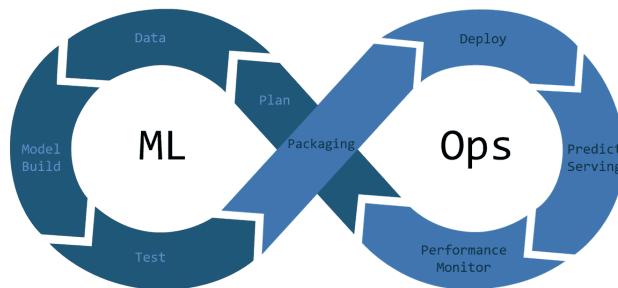


MLOPS : DÉVELOPPEMENT, CI/CD ET MONITORING D'UN MODÈLE DE MACHINE LEARNING

Par : Meriam Hfaidhia



Prédiction du Churn des Clients Télécom



Année universitaire : 2024/2025

Sommaire

Introduction générale.
Chapitre 1 : Architecture du Pipeline MLOps.
Section 1 : Schéma global du pipeline.
Section 2 : Outils et technologies utilisés.
Chapitre 2 : Étapes du Pipeline.
Section 1 : Intégration Continue (CI).
Section 2 : Déploiement Continu (CD).
Conclusion générale.

Introduction générale

Dans un monde où la concurrence entre les entreprises de télécommunications est de plus en plus féroce, la fidélisation des clients est devenue un enjeu stratégique majeur. Le churn, représente la proportion de clients qui abandonnent un service au profit d'un concurrent. La capacité à prédire et anticiper ce phénomène permet aux entreprises d'adopter des stratégies proactives pour améliorer la satisfaction client et optimiser leurs offres.

L'intelligence artificielle, et plus particulièrement le Machine Learning (ML), joue un rôle clé dans cette problématique en permettant d'analyser de vastes quantités de données et d'identifier les profils à risque de désabonnement. Cependant, le développement d'un modèle de machine learning performant ne suffit pas à lui seul ; il est essentiel de garantir son intégration efficace dans un environnement de production. C'est ici qu'intervient MLOps, une approche qui combine le développement, le déploiement et la maintenance continue des modèles de machine learning à grande échelle.

Ce projet vise à concevoir un pipeline MLOps complet pour la prédiction du churn des clients télécoms. L'objectif est d'automatiser les différentes étapes du cycle de vie du modèle, depuis l'entraînement et la validation jusqu'à la mise en production et le monitoring en continu. Nous utiliserons une approche CI/CD (Intégration Continue et Déploiement Continu) pour garantir une mise à jour fluide et sécurisée du modèle.

Dans ce document, nous allons tout d'abord présenter l'architecture du pipeline MLOps, en détaillant les outils et technologies utilisés. Ensuite, nous décrirons les différentes étapes du pipeline, avec un focus sur les pratiques d'Intégration Continue (CI) et de Déploiement Continu (CD). Enfin, nous conclurons en mettant en avant les défis rencontrés et les perspectives d'amélioration de cette approche.

ARCHITECTURE DU PIPELINE MLOPS

Introduction

Dans ce premier chapitre, nous allons explorer l'architecture globale du pipeline MLOps. Nous fournirons un aperçu détaillé des différentes composantes qui composent ce pipeline et des outils utilisés à chaque étape pour garantir l'efficacité, la qualité et la rapidité des processus.

1.1 Schéma global du pipeline

Dans cette section, nous allons détailler les différentes étapes (stages) du pipeline MLOps :

Préparation

L'installation de l'environnement et des dépendances nécessaires à la mise en place du pipeline.

Data Preprocessing

La collecte et la préparation des données nécessaires à l'entraînement du modèle. Cela inclut le nettoyage des données, la gestion des valeurs manquantes, et la sélection des caractéristiques pertinentes ainsi que la sauvegarde des données prétraitées dans des fichiers CSV.

Code Linting

Vérification du code pour s'assurer qu'il respecte les conventions de codage et les bonnes pratiques, avant de procéder aux étapes suivantes.

Run Unit Tests

Lancement des tests unitaires pour valider les fonctionnalités du code et garantir qu'il n'y a pas d'erreurs logiques dans les différentes parties du pipeline.

Train Model

L'entraînement du modèle à l'aide des données prétraitées et des algorithmes choisis, avec ajustement des hyperparamètres.

Fetch Logs from Elasticsearch

Collecte des logs pour assurer la traçabilité et détecter d'éventuels problèmes.

Evaluate Model

Évaluation des performances du modèle sur un jeu de test pour valider sa capacité à généraliser à de nouvelles données.

Save Model

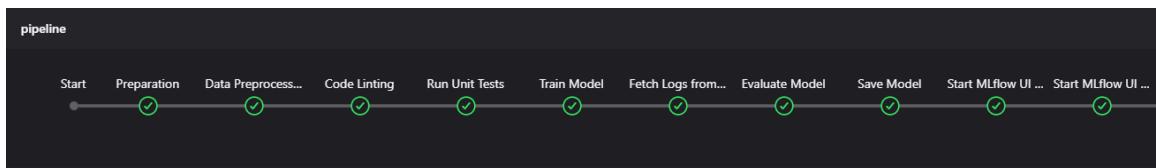
Enregistrement du modèle entraîné pour qu'il puisse être utilisé pour les prédictions futures ou déployé en production.

Start MLflow UI Command

Lancement de l'interface utilisateur de MLflow pour suivre et gérer les expériences de machine learning et leurs résultats.

Start MLflow UI Docker Container

Déploiement de l'interface MLflow dans un container Docker pour garantir la portabilité et la gestion des versions.



Promote Model

Le modèle validé est promu pour un usage en production.

Start FastAPI Application

Lancement de l'API FastAPI pour permettre aux utilisateurs d'interagir avec le modèle en envoyant des requêtes et en recevant des prédictions.

Start FastAPI Application

Lancement de l'API FastAPI pour permettre aux utilisateurs d'interagir avec le modèle en envoyant des requêtes et en recevant des prédictions.

Deploy API commande

Déploiement du modèle .

Deploy Flask API

Lancement de l'API Flask, alternative à FastAPI.

Test Flask API Deployment

Test des endpoints de l'API Flask pour vérifier qu'ils fonctionnent comme prévu **Test API**

Test des endpoints de l'API Fast pour vérifier qu'ils fonctionnent comme prévu.

Test Prediction

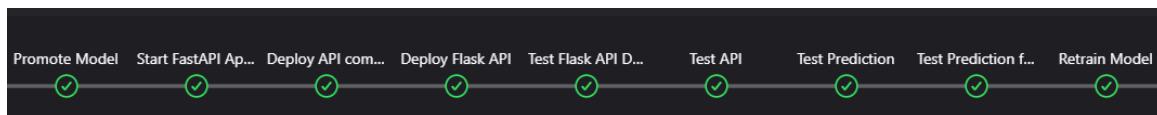
Test des prédictions effectuées par l'API Fast pour valider les résultats et leur pertinence.

Test Prediction for FlaskApp

Test des prédictions effectuées par l'API Flask pour valider les résultats et leur pertinence.

Retrain Model

Réentraînement du modèle avec de nouvelles données.



Generate Model Card

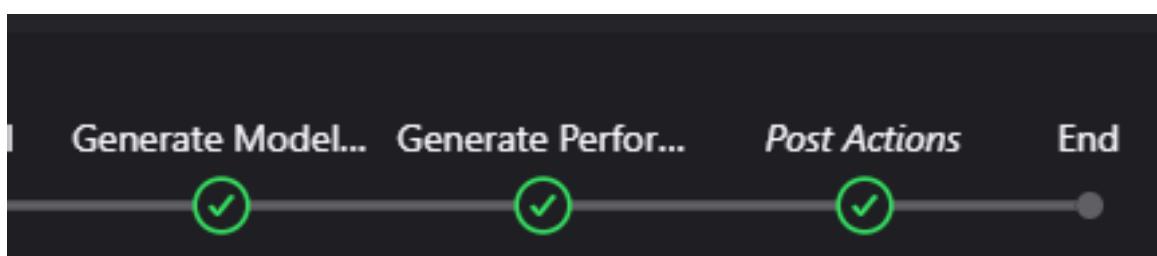
Création d'un document décrivant le modèle, son objectif, ses performances et les conditions d'utilisation.

