Prêt à dépenser

Credit Scoring

GitHub Repository

Ovidiu PASCAL

Context & Mission

→ **"Prêt à Dépenser"** is a financial company offering consumer credit to individuals with minimal or no loan history. The company aims to broaden the financial inclusion for the unbanked population.

With this intent in mind, financial, statutory & behavioral data were acquired to bridge the lack of loan history to:

- Build a classifier model which accepts or reject a credit application
- Make the model inference available through an API
- (Future Work) provide a Dashboard which:
  - Display the model decision
  - Assist customer relation managers in transparency by explaining the main factors that influenced the model decision
  - Facilitates navigation through customers's data

Constraints

Embrace a MLOps approach

- ML Flow → Track experiments, store models in a registry, and test ML serving.
- Git & Github → Manage and share code, integrate continuously.
- CI/CD with GitHub Actions → Automate API deployment in the cloud.
- Testing → Design and automate unit tests using PyTest
- Logging → Through the Loguru library

Deploy the API

Test evidently → As a tool to detect datadrift when the model is in production

Document the steps explored to build the classifier

Plan

Prêt à dépenser

Modeling Approach & Results

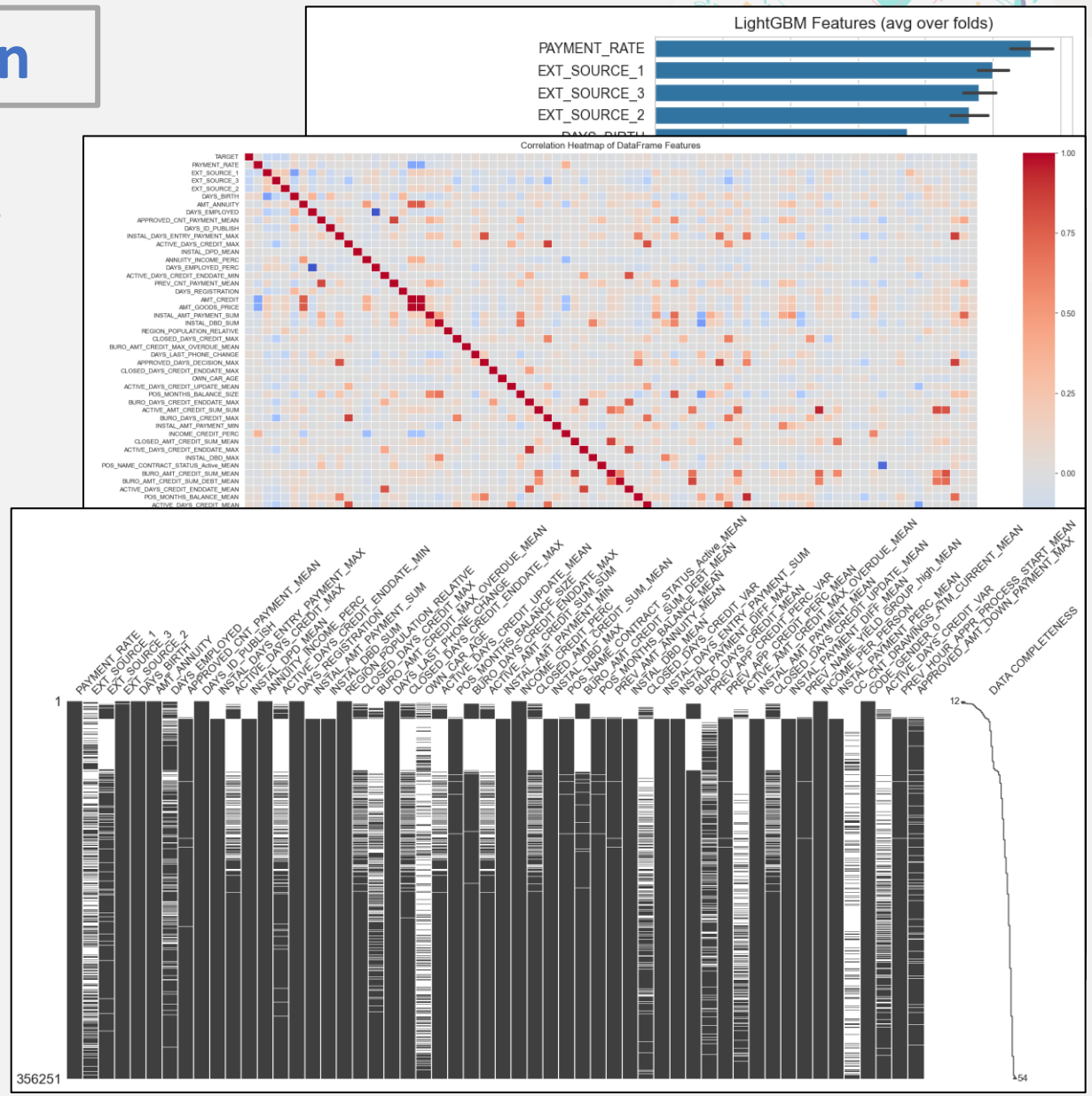
Feature engineering + Data Preparation

Feature Engineering → LightGBM with Simple Features (Kaggle)

