

## Lab 1 : Du notebook au mini-système production-ready

Réalisée par :

El Hamzaoui Aya

## 1. Introduction

Le but de ce lab est de mettre en place un pipeline MLOps minimal pour un modèle de prédition de churn. Les objectifs principaux sont :

1. Préparer et entraîner un modèle scikit-learn sur un dataset de churn.
2. Evaluer et versionner le modèle via un registry local (metadata.json et current\_model.txt).
3. Déployer le modèle via une API FastAPI /predict.
4. Monitorer les requêtes pour détecter un **data drift**.
5. Gérer les versions et effectuer des rollback en cas de problème.

```
PS C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mllops-lab-01> mkdir models
●

Répertoire : C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mllops-lab-01

Mode                LastWriteTime        Length Name
----              -----          ----
d----       14/12/2025      15:38           models

PS C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mllops-lab-01> mkdir registry
●

Répertoire : C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mllops-lab-01

Mode                LastWriteTime        Length Name
----              -----          ----
d----       14/12/2025      15:38           registry

PS C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mllops-lab-01> mkdir logs
```

```
PS C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mlops-lab-01> echo "" > registry\current_model.txt
PS C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mlops-lab-01> python -m venv venv_mlops
PS C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mlops-lab-01> pip install pandas numpy scikit-learn fastapi uvicorn joblib
Requirement already satisfied: pip in c:\users\pc\desktop\master_sdia\s3\mlops\mlps-lab-01\venv_mlops\lib\site-packages (25.3)
(venv_mlops) PS C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mlops-lab-01>
(venv_mlops) PS C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mlops-lab-01> pip install pandas numpy scikit-learn fastapi uvicorn joblib
Collecting pandas
  Using cached pandas-2.3.3-cp313-cp313-win_amd64.whl.metadata (19 kB)
Collecting numpy
  Using cached numpy-2.3.5-cp313-cp313-win_amd64.whl.metadata (60 kB)
Collecting scikit-learn
  Using cached scikit_learn-1.8.0-cp313-cp313-win_amd64.whl.metadata (11 kB)
Collecting fastapi
  Using cached fastapi-0.124.4-py3-none-any.whl.metadata (38 kB)
Collecting uvicorn
  Using cached uvicorn-0.38.0-py3-none-any.whl.metadata (6.8 kB)
Collecting joblib
  Using cached joblib-1.5.2-py3-none-any.whl.metadata (5.6 kB)
```

◀ MLOPS-LAB-01

```
> data  
> logs  
> models  
> registry  
> src  
> venv_mlops
```

The screenshot shows a Jupyter Notebook environment with the following details:

- File Explorer:** Shows a project structure with folders like 'data', 'raw.csv', 'logis', 'models', 'registry', and 'src'. Inside 'src', there is a file named 'generate\_data.py'.
- Code Editor:** The content of 'generate\_data.py' is displayed:

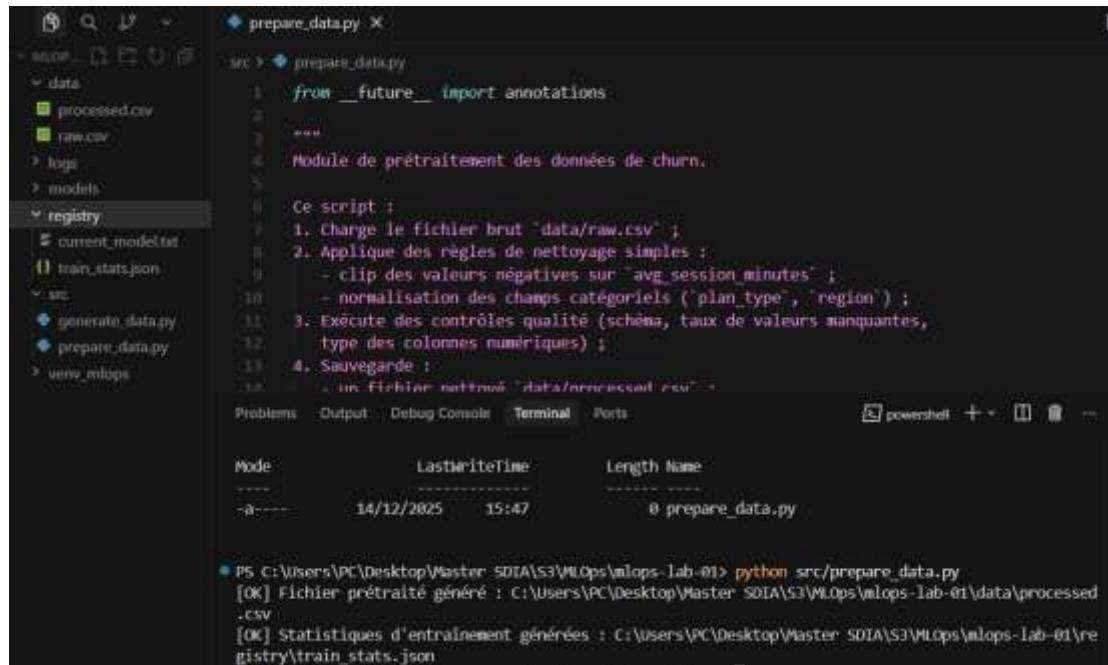
```
def generate_churn_dataset(n: int, seed: int = 42) -> pd.DataFrame:
    # Modèle Logistique : combinaison linéaire des features
    base_logit = (
        1.2
        + 0.55 * num_complaints
        - 0.03 * tenure_months
        - 0.02 * avg_session_minutes
        + np.where(plan_type == "premium", -0.35, 0.0)
    )
    ...
    # Effet de la région sur le logit (ajustements additifs)
    base_logit += np.where(region == "EU", -0.05, 0.0)
    base_logit += np.where(region == "AF", 0.08, 0.0)
    ...
    # Passage en probabilité via la sigmoïde
    churn_proba = sigmoid(base_logit)
```
- Toolbar:** Includes tabs for 'Problems', 'Output', 'Debug Console', 'Terminal', and 'Ports'. It also has icons for 'powerhell', 'terminal', and 'file' operations.
- Terminal:** Displays command-line output:

```
PS C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mllops-lab-01>
PS C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mllops-lab-01>
PS C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mllops-lab-01> python src/generate_data.py
[OK] Dataset généré : C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mllops-lab-01\data\raw.csv (rows=1200,
seed=42)
PS C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mllops-lab-01>
```

## 2. Préparation des données et statistiques

### 2.1 Prétraitement

- Les données brutes ont été nettoyées et transformées via `prepare_data.py`.
- Les features numériques ont été standardisées et les catégorielles encodées.
- Les statistiques d'entraînement (moyenne, écart-type) ont été sauvegardées dans :



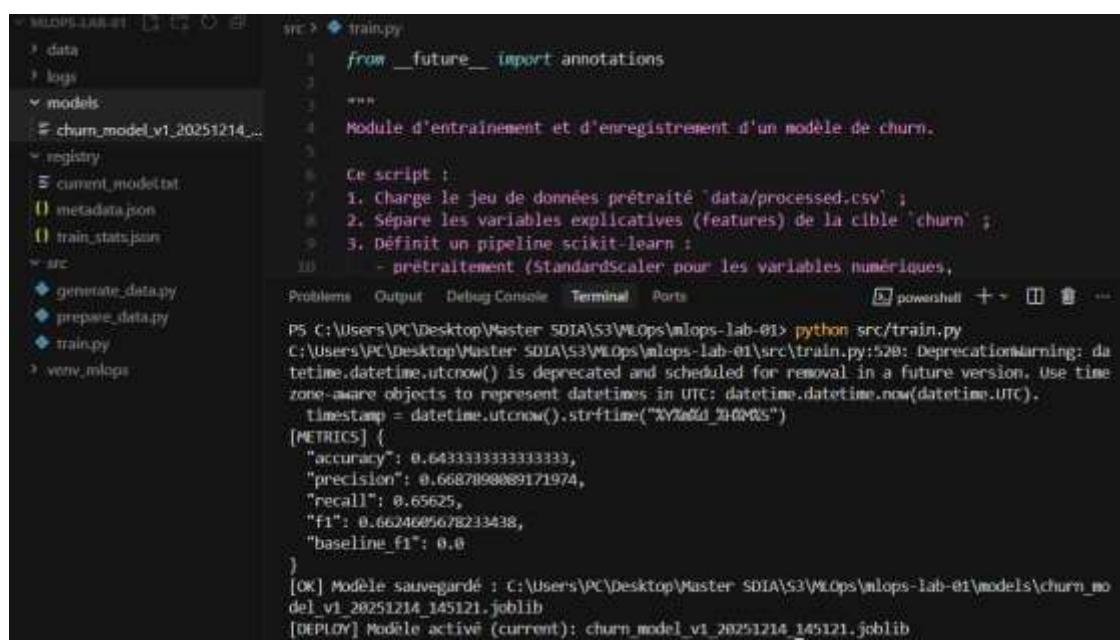
```
prepare_data.py
from __future__ import annotations

Module de prétraitement des données de churn.

Ce script :
1. Charge le fichier brut 'data/raw.csv' ;
2. Applique des règles de nettoyage simples :
   - clip des valeurs négatives sur 'avg session minutes' ;
   - normalisation des champs catégoriels ('plan type', 'region') ;
3. Exécute des contrôles qualité (schéma, taux de valeurs manquantes,
   type des colonnes numériques) ;
4. Sauvegarde :
   - un fichier nettoyé "data/processed.csv" ;
   - des statistiques d'entraînement "train_stats.json" .
```