Generate Performance Report

Création d'un rapport détaillant les performances du modèle, y compris des métriques clés comme la précision, le rappel.

Post Actions

Les actions à réaliser après le déploiement du modèle.



1.2 Outils et technologies utilisés

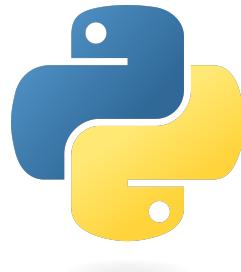
Dans cette section, nous détaillerons les outils et technologies qui composent le pipeline MLOps.

Chaque outil est utilisé pour une étape spécifique du pipeline, afin de garantir l'efficacité, la reproductibilité, et l'automatisation des processus.

Voici les principaux outils et technologies utilisés :

Environnement de développement

- **Python** : Le langage de programmation principal utilisé pour développer et automatiser les différentes étapes du pipeline. Il est utilisé pour l'entraînement du modèle, les tests unitaires et l'intégration des outils.



- **Jupyter Notebooks** : Utilisé pour l'exploration des données, les analyses expérimentales et la visualisation. Permet de documenter le processus d'analyse et d'expérimentation.

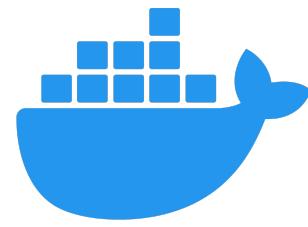


Gestion des versions et des dépendances

- **Git** : Outil de gestion des versions utilisé pour suivre les modifications du code source, collaborer avec d'autres développeurs et garantir la gestion du code dans un environnement contrôlé.



- **Docker** : Permet de containeriser les différentes parties du pipeline pour garantir la portabilité et l'isolation des processus.



docker[®]

Monitoring et gestion des logs

- **Elasticsearch** : Outil utilisé pour la collecte et l'analyse des logs générés par le pipeline MLOps.



- **Kibana** : Interface de visualisation pour explorer et analyser les logs collectés par Elasticsearch.



Déploiement et API

- **FastAPI** : Framework Python utilisé pour développer des API modernes, performantes et simples à maintenir pour le déploiement du modèle en production.



- **Flask** : Framework léger pour la création d'API RESTful



Suivi des performances et des résultats

- **MLflow UI** : Interface de gestion des expériences de machine learning, permettant de suivre les métriques et les résultats des différents modèles et itérations.



Outils de tests et validation

- **PyTest** : Framework de tests pour Python, utilisé pour l'automatisation des tests unitaires et

des tests d'intégration dans le pipeline.



- ****Flake8**** : Outil de linting pour Python, utilisé pour vérifier que le code respecte les conventions de style et de bonnes pratiques. Cela aide à maintenir un code propre et cohérent.



Outils de gestion des bases de données

- ****MongoDB**** : Base de données NoSQL utilisée pour stocker des données non structurées ou semi-structurées. Elle est particulièrement adaptée pour le stockage des logs et des données d'entraînement des modèles



- ****PostgreSQL**** : Système de gestion de base de données relationnelle (SGBDR) utilisé pour la gestion des données structurées, utiliser pour enregistrer les prédictions



Outils d'intégration continue (CI) et d'automatisation

- **Jenkins** : Outil d'intégration continue (CI) qui automatise la gestion du pipeline de développement, permettant d'exécuter des tests, de déployer des modèles et d'assurer la livraison continue (CD).



Conclusion

Ce pipeline MLOps assure un déploiement structuré et automatisé des modèles de machine learning. Grâce à des outils comme MLflow, Docker et FastAPI, il garantit la traçabilité, la scalabilité et la gestion efficace des modèles en production. L'intégration de bases de données et de solutions CI/CD optimise le suivi et la fiabilité du système, facilitant ainsi l'industrialisation des projets ML.

ÉTAPES DU PIPELINE.

Introduction

Ce chapitre décrit les étapes clés du pipeline MLOps. L'intégration continue (CI) automatise les tests et la validation du code, tandis que le déploiement continu (CD) assure une mise en production fluide et sécurisée des modèles.

2.1 Intégration Continue (CI)

2.1.1 Installer les dépendances

```
pipeline {
    agent any

    environment {
        PROJECT_DIR = '/home/meriam/meriam-hfaidhia-4DS4-mlops_project'
        MODEL_PATH = "${PROJECT_DIR}/prediction_model.joblib"
        REPORT_PATH="${PROJECT_DIR}/performance_report.md"
        MLFLOW_URL = 'http://127.0.0.1:5000'
    }
}
```

Le bloc `environment` permet de définir des variables d'environnement que tu peux utiliser dans l'ensemble du pipeline.

- `PROJECT_DIR` : Définit le répertoire principal du projet.
- `MODEL_PATH` : Utilise `PROJECT_DIR` pour construire le chemin d'accès au fichier du modèle entraîné. Cela permet de stocker et d'accéder facilement au modèle pour les prédictions. Le fichier du modèle est un fichier `.joblib` qui est un format utilisé pour enregistrer les modèles Python.
- `REPORT_PATH` : Définit le chemin pour le rapport de performance. Il s'agit d'un fichier `.md` où tu pourras probablement stocker des métriques ou des résultats liés au modèle.
- `MLFLOW_URL` : Définit l'URL de l'interface MLflow. MLflow est une plateforme pour gérer les expérimentations de machine learning, et l'URL est configurée ici pour se connecter à un serveur local de MLflow (`http://127.0.0.1:5000`).

```

stages {
    stage('Preparation') {
        steps {
            script {
                echo "⚙ Installation des dépendances..."
                sh """
                set -e
                cd ${env.PROJECT_DIR}
                python3 -m venv venv
                . venv/bin/activate
                pip install -r requirements.txt
                sudo apt-get update
                sudo apt-get install -y jq
                sudo chown -R jenkins:jenkins ${env.PROJECT_DIR}
                """
            }
        }
    }
}

```

```

requierements.txt
1 numpy
2 pandas
3 scikit-learn
4 matplotlib
5 jupyter
6 mlflow
7 fastapi
8 uvicorn
9 joblib
10 pytest
11 flake8
12 pymongo
13 tabulate
14 seaborn
15 sqlalchemy
16 psycopg2-binary
17 pydantic
18 elasticsearch
19 |

```

Stage : Préparation

Ce stage est chargé de configurer l'environnement du projet avant l'exécution. Il inclut les étapes suivantes :

- **Affichage d'un message d'état** pour indiquer l'installation des dépendances.
- **Navigation vers le répertoire du projet** (`PROJECT_DIR`).
- **Création et activation d'un environnement virtuel Python** pour gérer les dépendances.
- **Installation des dépendances requises** depuis le fichier `requirements.txt`.
- **Mise à jour des paquets système** avec `apt-get update`.
- **Installation d'outils supplémentaires** comme `jq` (pour traiter du JSON).

- **Attribution des permissions nécessaires** en définissant Jenkins comme propriétaire du répertoire du projet.

Ce stage garantit que toutes les configurations et dépendances essentielles sont en place avant de passer aux étapes suivantes du pipeline.