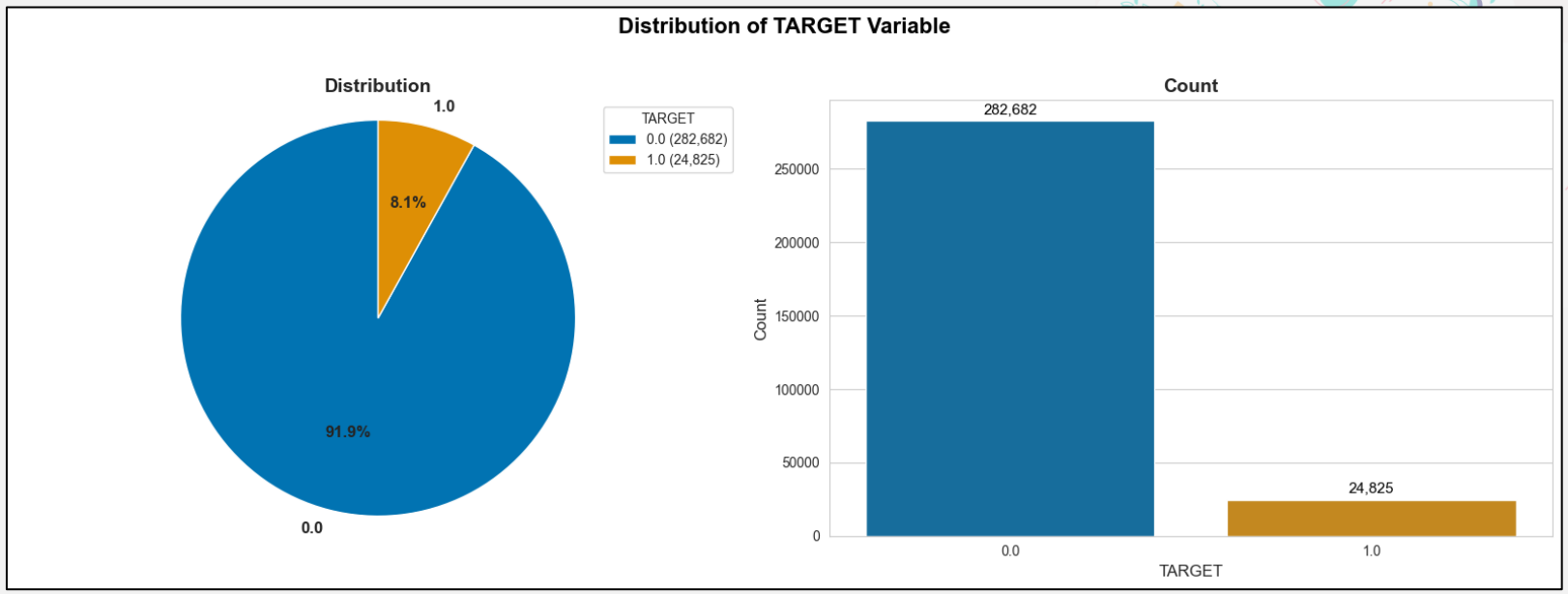
- One hot categorical, Numerical Aggregation/Group By
- (356 000 individuals, 800 features)
- Retained the 80 most important features
- Highly correlated pairs deleted through lowest importance
- Missing Values Imputation → Facebook AI Similarity Search (FAISS) KNN + median because of outliers

→ (356 000 individuals, 50 features)

- Only numerical variables, no categorical ones
- Skewed Distributions
- High Range between Min/Max values
- High count of outliers



Target Distribution



Imbalance ratio → 11.38

Problem → Classifiers tend to predict the dominant class

3 Techniques used to address this issue

- **Weight Class Errors** → Increase the impact of errors made on the minority class when evaluating & decrease the ones made on the majority class
- **Ressampling through** SMOTETomek → Combine over-sampling on the minority class using SMOTE & under-sampling on the majority class using Tomek links to get closer to class balance
- **Threshold optimization** → Finding the best threshold value to decide the cutoff class from probability prediction which default to 0,5.

Model Evaluation & Business Considerations

False Negative (FN) → Real loss of money, a client wrongly seen as capable of paying
False Positive (FP) → Hypothetical loss of money, a client capable of paying but seen as not capable

Assumption → *"The cost incurred by a FN is ten times that of the cost incurred by a FP"*

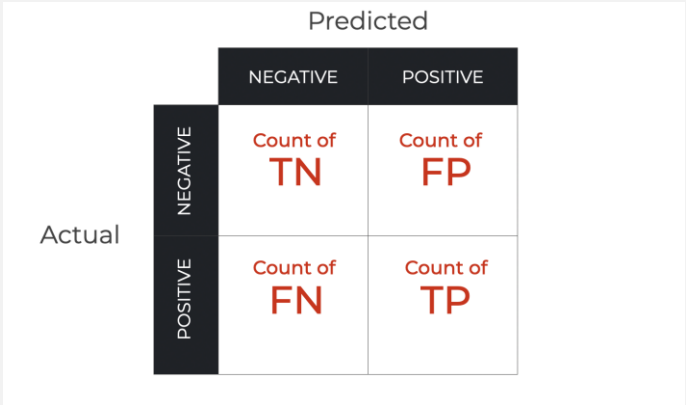
$$\text{Business Cost} = (FN \times 10) + FP$$

For the finetuning → *We adjusted the cost function to dynamically increase false negative penalties when recall is low, ensuring accurate identification of default risks*

$$\text{Business Cost} = \left(FN \times 10 \times \frac{1}{\text{Recall}}\right) \times FP$$

$$\text{Profit} = (6 \times TN - 4 \times FP) - (40 \times FN + 0 \times TP)$$

$$\text{Recall} = \frac{TP}{TP + FN} \qquad \text{FN Rate} = 1 - \text{Recall}$$



Finding the Best Model

Tested Models

- Dummy Classifier (Baseline)
- Logistic Regression (solver: saga/sag & Penalty :Lasso, Ridge, ElacticNet)
- RandomForest Classifier
- XGBoost Classifier
- LightGBM Classifier

