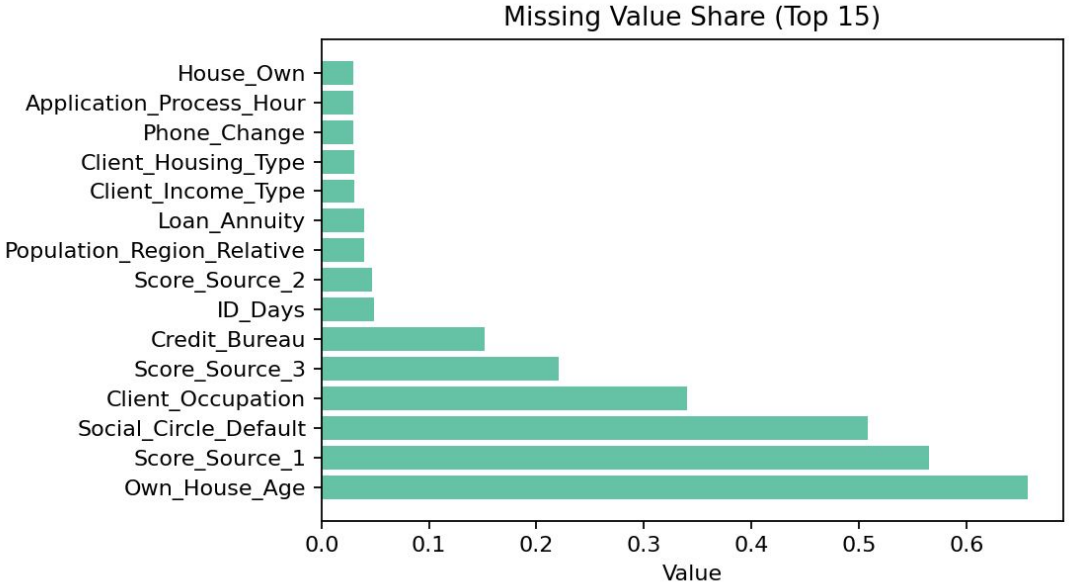


EDA Highlights:

Key highlights:

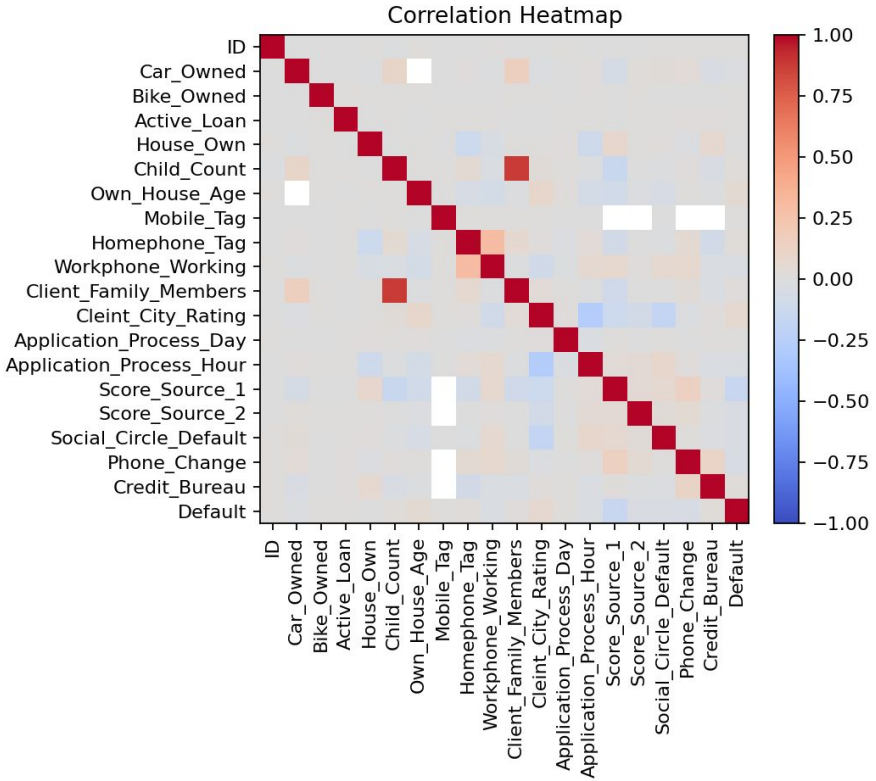
- **Imbalance: ~8.08%** defaults → PR-AUC & recall emphasized.
- **Missingness:** a handful of columns with moderate NaNs (imputed).
- **Signal:** income/credit magnitudes and score sources show strongest variation.