2.1.2 Préparation des données

```
stage('Data Preprocessing') {
    steps {
        script {
            echo "📊 Préparation des données..."
            sh """
                cd ${env.PROJECT_DIR}
                . venv/bin/activate
                python main.py --prepare --train_path churn-bigml-80.csv --test_path churn-bigml-20.csv
            """
        }
    }
}
```

Fonction Prepare Data dans Model Pipeline

```
def prepare_data(train_path, test_path):
    """
    Charger et préparer les données d'entraînement et de test.
    """

    # Charger les données
    train_data = pd.read_csv(train_path)
    test_data = pd.read_csv(test_path)

    # Fusionner les données pour prétraitement
    data = pd.concat([train_data, test_data], ignore_index=True)

    # Supprimer les colonnes inutiles
    columns_to_drop = [
        'State', 'Area code', 'Total day minutes', 'Total eve minutes',
        'Total night minutes', 'Total intl minutes'
    ]
    data = drop_columns(data, columns_to_drop)

    # Encodage des variables catégorielles
    label_encoder = LabelEncoder()
    data['International plan'] = label_encoder.fit_transform(
        data['International plan']
    )
    data['Voice mail plan'] = label_encoder.fit_transform(
        data['Voice mail plan']
    )
    data['Churn'] = label_encoder.fit_transform(data['Churn'])

    # Normalisation des données numériques
    numerical_columns = [
        'Account length', 'Number vmail messages', 'Total day calls',
        'Total day charge', 'Total eve calls', 'Total eve charge',
        'Total night calls', 'Total night charge', 'Total intl calls',
        'Total intl charge', 'Customer service calls'
    ]
```

Fonction Prepare Data dans Main

```
if args.prepare or X_train is None:
    logging.info(
        "⌚ Chargement et préparation des données..."
    )
    X_train, X_test, y_train, y_test = prepare_data(
        args.train_path, args.test_path
    )
    logging.info(
        "✅ Données préparées et sauvegardées."
    )
if args.prepare:
    return
```

```
if args.prepare or X_train is None:
    logging.info(
        "⌚ Chargement et préparation des données..."
    )
    X_train, X_test, y_train, y_test = prepare_data(
        args.train_path, args.test_path
    )
    logging.info(
        "✅ Données préparées et sauvegardées."
    )
if args.prepare:
    return
```

Stage : Data Preprocessing

Ce stage consiste à préparer les données avant l'entraînement du modèle. Elle exécute les actions suivantes :

- **Accès au projet** : Le script se place dans le répertoire du projet via la commande `cd $env.PROJECT_DIR`.
- **Activation de l'environnement virtuel** : Il active l'environnement Python avec `. venv/bin/activate` pour s'assurer que toutes les dépendances nécessaires sont disponibles.
- **Exécution du prétraitement des données** : Le script lance `main.py` avec l'option `-prepare`, en spécifiant les fichiers d'entraînement et de test :
 - **churn-bigml-80.csv** : Jeu de données d'entraînement.
 - **churn-bigml-20.csv** : Jeu de données de test.

Ce processus assure que les données sont nettoyées et prêtes pour l'entraînement du modèle.

Résultat

```
2025-03-05 09:24:25,991 - INFO - ⌚ Chargement et préparation des données...
2025-03-05 09:24:26,112 - INFO - ✅ Données préparées et sauvegardées.
```

2.1.3 Test des données

```
stage('Code Linting') {
    steps {
        script {
            echo "🔍 Vérification du style de code avec Flake8..."
            sh """
                set -e
                cd ${env.PROJECT_DIR}
                . venv/bin/activate
                flake8 main.py model_pipeline.py --count --show-source --statistics
            """
        }
    }
}
```

Stage : Code Linting

Ce stage est dédié à la vérification du style de code dans le projet en utilisant l'outil [Flake8](#), qui est utilisé pour analyser la qualité du code Python et détecter les erreurs de style ou les mauvaises pratiques.

- **Objectif** : Assurer que le code respecte les bonnes pratiques de programmation Python et éviter les erreurs communes.
- **Étapes exécutées** :
 - `cd $env.PROJECT_DIR` : Le répertoire du projet est défini par la variable d'environnement `PROJECT_DIR`, et nous nous y rendons pour exécuter la commande.
 - `. venv/bin/activate` : L'environnement virtuel Python est activé pour utiliser les dépendances installées spécifiquement pour ce projet.
 - `flake8` : L'outil `flake8` est exécuté pour analyser les fichiers `main.py` et `model_pipeline.py` du projet. Les options suivantes sont utilisées :
 - `-count` : Affiche le nombre total d'erreurs détectées.
 - `-show-source` : Affiche la source des erreurs directement dans le code.
 - `-statistics` : Donne un résumé des erreurs sous forme de statistiques.
- **Importance** : L'analyse du code avec Flake8 permet de maintenir un code propre et de prévenir les erreurs difficiles à détecter pendant l'exécution.

Cette étape garantit la qualité et la lisibilité du code Python, ce qui facilite sa maintenance et améliore la collaboration dans l'équipe de développement.

Résultat

```
+ hash -r
+ flake8 main.py model_pipeline.py --count --show-source --statistics
0
```

```
stage('Run Unit Tests') {
    steps {
        script {
            echo "💡 Exécution des tests unitaires..."
            sh """
                set -e
                cd ${env.PROJECT_DIR}
                . venv/bin/activate
                pytest tests/ --maxfail=1 --disable-warnings -q
            """
        }
    }
}
```

Stage : Run Unit Tests

Ce stage est dédié à l'exécution des tests unitaires pour vérifier le bon fonctionnement des différentes parties du code et s'assurer que les modifications récentes n'ont pas introduit d'erreurs.

- **Objectif** : Exécuter les tests unitaires pour vérifier la stabilité du code et garantir que les fonctionnalités sont conformes aux attentes.
- **Étapes exécutées** :
 - `cd $env.PROJECT_DIR` : Le répertoire du projet est défini par la variable d'environnement `PROJECT_DIR`, et nous nous y rendons pour exécuter les tests.
 - `. venv/bin/activate` : L'environnement virtuel Python est activé pour utiliser les dépendances nécessaires à l'exécution des tests.
 - `pytest` : L'outil `pytest` est exécuté pour lancer les tests unitaires situés dans le répertoire `tests/`. Les options suivantes sont utilisées :
 - `-maxfail=1` : Arrête l'exécution des tests après un premier échec.
 - `-disable-warnings` : Désactive l'affichage des avertissements pour une sortie plus propre.
 - `-q` : Exécute les tests en mode silencieux pour afficher uniquement les résultats importants.
- **Importance** : Exécuter les tests unitaires permet de détecter rapidement toute régression dans le code et de garantir que les nouvelles modifications ne cassent pas des fonctionnalités existantes.

```
tests > ⚡ test_model_pipeline.py
1  import sys
2  import os
3  import pytest
4
5  # Add the parent directory to the Python path so we can import model_pipeline
6  sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
7
8  from model_pipeline import prepare_data # Correct import for model_pipeline
9
10 def test_prepare_data():
11     # Correct file paths for the CSV files in the root directory
12     train_path = os.path.join(os.path.dirname(__file__), '..', 'churn-bigml-80.csv')
13     test_path = os.path.join(os.path.dirname(__file__), '..', 'churn-bigml-20.csv')
14
15     # Check if the files exist
16     assert os.path.exists(train_path), f"Train data file not found at {train_path}"
17     assert os.path.exists(test_path), f"Test data file not found at {test_path}"
18
19     # Call the prepare_data function
20     X_train, X_test, y_train, y_test = prepare_data(train_path, test_path)
21
22     # Check if the processed data files are created
23     assert os.path.exists('train_data_prepared.csv'), "Train data file was not created"
24     assert os.path.exists('test_data_prepared.csv'), "Test data file was not created"
25
26     # Further assertions to check the shape and types of the returned data
27     assert X_train.shape[0] == len(y_train), "Mismatch between X_train and y_train size"
28     assert X_test.shape[0] == len(y_test), "Mismatch between X_test and y_test size"
29     assert X_train.shape[1] == 13, "Incorrect number of features in X_train"
30     assert X_test.shape[1] == 13, "Incorrect number of features in X_test"
31
```

Cette étape assure que les tests de l'application sont passés avec succès, ce qui est crucial avant de passer à l'étape de déploiement.