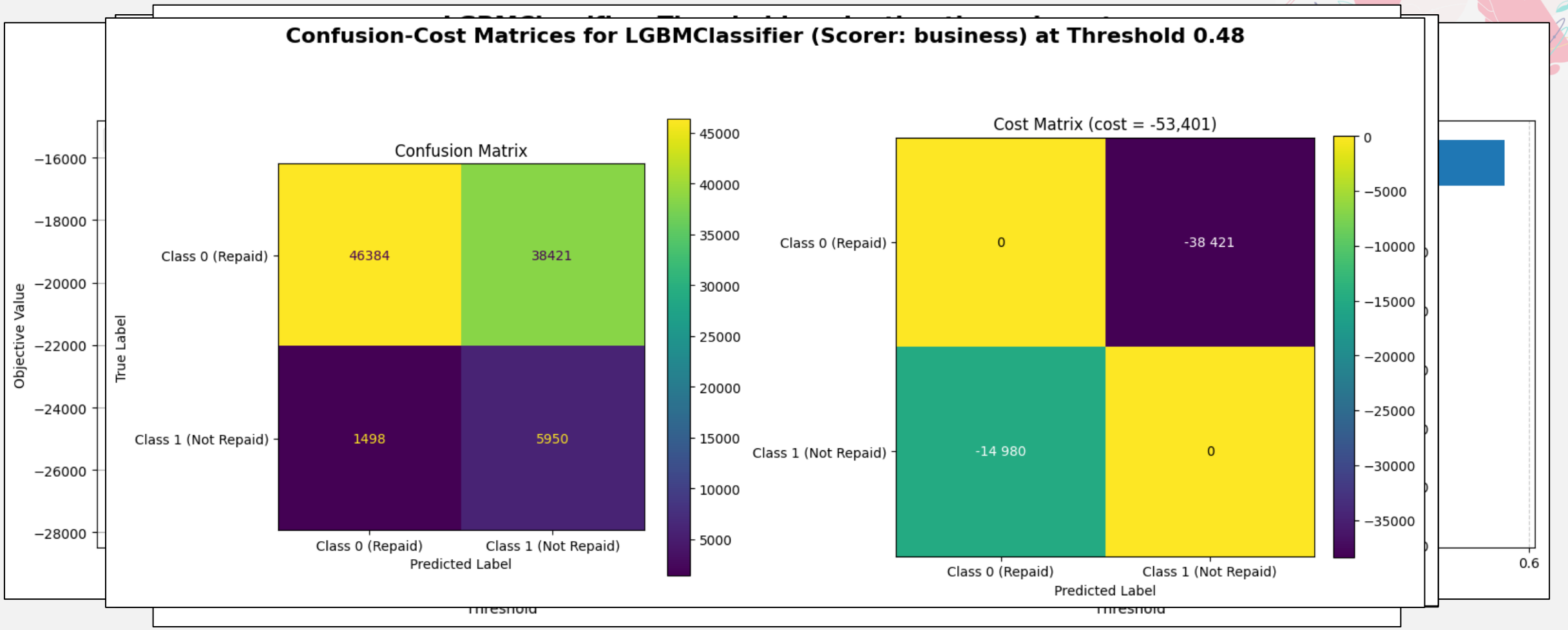
Best Hyperparameters Search with Optuna

→ **Sampled 20%** of the original dataset = 61 501 individuals
→ **Preprocessing Pipeline :** RobustScaler + SMOTETomek ("Sampling Strategy" : [0.15, 0.25, 0.30, 0.35])
→ **Evaluation:** StratifiedKFold (5 splits) + Cross Validation Score
→ **Scorer:** Maximized (ROC AUC; - Business Cost)
→ **Sampler:** *TPE* (Tree-structured Parzen Estimator) algorithm
→ **Trials:** 15 & 45 for the Best Model

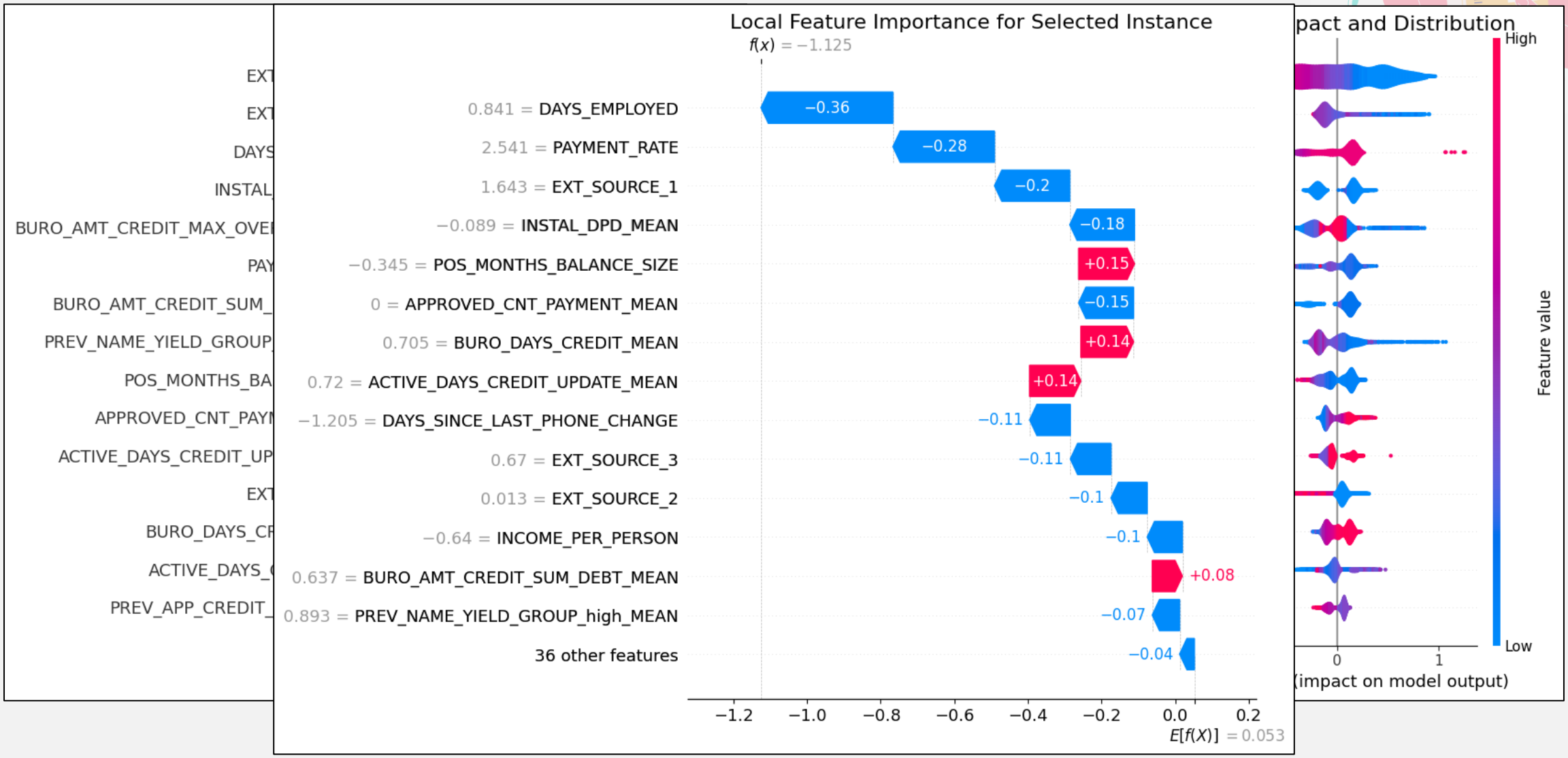
ML Flow → Recorded each run, save plots as artifacts and register the fitted model inside the Model Registry

Metrics → Precision/Recall, FNR, F2 Beta Score (Beta =2), ROC AUC, FN, FP, Business Cost, Profit, Business Cost Std