Mode	LastWriteTime	Length	Name
-a----	14/12/2025	15:47	0 prepare_data.py

```
PS C:\Users\PC\Desktop\Master SDIA\S3\MLops\mllops-lab-01> python src\prepare_data.py
[OK] Fichier prétraité généré : C:\Users\PC\Desktop\Master SDIA\S3\MLops\mllops-lab-01\data\processed.csv
[OK] Statistiques d'entraînement générées : C:\Users\PC\Desktop\Master SDIA\S3\MLops\mllops-lab-01\registry\train_stats.json
```



```
train.py
from __future__ import annotations

Module d'entraînement et d'enregistrement d'un modèle de churn.

Ce script :
1. Charge le jeu de données prétraité 'data/processed.csv' ;
2. Sépare les variables explicatives (features) de la cible 'churn' ;
3. Définit un pipeline scikit-learn :
   - prétraitement (StandardScaler pour les variables numériques,
```

Mode	LastWriteTime	Length	Name
-a----	14/12/2025	15:47	0 train.py

```
PS C:\Users\PC\Desktop\Master SDIA\S3\MLops\mllops-lab-01> python src\train.py
C:\Users\PC\Desktop\Master SDIA\S3\MLops\mllops-lab-01\src\train.py:529: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use time zone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
    timestamp = datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S")
[METRICS] {
    "accuracy": 0.6433333333333333,
    "precision": 0.668789889171974,
    "recall": 0.65625,
    "f1": 0.6624685678233438,
    "baseline_f1": 0.6
}
[OK] Modèle sauvegardé : C:\Users\PC\Desktop\Master SDIA\S3\MLops\mllops-lab-01\models\churn_no_de_l_v1_20251214_145121.joblib
[DEPLOY] Modèle activé (current): churn_model_v1_20251214_145121.joblib
```

### 3. Entraînement et évaluation du modèle (evaluate.py)

#### 3.1 Pipeline scikit-learn

Le pipeline inclut :

- **StandardScaler** sur les features numériques : tenure\_months, num\_complaints, avg\_session\_minutes.
- **OneHotEncoder** sur les features catégorielles : plan\_type, region.
- **LogisticRegression** comme classificateur.

#### 3.2 Découpage train/test

- Stratification sur la cible churn.
- Test set = 25% des données.

#### 3.3 Tuning du seuil optimal

- La probabilité de churn est convertie en prédiction binaire selon un **seuil optimisé pour la F1**.
- Baseline triviale : prédire toujours 0 pour comparer.

#### 3.4 Sauvegarde et registry

- Le modèle est sauvegardé dans models/ avec un timestamp et la version.
- Les métadonnées sont mises à jour dans registry/metadata.json.
- Si la F1 dépasse le **gate (0.70)** et bat la baseline, le modèle devient le current\_model.txt.

```
src/evaluate.py ④ evaluate.py
> data
> logs
< models
  - churn_model_v1_20251214_14...
  - churn_model_v1_20251214_14...
< registry
  - current_model.txt
  - metadata.json
  - eval_stats.json
< src
  - evaluate.py
  - generate_data.py
  - prepare_data.py
  - main.py
  - venvScripts
> .
[OK] Modèle sauvegardé : C:/Users/PC/Desktop/Master-SDA/S3/MLops/ValOps-1ab-01/models/churn_model_v1_20251214_145306.joblib
[INFO] Modèle actuel : churn_model_v1_20251214_145306.joblib
```

python src/evaluate.py

c:\users\pc\desktop\master-sda\s3\mlops\valops-1ab-01\src\evaluate.py:292: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for re: re.datetime.datetime.utcnow()

is deprecated and scheduled for removal. timezone.timezone-aware objects se se timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC)

timestamp = datetime.utcnow().strftime("%Y%m%d\_%H%M%S")