Résultat

```
+ pytest tests/ --maxfail=1 --disable-warnings -q
.
[100%]
1 passed in 1.85s
```

2.1.4 Entrainement des données

```
stage('Train Model') {
    steps {
        script {
            echo "🚀 Entraînement du modèle..."

            // Vérification si MongoDB est démarré et démarrer si nécessaire
            sh """
                set -e
                cd ${env.PROJECT_DIR}
                . venv/bin/activate

                # Vérification si MongoDB est en cours d'exécution
                if ! docker ps -q -f name=mongodb_official; then
                    echo "MongoDB n'est pas en cours d'exécution. Tentative de démarrage..."
                    if ! docker ps -a -q -f name=mongodb_official; then
                        # MongoDB n'existe pas, création du conteneur
                        echo "Création du conteneur MongoDB..."
                        docker run --name mongodb_official -d -p 27017:27017 mongo || echo "Échec de la création du conteneur MongoDB"
                    else
                        # MongoDB est arrêté, démarrage du conteneur
                        echo "Démarrage du conteneur MongoDB..."
                        docker start mongodb_official || echo "Échec du démarrage de MongoDB"
                    fi
                else
                    echo "✅ MongoDB est déjà en cours d'exécution."
                fi
            """
        }
    }
}
```

```
// Attendre que MongoDB soit prêt
sh """
set -e
cd ${env.PROJECT_DIR}
. venv/bin/activate

# Attendre que MongoDB soit prêt
echo "⏳ Attente que MongoDB soit prêt..."
until docker exec mongodb_official mongosh --eval "db.adminCommand('ping')" > /dev/null 2>&1; do
    sleep 1
done
echo "✅ MongoDB est prêt."
"""

// Activation de l'environnement virtuel et exécution du script Python
sh """
set -e
cd ${env.PROJECT_DIR}
. venv/bin/activate
python3 main.py --train --train_path "train_data_prepared.csv" --test_path "test_data_prepared.csv"
"""
}
```

Stage : Train Model

L'objectif de ce stage est d'entraîner le modèle de machine learning sur les données préparées.

Avant de commencer l'entraînement, le stage vérifie si MongoDB est en cours d'exécution et démarre ou crée un conteneur Docker si nécessaire.

— **Objectif** : Entraîner le modèle de machine learning sur les données de formation et de test, après avoir validé la disponibilité de MongoDB.

— **Étapes exécutées** :

— `cd ${env.PROJECT_DIR}` : Le répertoire du projet est défini et navigué.

- `. venv/bin/activate` : Activation de l'environnement virtuel pour utiliser les dépendances nécessaires.
- **Vérification et démarrage de MongoDB :**
 - `docker ps -q -f name=mongodb_official` : Vérifie si MongoDB est déjà en cours d'exécution.
 - Si MongoDB n'est pas en cours d'exécution :
 - `docker run -name mongodb_official -d -p 27017:27017 mongo` : Crée un conteneur Docker MongoDB.
 - `docker start mongodb_official` : Démarré MongoDB si le conteneur existe mais est arrêté.
- **Attente de la disponibilité de MongoDB :**
 - `docker exec mongodb_official mongosh -eval "db.adminCommand('ping')"` : Vérifie la disponibilité de MongoDB.
 - Attente jusqu'à ce que MongoDB soit prêt, puis message de confirmation.
- `python3 main.py -train -train_path -test_path` : Exécute le script Python pour entraîner le modèle avec les données préparées.
- **Importance** : Cette étape est cruciale pour l'entraînement du modèle, et la gestion de MongoDB est importante pour stocker ou récupérer des informations liées au processus d'entraînement.

```

if args.train:
    logging.info(
        "🚀 Entraînement du modèle..."
    )
    model, params = train_model(X_train, y_train)

    if model:
        logging.info(
            "✅ Modèle entraîné avec succès. Paramètres :"
        )
        for param, value in params.items():
            logging.info(f"{param}: {value}")
            mlflow.log_param(param, value)

    signature = infer_signature(X_train, y_train)

    mlflow.sklearn.log_model(
        model,
        "churn_model",
        signature=signature)

    model_version = register_model(model, X_train=X_train)
    if model_version:
        logging.info(
            f"Le modèle a été enregistré avec "
            f"la version {model_version.version}."
        )

    accuracy, precision, recall, f1 = (
        evaluate_model(model, X_test, y_test)
    )

    result_message = (
        f"✅ Résultats de l'entraînement :\n"
        f"- Accuracy: {accuracy:.4f}\n"
        f"- Precision: {precision:.4f}\n"
        f"- Recall: {recall:.4f}\n"
    )

```

Résultat

```

+ python3 main.py --train --train_path train_data_prepared.csv --test_path test_data_prepared.csv
2025-03-05 09:24:39,431 - INFO - ✅ MongoDB est déjà en cours d'exécution dans Docker.
2025-03-05 09:24:39,440 - INFO - ✅ Connexion à MongoDB réussie.
2025-03-05 09:24:39,441 - INFO - ✅ MLflow tracking URI set to: sqlite:///mlflow.db
2025-03-05 09:24:40,072 - INFO - Context impl SQLiteImpl.
2025-03-05 09:24:40,072 - INFO - Will assume non-transactional DDL.
2025-03-05 09:24:40,099 - INFO - ✅ Expérience 'churn' existe déjà. ID: 3
2025-03-05 09:24:40,102 - INFO - ✅ Expérience actuelle définie sur 'churn'.
2025-03-05 09:24:40,105 - INFO - 📁 Chargement des données préparées depuis les fichiers existants...
2025-03-05 09:24:40,795 - INFO - 🚀 Entraînement du modèle...
2025-03-05 09:24:41,415 - INFO - ✅ Modèle entraîné avec succès. Paramètres :
2025-03-05 09:24:41,415 - INFO - bootstrap: True
2025-03-05 09:24:41,442 - INFO - ccp_alpha: 0.0
2025-03-05 09:24:41,467 - INFO - class_weight: None
2025-03-05 09:24:41,489 - INFO - criterion: gini
2025-03-05 09:24:41,510 - INFO - max_depth: None
2025-03-05 09:24:41,531 - INFO - max_features: sqrt
2025-03-05 09:24:41,554 - INFO - max_leaf_nodes: None
2025-03-05 09:24:41,578 - INFO - max_samples: None
2025-03-05 09:24:41,598 - INFO - min_impurity_decrease: 0.0
2025-03-05 09:24:41,624 - INFO - min_samples_leaf: 1
2025-03-05 09:24:41,646 - INFO - min_samples_split: 2
2025-03-05 09:24:41,669 - INFO - min_weight_fraction_leaf: 0.0

```

```
2025-03-05 09:24:47,502 - INFO - ✅ Résultats de l'entraînement :  
- Accuracy: 0.9783  
- Precision: 1.0000  
- Recall: 0.7500  
- F1-score: 0.8571
```

2.1.5 Envoi des logs au ElasticSearch

```
stage('Fetch Logs from Elasticsearch') {  
    steps {  
        script {  
            echo "🌐 Récupération des logs depuis Elasticsearch..."  
  
            // Effectuer une requête GET vers Elasticsearch pour récupérer les logs  
            def response = sh(script: """  
                curl -X GET "http://localhost:9200/mlflow-metrics/_search?pretty" -H 'Content-Type: application/json'  
                """", returnStdout: true).trim()  
  
            // Afficher la réponse dans la console de Jenkins  
            echo "Réponse Elasticsearch: ${response}"  
        }  
    }  
}
```

```

def connect_to_elasticsearch():
    """
    Connexion à Elasticsearch.
    """

    try:
        es = Elasticsearch(
            "http://localhost:9200",
            verify_certs=False
        )
        if es.info():
            logging.info("✓ Connexion à Elasticsearch réussie.")
            return es
        else:
            logging.error("✗ Impossible de se connecter à Elasticsearch.")
            return None
    except Exception as e:
        logging.error(f"✗ Erreur lors de la connexion à Elasticsearch : {e}")
        return None


def send_logs_to_elasticsearch(es, run_id, params, metrics):
    """
    Envoie les logs de MLflow (paramètres et métriques) à Elasticsearch.
    """

    if es is None:
        logging.warning("✗ Elasticsearch n'est pas connecté.")
        return

    log_data = {
        "run_id": run_id,
        "params": params,
        "metrics": metrics,
        "timestamp": pd.Timestamp.now().isoformat()
    }

    try:
        es.index(index="mlflow-metrics", body=log_data)
        logging.info("✓ Logs envoyés à Elasticsearch avec succès.")
    except:

```

L'objectif de ce stage est de récupérer les logs de performance ou d'exécution stockés dans Elasticsearch à l'aide d'une requête GET. Ces logs peuvent être utilisés pour surveiller le modèle ou analyser les métriques des expérimentations de machine learning.