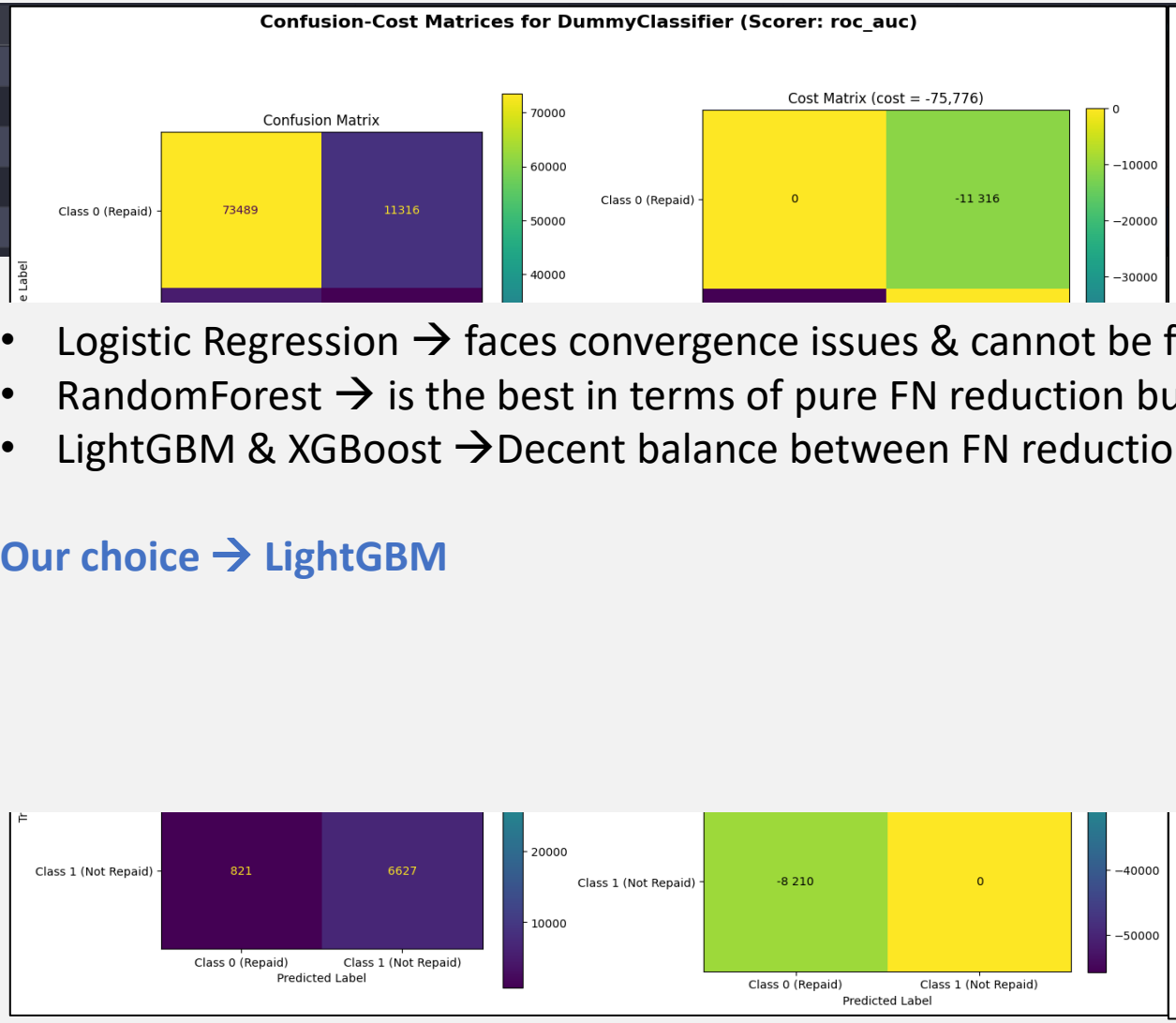
LightGBM

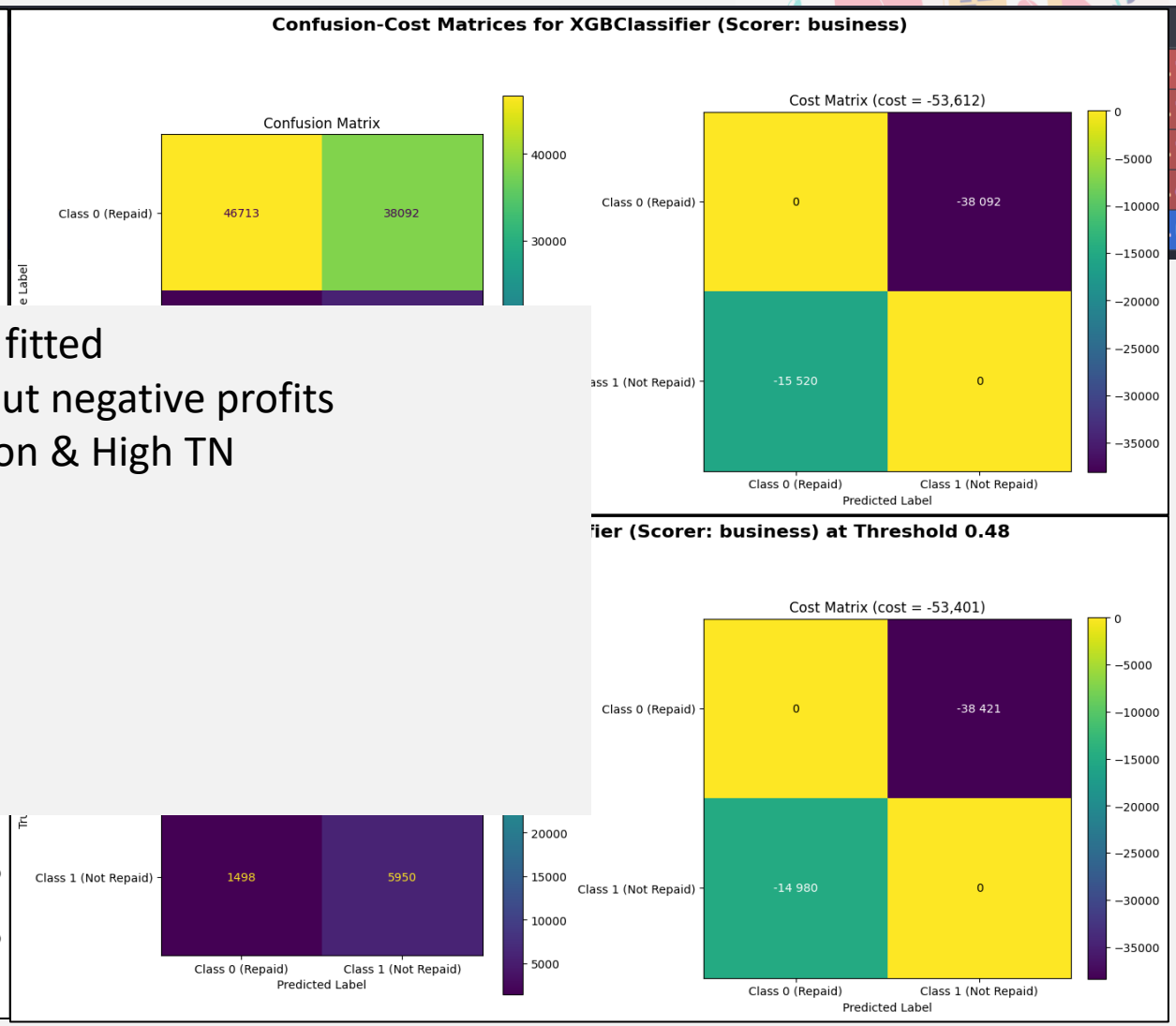


Feature Importance



Summary

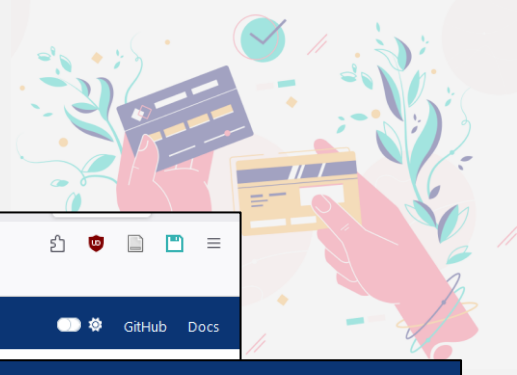


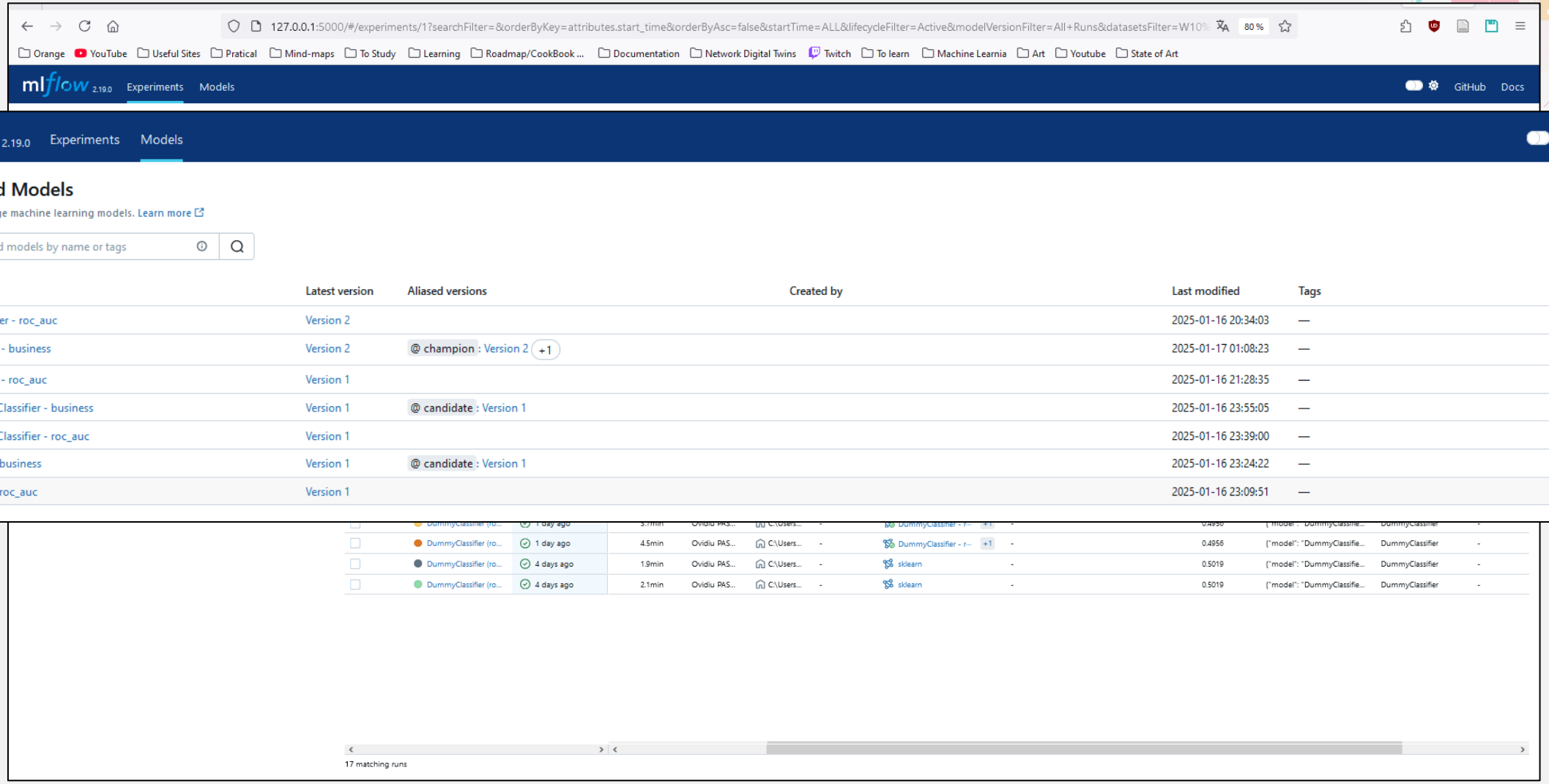


- Logistic Regression → faces convergence issues & cannot be fitted
- RandomForest → is the best in terms of pure FN reduction but negative profits
- LightGBM & XGBoost →Decent balance between FN reduction & High TN

Our choice → LightGBM

ML Flow





mlflow 2.19.0 Experiments Models GitHub Docs

Registered Models

Share and manage machine learning models. Learn more

Create Model

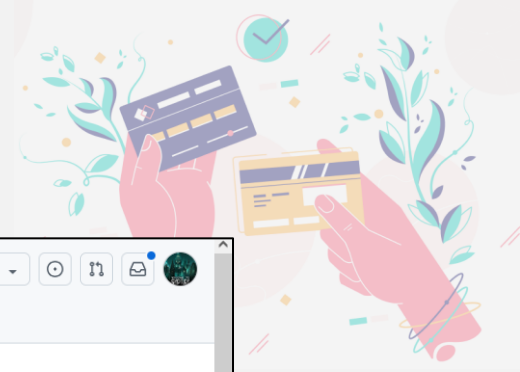
Filter registered models by name or tags