[METRICS]

"accuracy": 0.6433333333333333,

"precision": 0.66078900009171974,

"recall": 0.66625,

"F1\_threshold\_05": 0.6624695678233438,

"F1": 0.7164179184477612,

"best\_threshold": 0.36,

"baseline\_F1": 0.0

[OK] Modèle sauvegardé : C:/Users/PC/Desktop/Master-SDA/S3/MLops/ValOps-1ab-01/models/churn\_model\_v1\_20251214\_145306.joblib

[INFO] Modèle actuel : churn\_model\_v1\_20251214\_145306.joblib

## 4. Déploiement via API FastAPI (api.py)

The screenshot shows the PyCharm interface with the file structure on the left and the code editor on the right. The code in `api.py` defines a `predict` function that takes a `PredictRequest` and returns a dictionary. It uses a model to predict probabilities and then rounds them to 0 or 1. An exception is raised if the prediction fails.

```
def predict(req: PredictRequest) -> dict[str, Any]:  
    start = time.perf_counter()  
    try:  
        proba = float(model.predict_proba(x_df)[e][1])  
        pred = int(proba >= 0.5) # seuil fixe à 0.5 (peut être tuné)  
    except Exception as exc:  
        raise HTTPException(  
            status_code=400,  
            detail=f"Erreur de prédiction : {exc}",  
        )
```

Terminal output:

```
Successfully installed annotated-doc-0.0.4 annotated-types-0.7.0 anyio-4.12.0 click-a.3.1.0  
colorama-0.4.6 fastapi-0.124.0 h11-0.16.0 idna-3.11 joblib-1.5.2 numpy-1.23.5 pandas-2.3.3 py  
dantic-2.12.5 pydantic-core-1.41.5 python-dateutil-2.9.0 pytz-2025.2 scikit-learn-1.8  
0 scipy-1.16.3 six-1.17.0 starlette-0.50.0 threadpoolctl-3.6.0 typing-extensions-4.15.0 ty  
ping-inspection-0.4.2 tzdata-2025.3 uvicorn-0.38.0  
(venv) C:\Users\PC\Desktop\Master SDIA\S3\MLops\mllops-lab-01> uvicorn src.api:app  
--reload  
INFO: Will watch for changes in these directories: ['C:\Users\PC\Desktop\Master SDIA\S3\MLops\mllops-lab-01']  
INFO: uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)  
INFO: started reloader process [14892] using StatReload  
INFO: started server process [25768]  
INFO: Waiting for application startup.  
INFO: Application startup complete.
```

### 4.1 Endpoints

#### 1. GET /health

- Vérifie la présence du modèle courant et retourne son nom.

The screenshot shows the Postman interface for the `/health` endpoint. The request method is `GET`. The description indicates it checks if a current model is correctly configured. The parameters section shows no parameters. The responses section shows a successful `200` response with a JSON body containing the status and current model name, and a response header indicating the content type is `application/json`.

Request URL: `https://127.0.0.1:8000/health`

Response Body (JSON):

```
{"status": "ok",  
 "current_model": "churn_model_v1_20251214_14_..."}  
Content-Type: application/json
```

Response Headers:

```
content-length: 28  
content-type: application/json  
date: Sun, 10 Dec 2023 14:56:10 GMT  
server: uvicorn
```

## 2. POST /predict

- Reçoit un JSON avec les features du client.
- Retourne :
  - prediction : 0 ou 1
  - probability : probabilité de churn
  - latency\_ms : temps d'inférence
  - model\_version : nom du modèle utilisé
  - request\_id : si fourni
- Les requêtes sont journalisées dans logs/predictions.log.

The screenshot shows a REST API testing interface with the following sections:

- Edit Value / Schema:** A JSON editor containing the following input data:

```
{ "tenure_months": 4, "low_complaints": 1, "high_complaints": 0, "item_type": "music", "region": "AF", "request_id": "req-001" }
```
- Execute** and **Clear** buttons at the bottom of the editor.
- Responses:** A section titled "curl" showing the command used to make the request:

```
curl -X POST "http://127.0.0.1:8000/predict" -H "Content-Type: application/json" -d {"tenure_months": 4, "low_complaints": 1, "high_complaints": 0, "item_type": "music", "region": "AF", "request_id": "req-001"}
```
- Request (raw):** A button labeled "http://127.0.0.1:8000/predict".
- Server response:** A section showing the response status code and details:
  - Code:** 200
  - Details:** Response body
- Response body:** A JSON object identical to the input data, indicating no changes were made:

```
{ "request_id": "req-001", "model_version": "mlops-music-v1-00000000000000000000", "prediction": 0, "probability": 0.0001, "latency_ms": 0.0001, "features": { "tenure_months": 4, "low_complaints": 1, "high_complaints": 0, "item_type": "music", "region": "AF" }, "tags": [ "mlops" ] }
```
- Download** button next to the response body.