- **Objectif** : Récupérer les logs depuis Elasticsearch et les afficher dans la console Jenkins.
- **Étapes exécutées :**
 - **curl -X GET "http://localhost:9200/mlflow-metrics** : Effectuer une requête GET vers Elasticsearch pour récupérer les logs envoyées et liés à `mlflow-metrics`.
 - **Réponse affichée** : Le contenu de la réponse JSON obtenue par la requête est ensuite affiché dans la console Jenkins.

- **echo "Réponse Elasticsearch : \$response"** : Afficher la réponse dans la console pour qu'elle soit accessible et analysée par l'utilisateur.
- **Importance** : Cette étape permet de visualiser les métriques ou logs stockés dans Elasticsearch pour effectuer un suivi ou une analyse post-exécution du modèle de machine learning.

Résultat

```
⌚ Récupération des logs depuis Elasticsearch...
[Pipeline] sh
+ curl -X GET http://localhost:9200/mlflow-metrics/_search?pretty -H Content-Type: application/json
% Total    % Received % Xferd  Average Speed   Time     Time      Time  Current
          Dload  Upload   Total Spent    Left Speed
0       0     0     0     0       0       0       0 ---:--- ---:--- ---:---     0
100  6006     0  6006     0       0  508k       0 ---:--- ---:--- ---:---  533k
[Pipeline] echo
Réponse Elasticsearch: {
  "took" : 3,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    }
  }
}
```

```

    ],
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "mlflow-metrics",
        "_id" : "Lcd6YZUBM-oRn_M6uKh4",
        "_score" : 1.0,
        "_source" : {
          "run_id" : "1c90217d1cd84ff3b0514ede2a12af90",
          "params" : {
            "bootstrap" : true,
            "ccp_alpha" : 0.0,
            "class_weight" : null,
            "criterion" : "gini",
            "max_depth" : null,
            "max_features" : "sqrt",
            "max_leaf_nodes" : null,
            "max_samples" : null,
            "min_impurity_decrease" : 0.0,
            "min_samples_leaf" : 1,
            "min_samples_split" : 2,
            "min_weight_fraction_leaf" : 0.0,
            "monotonic_cst" : null,
            "n_estimators" : 100,

```

2.1.6 Evaluation du modèle

```

stage('Evaluate Model') {
  steps {
    script {
      echo "💡 Évaluation du modèle..."
      sh """
        set -e
        cd ${env.PROJECT_DIR}
        . venv/bin/activate
        python main.py --evaluate --train_path train_data_prepared.csv --test_path test_data_prepared.csv --l
      """
    }
  }
}

```

Stage : Evaluate

L'objectif de ce stage est d'évaluer les performances du modèle en utilisant un jeu de données d'entraînement et de test préalablement préparé. L'évaluation est effectuée à l'aide d'un modèle préexistant sauvegardé dans un fichier `joblib`.

- **Objectif** : Évaluer les performances du modèle en utilisant les données préparées.

- **Importance** : Cette étape permet de mesurer les performances du modèle, notamment son taux de précision, ses erreurs, et d'autres métriques importantes pour évaluer son efficacité sur les données de test.

Résultat

```
+ python main.py --evaluate --train_path train_data_prepared.csv --test_path test_data_prepared.csv --load prediction_model.joblib
2025-03-05 09:24:54,476 - INFO - 📁 Chargement des données préparées depuis les fichiers existants...
2025-03-05 09:24:54,501 - INFO - 🚀 Chargement du modèle depuis prediction_model.joblib...
2025-03-05 09:24:54,537 - INFO - 🎯 Évaluation du modèle...
2025-03-05 09:24:54,556 - INFO - ✅ Résultats de l'évaluation :
- Accuracy: 0.9783
- Precision: 1.0000
- Recall: 0.7500
- F1-score: 0.8571
```

2.1.7 Enregistrement du modèle

```
stage('Save Model') {
    steps {
        script {
            echo "💾 Sauvegarde du modèle..."
            sh """
                set -e
                cd ${env.PROJECT_DIR}
                . venv/bin/activate
                python main.py --load prediction_model.joblib --save churn_model_backup.joblib --train_path train_data_prepared.csv --test_path test_data_prepared.csv
            """
        }
    }
}
```

Stage : Save

L'objectif de ce stage est de sauvegarder le modèle de machine learning entraîné afin de pouvoir le réutiliser ultérieurement. Cette étape est essentielle pour garantir la persistance du modèle après son évaluation et permettre des déploiements futurs.

- **Objectif** : Sauvegarder le modèle évalué pour une utilisation future.
- **Étapes exécutées** :
 - `python main.py --load prediction_model.joblib --save churn_model_backup.joblib --train_path train_data_prepared.csv --test_path test_data_prepared.csv` :

Chargement du modèle pré-entraîné `prediction_model.joblib`, puis sauvegarde sous le nom `churn_model_backup.joblib` en utilisant les jeux de données `train_data_prepared.csv` et `test_data_prepared.csv`.
- **Importance** : Cette étape garantit que le modèle validé peut être conservé pour de futures prédictions sans nécessiter un nouvel entraînement. Cela permet aussi de comparer différentes

valeurs du modèle et de suivre son amélioration dans le temps.

Résultat

```
+ python main.py --load prediction_model.joblib --save churn_model_backup.joblib --train_path train_data_prepared.csv --test_path test_data_prepared.csv
2025-03-05 09:24:59,369 - INFO - ⚡ Chargement des données préparées depuis les fichiers existants...
2025-03-05 09:24:59,396 - INFO - 🚀 Chargement du modèle depuis prediction_model.joblib...
2025-03-05 09:24:59,437 - INFO - 📁 Sauvegarde du modèle dans churn_model_backup.joblib...
```

2.1.8 Mlflow Commande

```
stage('Start MLflow UI commande') {
    steps {
        script {
            echo "🚀 Démarrage du serveur MLflow..."
            sh """
                set -e
                cd ${env.PROJECT_DIR}
                . venv/bin/activate
                nohup mlflow ui --backend-store-uri sqlite:///mlflow.db --default-artifact-root mongodb://localhost:27017/mlflow_artifacts > mlflow.log 2>&1 &
            """
            echo "✅ MLflow UI a démarré avec succès !"
            echo "Accédez à l'interface MLflow à l'adresse : ${env.MLFLOW_URL}"
            sleep 10
        }
    }
}
```

Stage : Start MLflow UI commande

L’objectif de ce stage est de lancer l’interface utilisateur de MLflow afin de suivre les expériences et les métriques du modèle de machine learning. MLflow UI permet de visualiser les runs, les paramètres d’entraînement et les artefacts stockés.

- **Objectif** : Démarrer l’interface utilisateur de MLflow pour le suivi des expérimentations.
- **Étapes exécutées** :
- **nohup mlflow ui --backend-store-uri sqlite ://mlflow.db --default-artifact-root mongodb ://localhost :27017/mlflow_artifacts > mlflow.log 2>&1 &** : Lancement du serveur MLflow en arrière-plan avec une base de données SQLite pour stocker les métriques et MongoDB pour les artefacts.
- **Résultat** : Une fois le serveur démarré, l’interface MLflow est accessible via l’URL \${env.MLFLOW_URL}.
- **Importance** : Cette étape est essentielle pour suivre les performances du modèle, comparer les runs et analyser l’impact des hyperparamètres sur les résultats.

Résultat

```
+ VIRTUAL_ENV_PROMPT=(venv)
+ export VIRTUAL_ENV_PROMPT
+ hash -r
+ nohup mlflow ui --backend-store-uri sqlite:///mlflow.db --default-artifact-root
mongodb://localhost:27017/mlflow_artifacts
[Pipeline] echo
✓ MLflow UI a démarré avec succès !
[Pipeline] echo
Accédez à l'interface MLflow à l'adresse : http://127.0.0.1:5000
[Pipeline]
```

The screenshot shows the MLflow UI interface. At the top, there's a navigation bar with 'mlflow 2.20.2' and tabs for 'Experiments' and 'Models'. Below this, the 'Experiments' section is active, showing a list of experiments. One experiment, 'churn_prediction', is selected and shown in more detail. The 'Runs' tab is selected, displaying a single run named 'overjoyed-lynx-999' which was created 45 seconds ago.