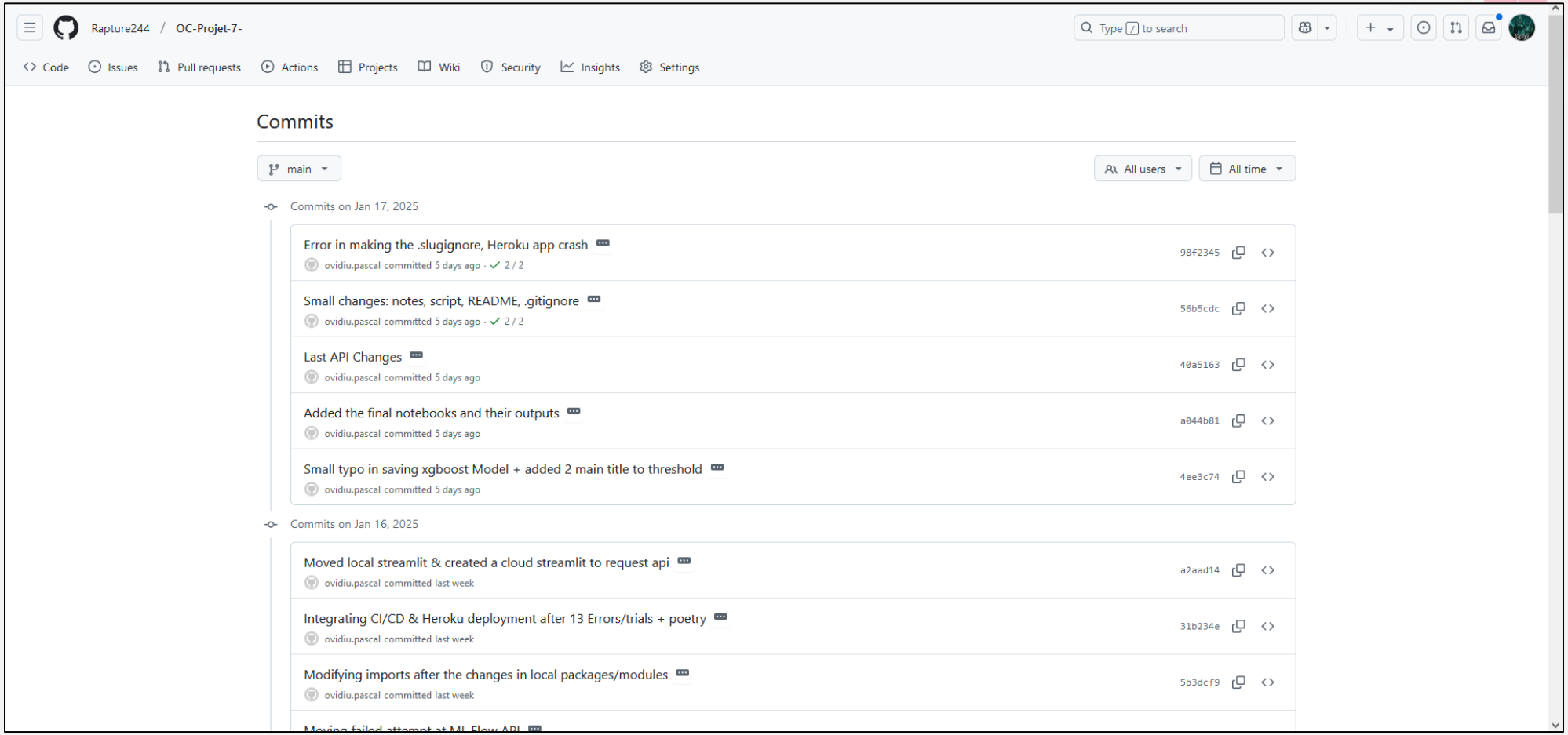
Name	Latest version	Aliased versions	Created by	Last modified	Tags
DummyClassifier - roc_auc	Version 2			2025-01-16 20:34:03	—
LGBMClassifier - business	Version 2	@ champion : Version 2 +1		2025-01-17 01:08:23	—
LGBMClassifier - roc_auc	Version 1			2025-01-16 21:28:35	—
RandomForestClassifier - business	Version 1	@ candidate : Version 1		2025-01-16 23:55:05	—
RandomForestClassifier - roc_auc	Version 1			2025-01-16 23:39:00	—
XGBClassifier - business	Version 1	@ candidate : Version 1		2025-01-16 23:24:22	—
XGBClassifier - roc_auc	Version 1			2025-01-16 23:09:51	—