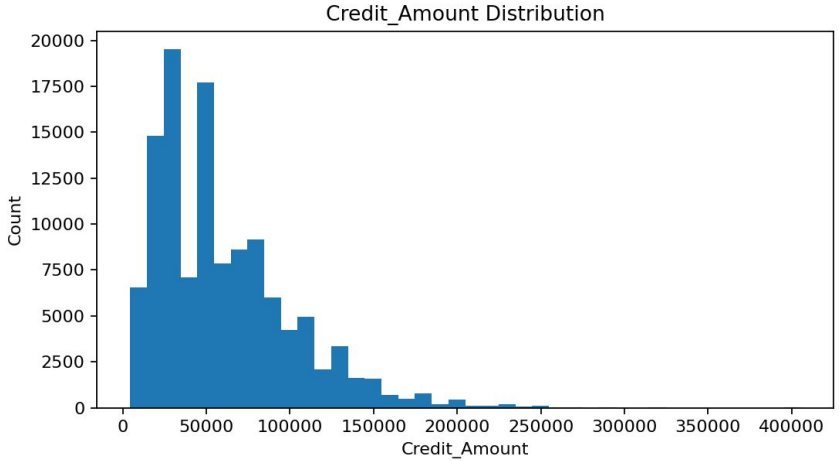
Some EDA analysis outcomes:



EDA Highlights:

Some EDA analysis outcomes:





Data Preprocessing & Features

Major Preprocessing Steps:

- **NumericCoercer** → fixes mixed types (e.g., `"19800"` vs `18000.0`).
  Imputation: `SimpleImputer` (median for numeric, most-frequent for categorical).
- **Scaling:** `RobustScaler` on numeric (outlier-robust).
- **Encoding:** `OneHotEncoder` on categorical **(sparse)** to keep memory low.
- **Schema-driven** via `src/features/preprocess.py`; reproducible in pipeline.

Outputs to `data/processed/` (parquet) during training.

Processed data can be accessed here.

Train/Validation/Test Strategy & Tracking:

Major steps:

- Split: **70/10/20** (train/val/test) with stratification on `Default`.
- Model: **XGBoost** (sparse OHE), cost-aware thresholding later.
- **MLflow** logs params, metrics, artifacts for auditability (`mlruns/`).
- Data version & splits saved for reproducibility (`data/processed/`).

Class Imbalance & Metric Strategy:

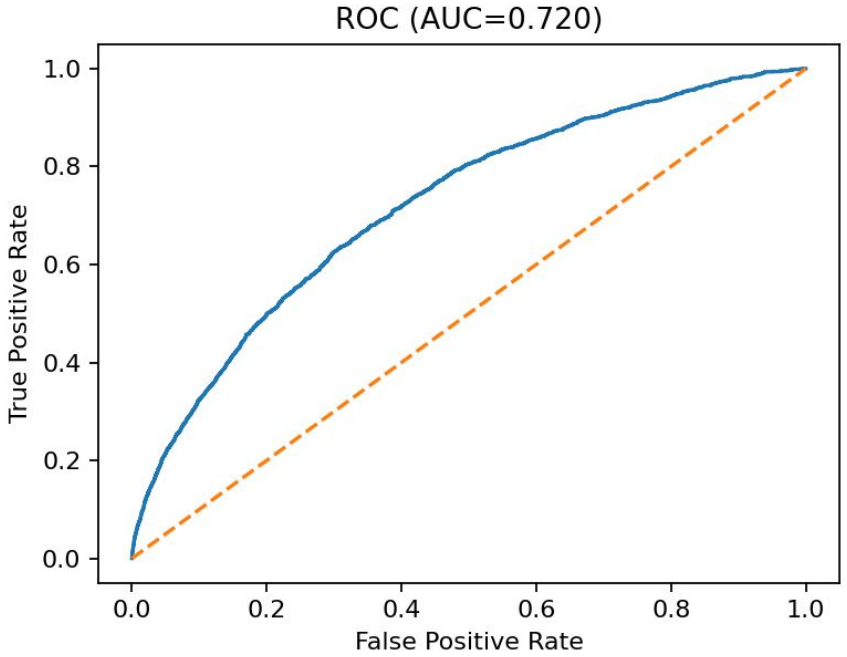
Major steps:

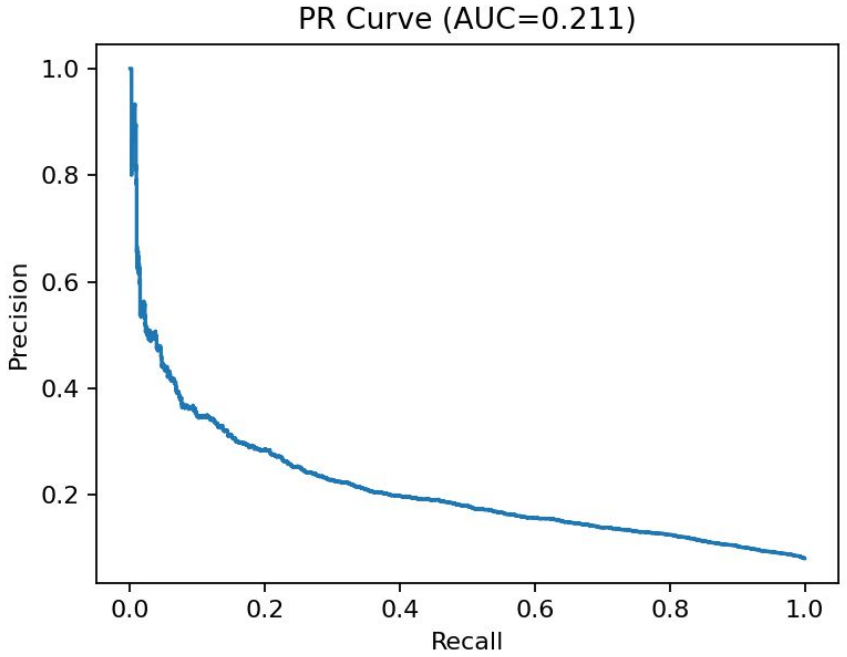
- **Severe class imbalance (8.08% default)** → optimize for **PR-AUC / Recall**, not accuracy.
- Tune operating point via **threshold sweep** (precision/recall/F1 vs threshold).
- XGBoost configured with **sparse inputs** and **scale_pos_weight** (neg/pos) when applicable.
- Business-aligned option: **cost-min thresholding** (e.g., 5× cost for FN vs 1× FP).

Evaluation Metrics:

- Test Results Overview:
  - **ROC-AUC:** 0.72
  - **PR-AUC:** 0.21
  - **F1**: 0.26
  - **Precision**: 0.21
  - **Recall**: 0.36
  - **Threshold**: 0.13

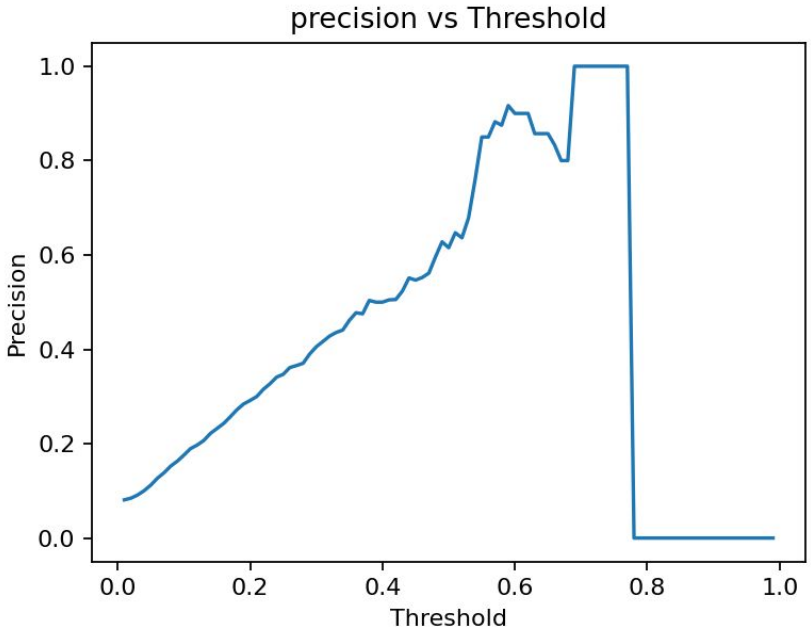
Evaluation Metrics:

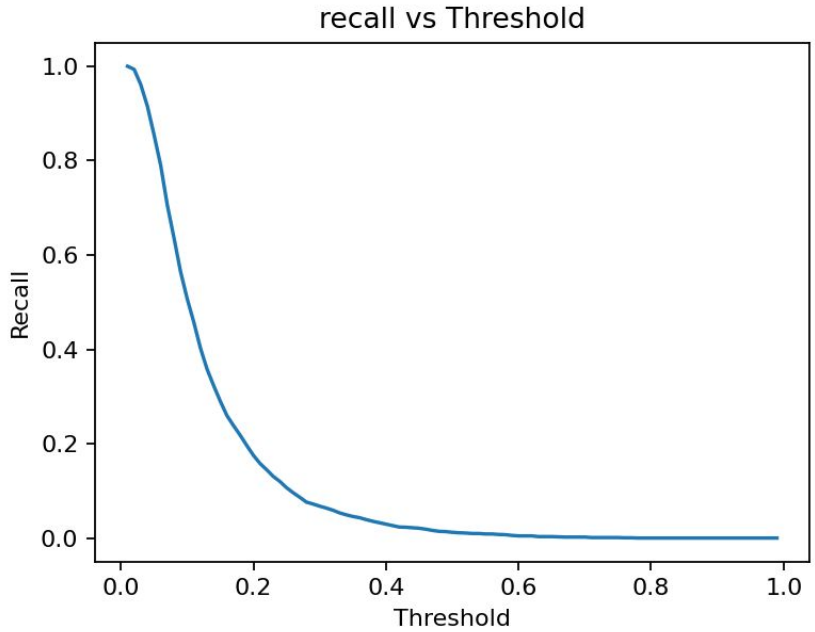




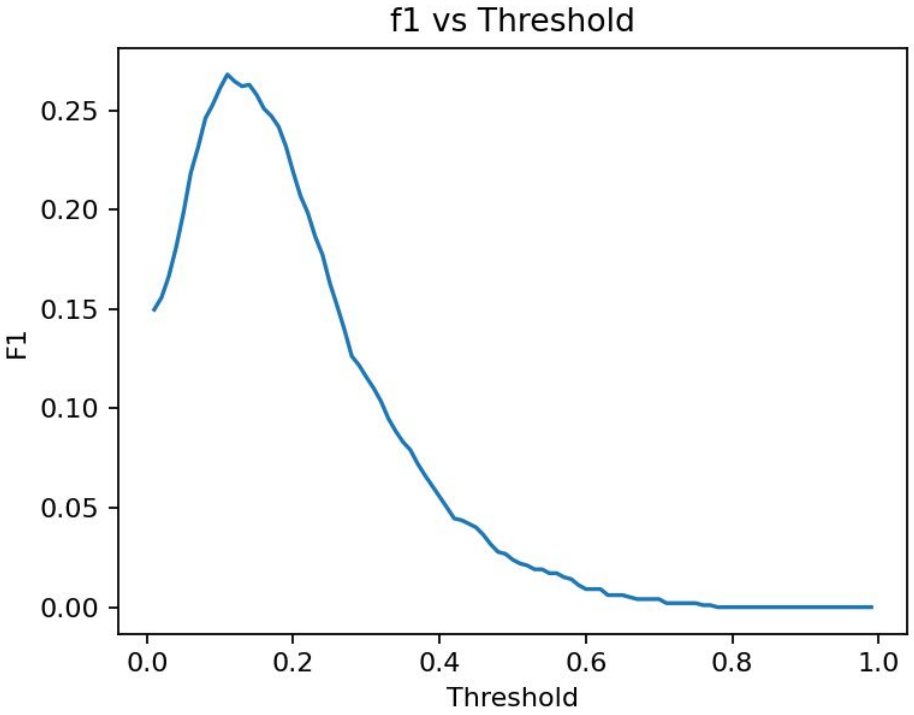
Evaluation Metrics-Business Interpretation:

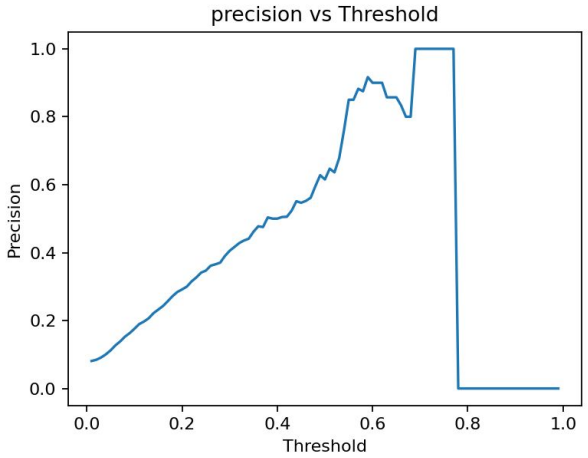
- Lowering the threshold increases recall (catch more defaulters) at the cost of precision (more false positives).
- Selected threshold(0.13) balances ops capacity with risk tolerance; see confusion matrix for expected alert volumes in the following slides.

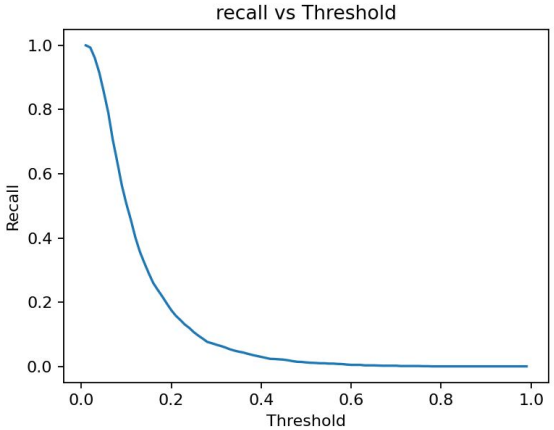




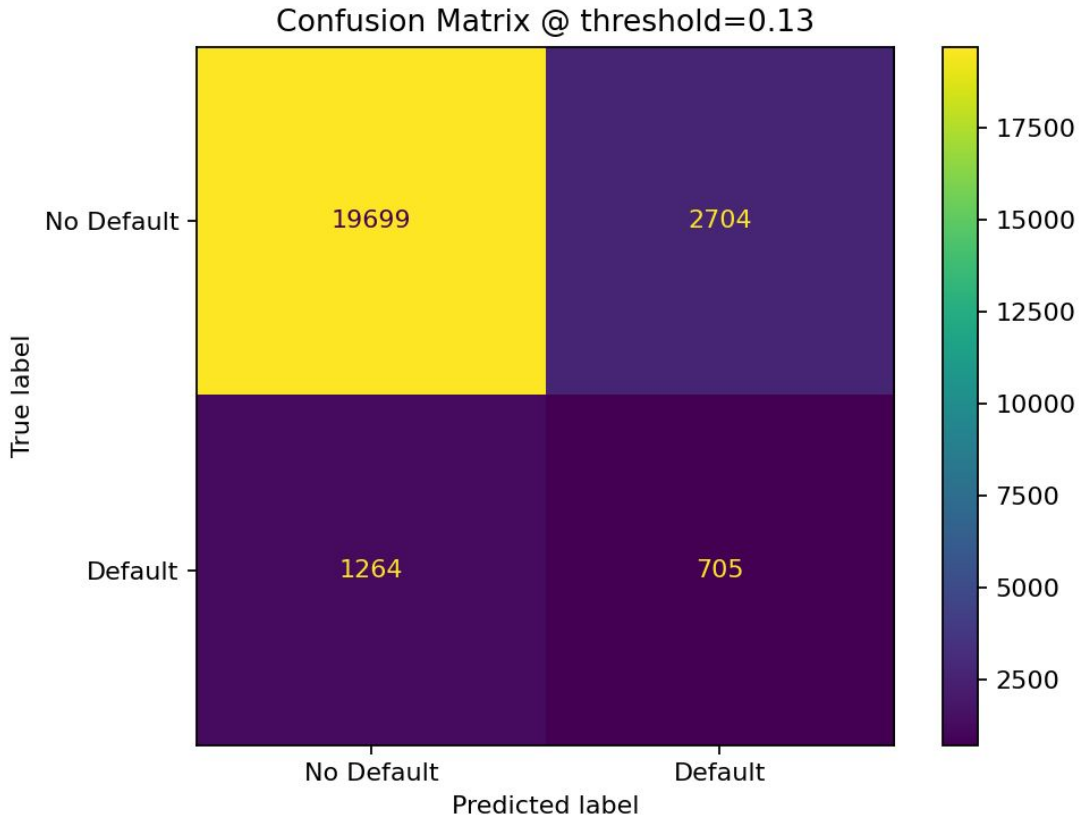
Evaluation Metrics- Business Interpretation:







Evaluation Metrics-Business Interpretation:

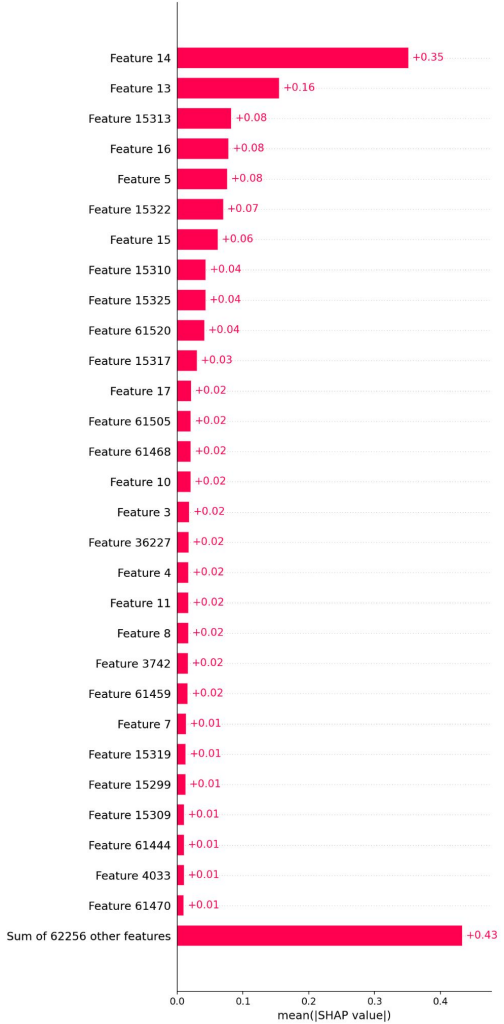


Feature Importance:

Using the SHAP importance value report of features, following conclusions can be drawn:

- ***Score_Source_2*** emerged as the most important driver of predictions (SHAP ≈ 0.35), indicating it contributes more than double the next feature.
- ***Score_Source_1*** also played a strong role (SHAP ≈ 0.15), showing the relevance of multiple scoring sources in the model.
- Other significant factors include ***Graduation status, Phone number change***, and ***House ownership age***, highlighting the influence of demographic and behavioral attributes.

Full SHAP value report csv file can be accessed here.



Drift Analysis:

Dataset Drift

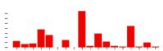
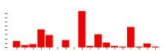
Dataset Drift is NOT detected. Dataset drift detection threshold is 0.5

40	0	0.0
Columns	Drifted Columns	Share of Drifted Columns

Data Drift Summary

Drift is detected for 0.0% of columns (0 out of 40).

Search

Column	Type	Reference Distribution	Current Distribution	Data Drift	Stat Test	Drift Score
Score_Source_3	cat			Not Detected	Jensen-Shannon distance	0.084708
Type_Organization	cat			Not Detected	Jensen-Shannon distance	0.020225
Own_House_Age	num			Not Detected	Wasserstein distance (normed)	0.018277
Score_Source_1	num			Not Detected	Wasserstein distance (normed)	0.013405
Client_Income	num			Not Detected	Wasserstein distance (normed)	0.011781
Client_Family_Members	num			Not Detected	Wasserstein distance (normed)	0.01161
Client_Occupation	cat			Not Detected	Jensen-Shannon distance	0.011214