This screenshot provides a detailed view of the 'overjoyed-lynx-999' run within the 'churn_prediction' experiment. The 'Overview' tab is selected. Key details include:

- Description:** No description
- Details:**
 - Created at: 2025-02-24 00:14:14
 - Created by: meriam
 - Experiment ID: 640981784163603198
 - Status: Finished
 - Run ID: bedf48c00ca942319add57be13cf339
 - Duration: 5.0s
 - Datasets used: —
 - Tags: Add
 - Source: main.py (linked to cb0add0)
 - Logged models: sklearn
 - Registered models: —

2.1.9 Mlflow Conteneur

```

stage('Start MLflow UI docker conteneur') {
    steps {
        script {
            echo "💡 Démarrage du serveur MLflow..."
            // Stop and remove existing container if it exists
            sh """
                set -e
                cd ${env.PROJECT_DIR}
                docker ps -aq --filter name=mlflow_server | xargs -r docker stop
                docker ps -aq --filter name=mlflow_server | xargs -r docker rm
                docker run -d -p 5011:5000 --name mlflow_server \
                -v "${WORKSPACE}/mlflow.db:/mlflow.db" \
                -e MLFLOW_BACKEND_STORE_URI.sqlite:///mlflow.db \
                -e MLFLOW_DEFAULT_ARTIFACT_ROOT=mongodb://mongodb_official:27017/mlflow_artifacts \
                meriam-hfaidhia-4ds4-mlops_project-mlflow_server
            """
            echo "✅ MLflow UI a démarré avec succès !"
            echo "Accédez à l'interface MLflow à l'adresse : ${env.MLFLOW_URL}"
            sleep 10
        }
    }
}

```

Stage : Start MLflow UI docker conteneur

L'objectif de ce stage est de lancer l'interface utilisateur de MLflow dans un conteneur Docker afin de faciliter le suivi des expérimentations et des métriques du modèle de machine learning. L'utilisation de Docker garantit un environnement reproductible et indépendant.

- **Objectif** : Démarrer MLflow UI dans un conteneur Docker pour gérer et visualiser les expérimentations de machine learning.
- **Étapes exécutées** :
 - **docker ps -aq --filter name=mlflow_server | xargs -r docker stop** : Arrêt du conteneur MLflow s'il est déjà en cours d'exécution.
 - **docker ps -aq --filter name=mlflow_server | xargs -r docker rm** : Suppression de l'ancien conteneur MLflow pour éviter tout conflit.
 - **docker run -d -p 5011 :5000 --name mlflow_server -v "\${WORKSPACE}/mlflow.db:/mlflow.db" -e MLFLOW_BACKEND_STORE_URI=sqlite:///mlflow.db -e MLFLOW_DEFAULT_ARTIFACT_ROOT=mongodb://mongodb_official:27017/mlflow_artifacts meriam-hfaidhia-4ds4-mlops_project-mlflow_server** : Lancement d'un conteneur Docker exécutant MLflow avec :
 - Un volume monté pour stocker la base de données SQLite de MLflow.
 - La configuration du stockage des artefacts sur MongoDB.
 - L'exposition du port 5011 pour accéder à l'interface utilisateur.
- **Résultat** : Une fois le serveur MLflow démarré, son interface est accessible à l'URL `${env.MLFLOW_URL}`.
- **Importance** : Cette étape est essentielle pour centraliser le suivi des expériences, enregistrer les métriques et visualiser les artefacts du modèle dans un environnement contrôlé et reproductible.

Résultat

```
🚀 Démarrage du serveur MLflow...
[Pipeline] sh
+ set -e
+ cd /home/meriam/meriam-hfaidhia-4DS4-mlops_project
+ docker ps -aq --filter name=mlflow_server
+ xargs -r docker stop
d50a99fe28cc
+ docker ps -aq --filter name=mlflow_server
+ xargs -r docker rm
d50a99fe28cc
+ docker run -d -p 5011:5000 --name mlflow_server -v /var/lib/jenkins/workspace/ML Churn Prediction/mlflow.db:/mlflow.db
-e MLFLOW_BACKEND_STORE=sqlite:///mlflow.db -e
MLFLOW_DEFAULT_ARTIFACT_ROOT=mongodb://mongodb_official:27017/mlflow_artifacts meriam-hfaidhia-4ds4-mlops_project-
mlflow_server
54f03fa59b931367dbff1b349ed749cdc3b7feffa5cf5da8bf392cbdb33810f3
[Pipeline] echo
✅ MLflow UI a démarré avec succès !
[Pipeline] echo
Accédez à l'interface MLflow à l'adresse : http://127.0.0.1:5000
```

The screenshot shows the MLflow UI interface. At the top, there is a navigation bar with the MLflow logo, 'Experiments', and 'Models' (which is underlined, indicating it is the active tab). Below the navigation bar, the title 'Registered Models' is displayed. A sub-instruction 'Share and manage machine learning models. Learn more' is present. There is a search bar with placeholder text 'Filter registered models by name or tags' and a magnifying glass icon. To the right of the search bar are two small icons: a gear and a question mark. The main content area displays a table of registered models. The columns are labeled 'Name', 'Latest version', 'Aliased versions', and 'Created by'. One row is visible in the table, showing 'churn_model' in the 'Name' column, 'Version 1' in the 'Latest version' column, and 'Created by' (the text is partially cut off). The entire interface has a dark theme with blue and white text.

Version 5

Registered At: 2025-02-24 01:07:18 Last Modified: 2025-02-24 01:07:18 Source

Stage (deprecated): Staging ⓘ

> Description [Edit](#)

> Tags

> Schema

Name	Type
Inputs (13)	
Account length (required)	double
International plan (required)	long
Voice mail plan (required)	long
Outputs (1)	
- (required)	Tensor (dtype: int64, shape: [-1])

Conteneur en docker

 meriamwa/mlflow-server

By [meriamwa](#) • Updated a month ago

Version	Registered at	Created by	Tags
Version 63	2025-02-27 19:20:35		Add
Version 62	2025-02-27 19:11:25		Add
Version 61	2025-02-27 19:02:36		Add
Version 60	2025-02-27 18:49:05		Add
Version 59	2025-02-27 18:08:33		Add
Version 58	2025-02-27 17:42:30		Add
Version 57	2025-02-27 17:22:37		Add
Version 56	2025-02-27 16:57:34		Add
Version 55	2025-02-26 17:11:54		Add
Version 54	2025-02-26 17:05:05		Add

2.1.10 Promotion du modèle

```

stage('Promote Model') {
steps {
    script {
        echo "🚀 Promotion du modèle..."
        sh """
            set -e
            cd ${env.PROJECT_DIR}
            . venv/bin/activate
            python main.py --promote Production --model_version 50
        """
    }
}
}

```

Stage : Promote Model

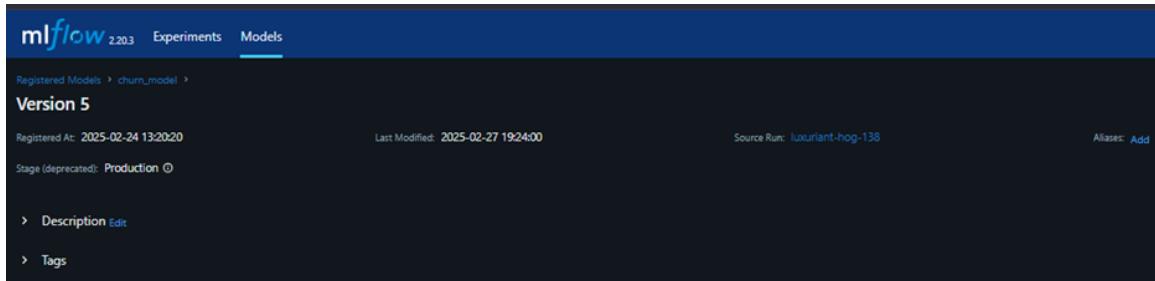
L'objectif de ce stage est de promouvoir une version spécifique du modèle vers l'environnement de production. Cette étape garantit que le modèle validé est officiellement marqué comme prêt pour une utilisation en conditions réelles.

- **Objectif** : Définir une version du modèle comme étant la version de production.