Prêt à dépenser

CI/CD Pipeline

Code Versionning



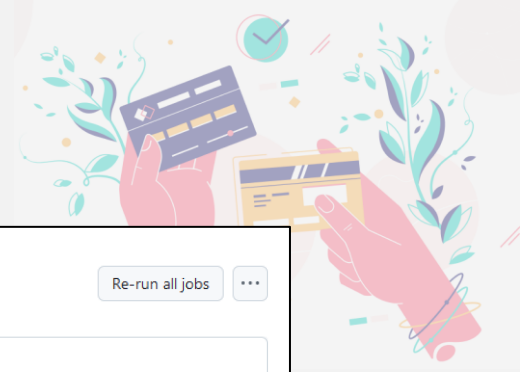


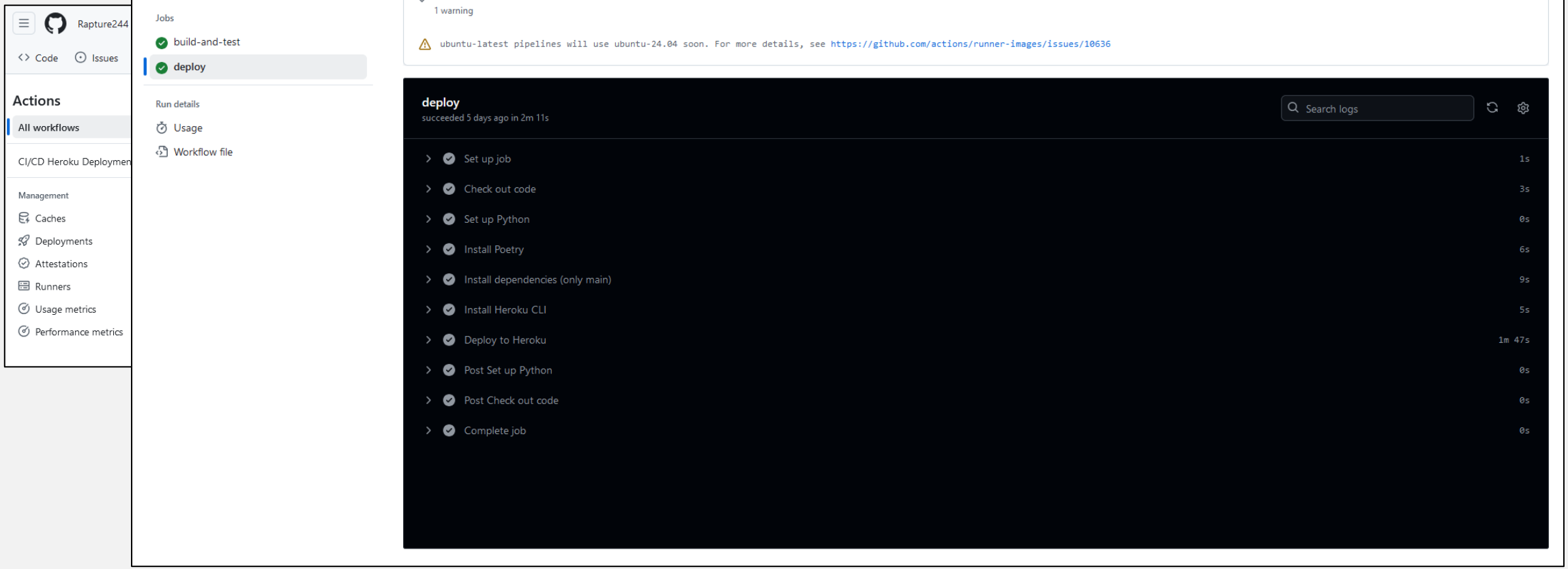
Tests made with PyTest

```
"""
This test suite validates the behavior and robustness of the `/predict` endpoint in the Flask API.

Key Features:
1. Utilizes `pytest` fixtures to create a reusable Flask test client for API requests.
2. Covers multiple scenarios to ensure comprehensive testing of the `/predict` endpoint.

Test Cases:
- `test_predict_valid_id`: Verifies correct handling of valid IDs that exist in the dataset.
- `test_predict_invalid_id`: Ensures appropriate error handling for IDs that do not exist in the dataset.
- `test_predict_missing_id`: Tests the API's response to payloads missing the required `id` field.
- `test_predict_invalid_payload`: Validates error handling for invalid, non-JSON payloads.
- `test_api_availability`: Confirms the API responds correctly to undefined endpoints.

Dependencies:
- Pytest: Framework for writing and running tests.
- Flask: Framework providing the API to test.
- JSON: Used for crafting request payloads and interpreting responses.

Notes:
- Ensure the `api.local_main` module and the Flask app are correctly configured and imported.
- Adjust `payload` values in test cases as needed to match the data in the actual dataset.
- This test suite assumes `/predict` is the only defined route; update the `test_api_availability` case if other endpoints exist.

"""
```

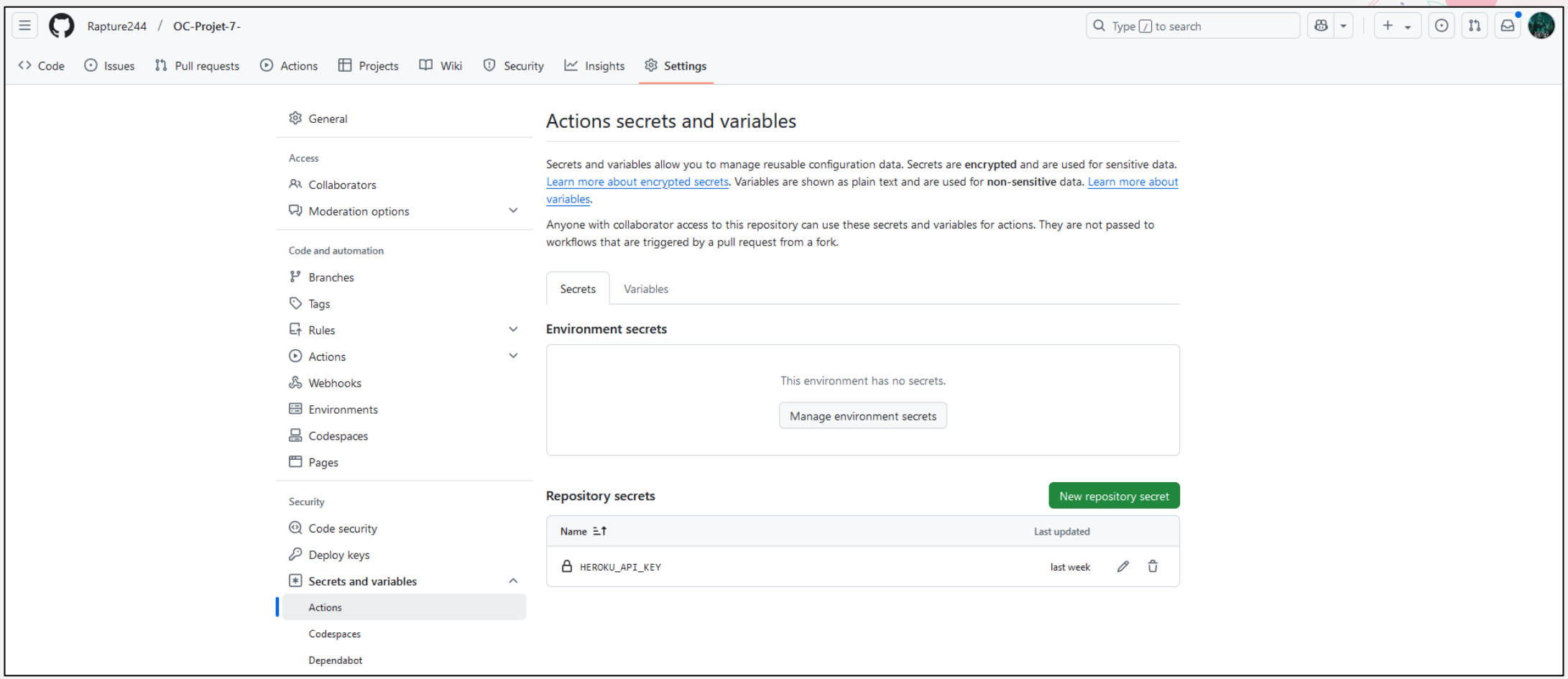
```python
# ================================================== IMPORTS ================================================== #
import pytest
from api.local_main import app


# ============================================================================================================== #
#                                                    TEST                                                        #
# ============================================================================================================== #
@pytest.fixture
def client():
    """
    Creates a test client for the Flask app.
    """
    with app.test_client() as client:
        yield client

def test_predict_valid_id(client):
    """
    Test case for a valid ID that exists in the dataset.
    """
    payload = {"id": 100001}  # Ensure this ID exists in the dataset as an integer
    response = client.post("/predict", json=payload)
    assert response.status_code == 200
    data = response.get_json()
    assert data is not None
    assert "SK_ID_CURR" in data
    assert data["SK_ID_CURR"] == payload["id"]
    assert "predicted_proba" in data
    assert "predicted_target" in data
    assert "status" in data

def test_predict_invalid_id(client):
    """
    Test case for an invalid ID that does not exist in the dataset.
    """
    payload = {"id": 999999}  # Non-existent ID as an integer
    response = client.post("/predict", json=payload)
    assert response.status_code == 404
    data = response.get_json()
    assert data is not None
    assert "error" in data
    assert "not found" in data["error"].lower()

def test_predict_missing_id(client):
    """
    Test case for a missing 'id' field in the payload.
    """
    payload = {}  # Missing 'id'
    response = client.post("/predict", json=payload)
    assert response.status_code == 400
    data = response.get_json()
    assert data is not None
    assert "error" in data
    assert "missing or invalid 'id'" in data["error"].lower()

def test_predict_invalid_payload(client):
    """
    Test case for an invalid payload that is not JSON.
    """
    response = client.post("/predict", data="not a json payload")
    assert response.status_code == 400
    data = response.get_json()
    assert data is not None
    assert "error" in data
    assert "invalid json payload" in data["error"].lower()

def test_api_availability(client):
    """
    Test case for accessing an undefined endpoint.
    """
    response = client.get("/")  # Assuming no endpoint at "/"
    assert response.status_code == 404