Problems	Output	Debug Console	Terminal	Ports
	INFO: 127.0.0.1:56109 - "GET /docs HTTP/1.1" 200 OK INFO: 127.0.0.1:56109 - "GET /openapi.json HTTP/1.1" 200 OK INFO: 127.0.0.1:56109 - "GET /docs HTTP/1.1" 200 OK INFO: 127.0.0.1:56109 - "GET /openapi.json HTTP/1.1" 200 OK INFO: 127.0.0.1:54447 - "POST /predict HTTP/1.1" 200 OK INFO: 127.0.0.1:56109 - "GET /openapi.json HTTP/1.1" 200 OK INFO: 127.0.0.1:54447 - "POST /predict HTTP/1.1" 200 OK INFO: 127.0.0.1:54447 - "POST /predict HTTP/1.1" 200 OK INFO: 127.0.0.1:59677 - "POST /predict HTTP/1.1" 200 OK		uvicorn	8000

## 5. Monitoring du data drift (monitor\_drift.py)

- Le script analyse les **N dernières requêtes** (par défaut 200).
  - Calcule un score Z pour chaque feature numérique :

$$z = \frac{|mean_{prod} - mean_{train}|}{std_{train}}$$

- Déclenche une alerte si  $z \geq 2.5$ .
  - Peut envoyer un hook vers un outil externe si MONITORING\_TOKEN est défini.

The screenshot shows a terminal window with the following content:

```
PS C:\Users\PC\Desktop\Master SDIA\S3\MLops\mllops-lab-01> python src/monitor_drift.py
--- Drift check sur 2 requêtes récentes ---
- tenure_months: mean_prod=103.000 | mean_train=39.246 | z=4.287
ALERTE: drift probable sur tenure_months (z >= 2.5)
- num_complaints: mean_prod=26.500 | mean_train=1.174 | z=22.775
ALERTE: drift probable sur num_complaints (z >= 2.5)
- avg_session_minutes: mean_prod=256.250 | mean_train=35.124 | z=18.686
ALERTE: drift probable sur avg_session_minutes (z >= 2.5)
Résultat : 3 alerte(s) de drift. Analyse recommandée + retraining possible.
PS C:\Users\PC\Desktop\Master SDIA\S3\MLops\mllops-lab-01>
```

## 6. Gestion des versions et rollback (rollback.py)

## **6.1 Activation d'une nouvelle version**

- Exemple pour entraîner et activer la version v2 :

```
  E predictions.log
  x models
    E churn_model_v1_20251214_154550.joblib
      (venv_mllops) PS C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mllops-lab-01> python -c "from src.
        train import main; main(version='v2', gate_f1=0.61)"
    E churn_model_v1_20251214_154550.joblib
      "precision": 0.668789889171974,
      "recall": 0.65625,
      "f1": 0.6624685678233438,
      "baseline_f1": 0.0
    }
    [OK] Modèle sauvegardé : C:\Users\PC\Desktop\Master SDIA\S3\MLOps\mllops-lab-01\models\churn
    model_v2_20251214_154550.joblib
    [DEPLOY] Modèle activé (current): churn_model_v2_20251214_154550.joblib
```

## 6.2 Rollback automatique

- Active le **modèle précédent** en cas de problème :

### Rollback vers un modèle précis

## 7. Conclusion

Ce lab illustre un **pipeline MLOps** :

1. Préparation des données et calcul des statistiques.
2. Entraînement, évaluation, tuning du seuil et enregistrement des métadonnées.
3. Déploiement via API avec logging des prédictions.
4. Monitoring simple de drift pour détecter les anomalies sur les features.
5. Gestion des versions et rollback pour maintenir la stabilité en production.

**Points clés :**

- L'importance du **gate F1** pour contrôler la qualité du modèle avant déploiement.
- La valeur des **logs de prédiction** pour moniter la production et détecter un drift.
- La **gestion des versions** comme outil de sécurité et pédagogie MLOps.