- **Étapes exécutées :**
 - **python main.py --promote Production --model_version 50** : Promotion de la version 50 du modèle vers l'environnement de production.
- **Résultat** : Le modèle promu devient la version officielle utilisée pour les prédictions en production.
- **Importance** : Cette étape est cruciale pour assurer la stabilité du système en garantissant que seule une version validée et performante du modèle est utilisée en production. Elle permet aussi de tracer l'historique des modèles et d'effectuer des rollbacks si nécessaire.

Résultat

```
+ python main.py --promote Production --model_version 5
2025-02-27 19:14:46,979 - INFO - ✓ MongoDB est déjà en cours d'exécution dans Docker.
2025-02-27 19:14:46,990 - INFO - ✓ Connexion à MongoDB réussie.
2025-02-27 19:14:46,990 - INFO - ✓ MLflow tracking URI set to: sqlite:///mlflow.db
2025-02-27 19:14:47,880 - INFO - Context impl SQLiteImpl.
2025-02-27 19:14:47,880 - INFO - Will assume non-transactional DDL.
2025-02-27 19:14:47,949 - INFO - ✓ Expérience 'churn' existe déjà. ID: 3
2025-02-27 19:14:47,956 - INFO - ✓ Expérience actuelle définie sur 'churn'.
/home/meriam/meriam-hfaidhia-4DS4-mlops_project/main.py:162: FutureWarning:
`mlflow.tracking.client.MlflowClient.transition_model_version_stage` is deprecated since 2.9.0. Model registry stages
will be removed in a future major release. To learn more about the deprecation of model registry stages, see our
migration guide here: https://mlflow.org/docs/latest/model-registry.html#migrating-from-stages
client.transition_model_version_stage(
2025-02-27 19:14:48,051 - INFO - ✓ Modèle churn_model version 5 promu au stage Production.
```



2.2 Déploiement Continue (CD)

2.2.1 FastAPI conteneur

```
stage('Start FastAPI Application') {
    steps {
        script {
            echo "🚀 Démarrage de l'application FastAPI..."

            // Debug: Ensure the PROJECT_DIR is correct
            echo "PROJECT_DIR is: ${env.PROJECT_DIR}"

            sh """
                set -e
                cd ${env.PROJECT_DIR}
            """

            // Stop and remove existing containers if they exist
            sh """
                docker-compose -f ${env.PROJECT_DIR}/docker-compose-app.yml down || true
            """

            // Start the FastAPI application
            sh """
                docker-compose -f ${env.PROJECT_DIR}/docker-compose-app.yml up -d
            """

            // Wait for FastAPI to start
            sh """
                until curl -sSF http://localhost:8000/docs; do
                    echo "En attente du démarrage de l'application FastAPI..."
                    sleep 10
                done
            """
        }
    }
}
```

```
// Check if FastAPI is running
sh """
if ! curl -sSF http://localhost:8000/docs; then
    echo "🔴 Échec du démarrage de l'application FastAPI."
    exit 1
fi
"""

echo "✅ L'application FastAPI a démarré avec succès !"
echo "Accédez à l'API FastAPI à l'adresse : http://localhost:8000/docs"

// Vérifier les prédictions dans la base de données
echo "🔵 Vérification des prédictions dans la base de données..."

sh """
    docker-compose -f ${env.PROJECT_DIR}/docker-compose-app.yml exec db psql -U postgres -d mllops_db -c "SELECT * FROM
"""

echo "✅ Vérification des prédictions terminée."
```

Stage : Start FastAPI Application

L'objectif de ce stage est de démarrer l'application FastAPI à l'aide de Docker Compose. Cette étape garantit que l'API est opérationnelle et accessible avant de passer aux étapes suivantes du pipeline.

- **Objectif** : Démarrer et vérifier le bon fonctionnement de l'application FastAPI.
- **Étapes exécutées** :
 - **Vérification du répertoire de travail** : Affichage du répertoire PROJECT_DIR pour s'assurer de sa validité.

- **Arrêt des conteneurs existants** : Arrêt et suppression des conteneurs existants à l'aide de la commande :

```
docker-compose -f docker-compose-app.yml down
```

- **Démarrage de l'application** : Lancement des services définis dans `docker-compose-app.yml` avec la commande :

```
docker-compose -f docker-compose-app.yml up -d
```

- **Vérification du démarrage** : Attente du démarrage de FastAPI en testant l'accès à `http://localhost:8000/docs`.
- **Validation de l'exécution** : Vérification finale que FastAPI est bien accessible, sinon affichage d'un message d'erreur.
- **Vérification des prédictions dans la base de données** : Exécution d'une requête SQL pour lister les prédictions enregistrées :

```
docker-compose -f docker-compose-app.yml exec db psql -U postgres -d mlops_db -
```

- **Résultat** : L'application FastAPI est accessible via `http://localhost:8000/docs`, et les prédictions en base de données sont vérifiées.
- **Importance** : Cette étape est essentielle pour s'assurer que l'API est correctement déployée, fonctionne comme prévu et que les prédictions sont bien enregistrées en base de données.

Résultat

```

Accédez à l'API FastAPI à l'adresse : http://localhost:8000/docs
[Pipeline] echo
  Vérification des prédictions dans la base de données...
[Pipeline] sh
+ docker-compose -f /home/meriam/meriam-hfaidhia-4DS4-mlops_project/docker-compose-app.yml exec db psql -U postgres -d mlops_db -c SELECT * FROM predictions;
   id |          features           |      prediction |      timestamp
-----+-----+-----+-----+
   1 | [1, 2, 4, 16, 17, 2, 3, 45, 3, 10, 11, 1, 0] |          1 | 2025-02-27 15:25:52.84394
   2 | [1, 2, 4, 1, 17, 2, 3, 45, 3, 10, 11, 0, 0] |          0 | 2025-02-27 15:28:05.943435
   3 | [1, 2, 3, 14, 13, 4, 8, 10, 4, 3, 6, 0, 1] |          0 | 2025-02-27 16:46:14.956936
   4 | [1, 2, 3, 4, 5, 6, 1, 13, 11, 10, 2, 0, 1] |          0 | 2025-02-27 16:46:34.874477
   5 | [1, 2, 3, 4, 5, 6, 1, 13, 11, 10, 2, 0, 1] |          0 | 2025-02-27 17:12:42.103122
   6 | [1, 2, 3, 4, 5, 6, 1, 13, 11, 10, 2, 0, 1] |          0 | 2025-02-27 17:53:06.044526
   7 | [1, 2, 3, 4, 5, 6, 1, 13, 11, 10, 2, 0, 1] |          0 | 2025-02-27 18:33:53.138173
   8 | [1, 2, 3, 4, 5, 6, 1, 13, 11, 10, 2, 0, 1] |          0 | 2025-03-04 11:10:05.074356
   9 | [1, 2, 3, 4, 5, 6, 1, 13, 11, 10, 2, 0, 1] |          0 | 2025-03-04 13:12:50.940062
  10 | [1, 2, 3, 4, 5, 6, 1, 13, 11, 10, 2, 0, 1] |          0 | 2025-03-04 13:29:18.446889
  11 | [1, 2, 3, 4, 5, 6, 1, 13, 11, 10, 2, 0, 1] |          0 | 2025-03-04 13:59:43.531536
  12 | [1, 2, 3, 4, 5, 6, 1, 13, 11, 10, 2, 0, 1] |          0 | 2025-03-04 14:10:45.320556
  13 | [1, 2, 3, 4, 5, 6, 1, 13, 11, 10, 2, 0, 1] |          0 | 2025-03-04 14:40:17.781753
  14 | [1, 2, 3, 4, 5, 6, 1, 13, 11, 10, 2, 0, 1] |          0 | 2025-03-04 14:50:51.059252

```

FastAPI 0.1.0 OAS 3.1

[/openapi.json](#)

default

POST [/predict/](#) Predict

Parameters

No parameters

Request body required

application/json

```
{
  "features": [
    1,2,3,14,13,4,8,10,4,3,6,0,1
  ]
}
```

Request URL

<http://localhost:8000/predict/>

Server response

Code	Details
200	Response body <pre>{ "prediction": 0 }</pre> <p>Download</p> Response headers <pre>content-length: 16 content-type: application/json date: Thu, 27 Feb 2025 16:46:14 GMT server: uvicorn</pre>