```

GitHub Actions to build, test & deploy the API

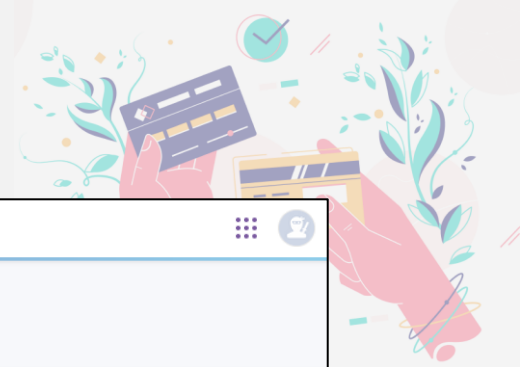




Used GitHub Secrets To Secure the Connection to Heroku



API Deployment on Heroku



HEROKU from Salesforce

Jump to Favorites, Apps, Pipelines, Spaces...

Personal > credit-score-attribution

☆ Open app More

GitHub Rapture244/OC-Projet-7-

Overview Resources Deploy Metrics Activity Access Settings

Activity Feed > Build Log

ID 6a600503-5b7c-4a52-b124-ec2f904483e7

```
-----> Installing llvm-14-runtime_1%3a14.0.0-1ubuntu1.1_amd64.deb
-----> Installing llvm-14-tools_1%3a14.0.0-1ubuntu1.1_amd64.deb
-----> Installing python3-pkg-resources_59.6.0-1.2ubuntu0.22.04.2_all.deb
-----> Installing python3-pygments_2.11.2+dfsg-2ubuntu0.1_all.deb
-----> Installing python3-yaml_5.4.1-1ubuntu1_amd64.deb
-----> Writing profile script
-----> Rewrite package-config files
-----> Python app detected
-----> Using Python 3.12 specified in .python-version
-----> Restoring cache
-----> Using cached install of Python 3.12.8
-----> Using cached Poetry 2.0.1
-----> Installing dependencies using 'poetry sync --only main'
       Installing dependencies from lock file

       No dependencies to install or update

       Installing the current project: p7-package (0.1.0)
-----> Discovering process types
       Procfile declares types -> web
-----> Compressing...
       Done: 393.3M
-----> Launching...
 !     Warning: Your slug size (393 MB) exceeds our soft limit (300 MB) which may affect boot time.
       Released v11
       https://credit-score-attribution-003464da4de3.herokuapp.com/ deployed to Heroku
This app is using the Heroku-22 stack, however a newer stack is available.
To upgrade to Heroku-24, see:
https://devcenter.heroku.com/articles/upgrading-to-the-latest-stack
```

Build finished

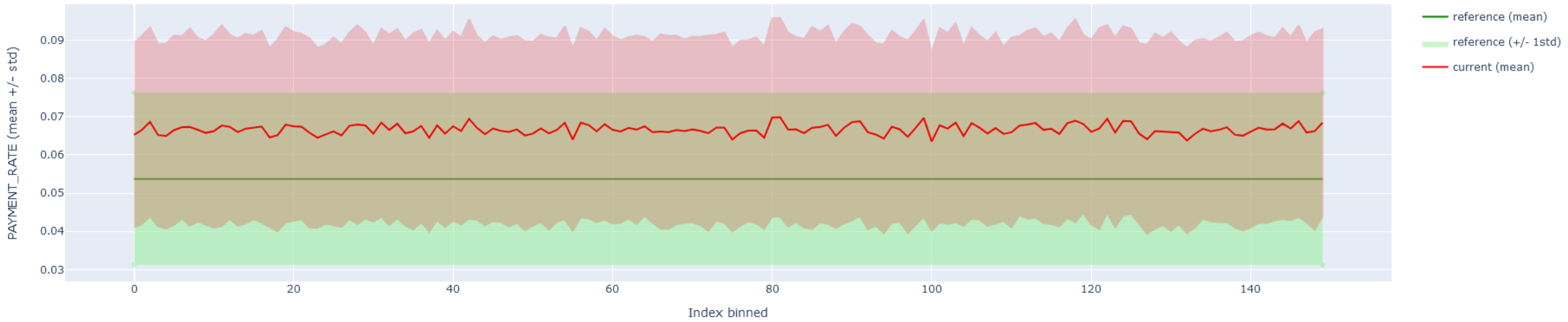
Prêt à dépenser

Data Drift

Data Drift

Column	Type	Reference Distribution	Current Distribution	Data Drift	Stat Test	Drift Score
PAYMENT_RATE	num			Detected	Wasserstein distance (normed)	0.574751

DATA DRIFT DATA DISTRIBUTION



Prêt à dépenser

Live Demonstration

- ML Flow UI
- Streamlit test of the API + Heroku Log of the app

Prêt à dépenser

Q&A