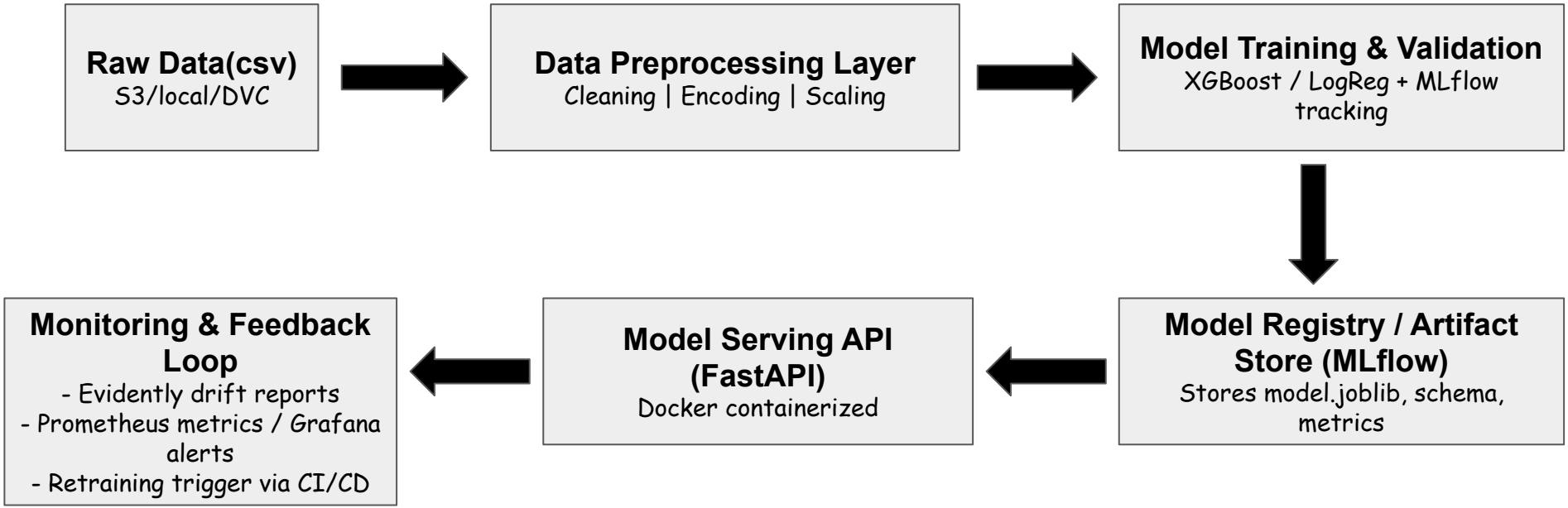
Drift Analysis:

Key Insights from Drift Report:

- **No drift detected:** Out of 40 monitored features, **0 showed significant data drift** (threshold = 0.5).
- **Stable distributions:** Both categorical (e.g., *Score_Source_3, Type_Organization*) and numerical features (e.g., *Own_House_Age, Client_Income*) remain statistically consistent between reference and current datasets.
- **Low drift scores:** The highest observed drift score is ~0.08, well below the alert threshold, indicating **model inputs remain stable and reliable**.
- **Target variable (Default) shows no significant drift** *(image attached below)* **— the mean of the current dataset (red line) stays within the reference distribution variability (green band), confirming stable class distribution over time.**

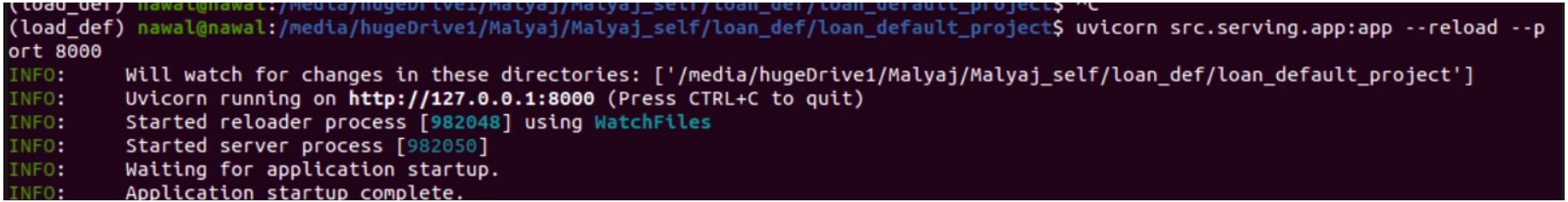
Full **Drift analysis HTML report** can be accessed here.

System Architecture Diagram:

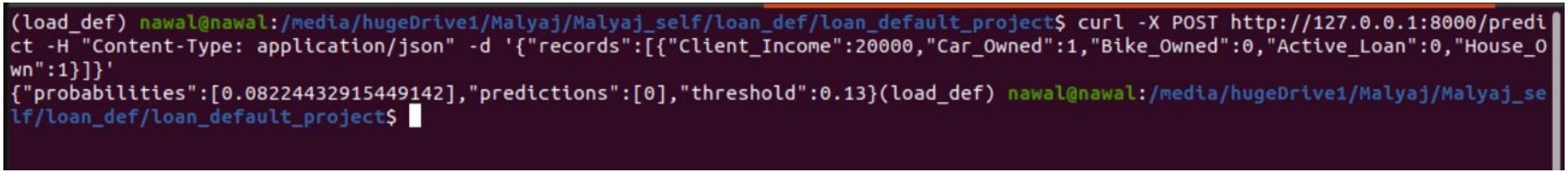


Deployment Plan : FastAPI Service

Terminal API running screenshot:



Sample curl response screenshot:



Deployment Summary:

- Containerized with Docker (fast startup, reproducible env)
- Served via FastAPI + Uvicorn
- Endpoints: /predict, /health

Deployment Plan : Containerization

Commands to run docker:

```
docker build -t loan-default:latest -f docker/Dockerfile .
docker run -p 8000:8000 loan-default:latest
```

Docker terminal during test run:

```
(load_def) nawal@nawal:/media/hugeDrive1/Malyaj/Malyaj_self/loan_def/loan_default_project$ docker run -p 8000:8000 loan-default:latest
INFO:     Started server process [1]
INFO:     Waiting for application startup.
/usr/local/lib/python3.11/pickle.py:1718: UserWarning: [15:58:07] WARNING: /workspace/src/collective/../data/../common/error_msg.h:82:
 or
configuration generated by an older version of XGBoost, please export the model by calling
`Booster.save_model` from that version first, then load it back in current version. See:

    https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html

for more details about differences between saving model and serializing.

  setstate(state)
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     172.17.0.1:34122 - "GET / HTTP/1.1" 404 Not Found
INFO:     172.17.0.1:34126 - "GET /favicon.ico HTTP/1.1" 404 Not Found
INFO:     172.17.0.1:55938 - "GET /docs HTTP/1.1" 200 OK
INFO:     172.17.0.1:55938 - "GET /openapi.json HTTP/1.1" 200 OK
```

CI/CD for ML Workflows:

Key steps:

- Version control with Git + GitHub Actions
- CI: lint, pytest, build Docker image, push artifact to registry
- CD: deploy container to staging → canary rollout → production
- Trigger retrain when drift > threshold or new data arrives
- Track experiments in MLflow

Monitoring, Canary & Governance:

Monitoring:

- Data Drift (Evidently)
- Model Metrics (Precision/Recall/F1, AUC)
- Infra Metrics (Latency, Error rate)
- Alerting via Prometheus/Grafana or Email

Canary Deployment:

- Route 10% traffic to new model
- Compare AUC/F1 in live data
- Auto-rollout if stable; rollback if degradation detected

Governance & Audit:

- MLflow stores parameters, metrics, artifacts
- DVC tracks data and preprocessing version
- Access control for production deployment

Load Testing & Scalability:

ApacheBench load-test Summary:

- **High Throughput:** API handled **1000 requests at 20 concurrent users** with **0 failed requests**, achieving **~3006 req/sec**.
- **Low Latency:** Median response time **5 ms**, 95% of requests completed within **10 ms**, max observed latency **50 ms**.
- **Stable Performance:** Consistent response distribution with efficient resource usage (411 KB/sec transfer rate).

ApacheBench full report screenshot —>

```
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        uvicorn
Server Hostname:        127.0.0.1
Server Port:            8000

Document Path:          /health
Document Length:        15 bytes

Concurrency Level:      20
Time taken for tests:   0.333 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      140000 bytes
HTML transferred:       15000 bytes
Requests per second:    3006.44 [#/sec] (mean)
Time per request:       6.652 [ms] (mean)
Time per request:       0.333 [ms] (mean, across all concurrent requests)
Transfer rate:          411.04 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
Processing:     3    7   7.1      5      49
Waiting:        3    6   7.1      5      49
Total:          3    7   7.1      5      50

Percentage of the requests served within a certain time (ms)
  50%      5
  66%      5
  75%      5
  80%      5
  90%      6
  95%     10
  98%     44
  99%     47
 100%     50 (longest request)
```

Auditability & Versioning:

MLflow:

- Each experiment run stores parameters, metrics, and model artifacts in MLflow. (Sample folder structure attached on the right side)
- This provides complete lineage for every model version.

```
mlruns/0/<run_id>/
├── metrics/
├── params/
├── artifacts/
├── tags/
```

DVC : for data versioning:

- Data can be versioned using DVC to ensure consistent train-test splits across experiments.

The End