Responses

Conteneur en docker



meriamwa/fastapi-mlops

By [meriamwa](#) · Updated a month ago

2.2.2 Test de l'application FastAPI

```

stage('Test Prediction') {
steps {
    script {
        echo "✓ Test de la prédiction via l'API FastAPI..."
        def response = sh(script: """
            set -e
            curl --request POST \
            --url http://192.168.93.6:8000/predict/ \
            --header 'Content-Type: application/json' \
            --data '{
                "features": [1, 2, 3, 4, 5, 6, 1, 13, 11, 10, 2, 0, 1]
            }'
        """
        , returnStdout: true).trim()

        // Utiliser jq pour parser la réponse et obtenir la prédiction
        def prediction = sh(script: "echo '${response}' | jq '.prediction'", returnStdout: true).trim()

        // Afficher le résultat en fonction de la prédiction
        if (prediction == "1") {
            echo "✓ Prédiction : Churn"
        } else if (prediction == "0") {
            echo "✓ Prédiction : Not Churn"
        } else {
            echo "✗ Erreur : Prédiction non valide"
        }
    }
}

```

Test Prediction

L'objectif de ce stage est de tester la prédiction du modèle à l'aide de l'API FastAPI en envoyant des données au point de terminaison `/predict/`. Cette étape permet de valider que l'API fonctionne correctement et renvoie des prédictions précises.

— **Objectif** : Tester la fonctionnalité de prédiction du modèle via l'API FastAPI.

— **Étapes exécutées** :

— **Envoi de la requête POST à l'API** : Envoi des données de caractéristiques (features) à l'API pour obtenir une prédiction.

```

curl --request POST \
--url http://192.168.93.6:8000/predict/ \
--header 'Content-Type: application/json' \

```

```
--data '{"features": [1, 2, 3, 4, 5, 6, 1, 13, 11, 10, 2, 0, 1]}
```

- **Analyse de la réponse** : Utilisation de jq pour analyser la réponse JSON et extraire la prédiction :

```
jq '.prediction'
```

- **Affichage de la prédiction** : En fonction de la valeur de la prédiction (1 pour churn ou 0 pour not churn), un message est affiché :

- **Prédiction : Churn** : Si la prédiction est égale à 1, un message de succès est affiché.
- **Prédiction : Not Churn** : Si la prédiction est égale à 0, un message de succès est affiché.
- **Erreur** : Si la prédiction n'est pas valide, un message d'erreur est affiché.

- **Résultat** : La prédiction du modèle est retournée par l'API et affichée dans la console Jenkins, indiquant si l'utilisateur risque de "churn" ou non.
- **Importance** : Cette étape est cruciale pour vérifier que l'API de prédiction fonctionne correctement et renvoie des résultats cohérents, permettant ainsi de valider la précision du modèle déployé.

Résultat

```
✓ Test de la prédiction via l'API FastAPI...
[Pipeline] sh
+ set -e
+ curl --request POST --url http://192.168.93.6:8000/predict/ --header Content-Type: application/json --data {
    "features": [1, 2, 3, 4, 5, 6, 1, 13, 11, 10, 2, 0, 1]
}
% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
          Dload  Upload   Total   Spent    Left  Speed
0       0      0      0      0      0      0      0      0      0
100  110  100    16  100     94    163    960      --:-- --:--:--:--:--:--  1122
100  110  100    16  100     94    162    957      --:-- --:--:--:--:--:--  1111
[Pipeline] sh
+ echo {"prediction":0}
+ jq .prediction
[Pipeline] echo
✓ Prédiction : Not Churn
```

Retrain

```
stage('Retrain Model') {
    steps {
        script {
            echo "⌚ Réentraînement du modèle..."
            sh """
                set -e
                curl --request POST \
                --url http://192.168.93.6:8000/retrain/ \
                --header 'Content-Type: application/json' \
                --data '{
                    "n_estimators": 100,
                    "max_depth": 10,
                    "min_samples_split": 2,
                    "train_path": "churn-bigml-20.csv",
                    "test_path": "churn-bigml-80.csv"
                }'
            """
        }
    }
}
```

Résultat

```
⌚ Réentraînement du modèle...
[Pipeline] sh
+ set -e
+ curl --request POST --url http://192.168.93.6:8000/retrain/ --header Content-Type: application/json --data {
    "n_estimators": 100,
    "max_depth": 10,
    "min_samples_split": 2,
    "train_path": "churn-bigml-20.csv",
    "test_path": "churn-bigml-80.csv"
}
% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
          Dload  Upload Total Spent   Left Speed
0       0      0      0      0      0      0      0      0      0
100  377  100     78  100   299    110    424      0      0      0      535
{"message": "Modèle réentraîné avec succès", "accuracy": 0.9181094992980814}
```

2.2.3 Test de l'application FlaskAPI

```

stage('Test Prediction for FlaskApp') {
    steps {
        script {
            echo "✓ Test de la prédiction via l'API Flask..."
            def response = sh(script: """
                set -e
                curl --request POST \
                --url http://192.168.93.6:5001/predict \
                --data 'account_length=100&number_vmail_messages=5&total_day_calls=150&total_day_charge=55.5&total_eve_
                --silent
                """, returnStdout: true).trim()

            // Extraire le résultat du HTML
            def predictionMatch = response =~ /<p>Le résultat de la prédiction est : <strong>(.*)</strong></p>/
            if (predictionMatch.find()) {
                def prediction = predictionMatch[0][1]
                echo "✓ Prédiction : ${prediction}"
            } else {
                echo "✗ Erreur : Impossible d'extraire la prédiction"
            }
        }
    }
}

```

FIGURE 2.1 : Test de FlaskAPI

Test Prediction

L'objectif de ce *stage* est de tester la prédiction du modèle à l'aide de l'API Flask en envoyant des données simulées via une requête HTTP **POST**. Le test est effectué automatiquement à l'aide d'un script exécuté dans un pipeline Jenkins.

Résultat : Le script envoie une requête HTTP contenant les caractéristiques d'un client. La réponse, de type HTML, est ensuite analysée pour extraire la valeur de prédiction. Cette étape permet de valider le bon fonctionnement de l'API en environnement déployé.

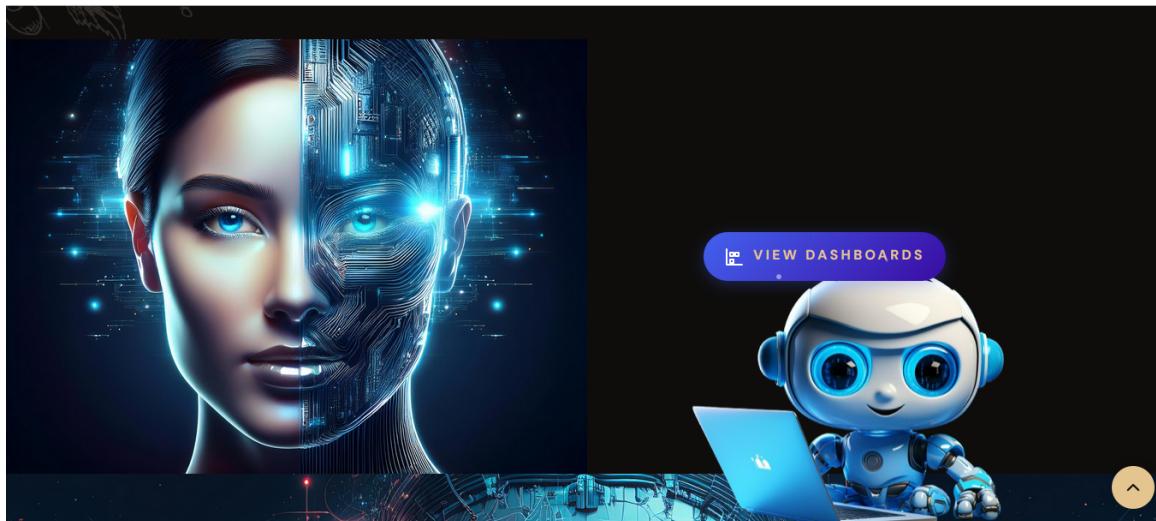
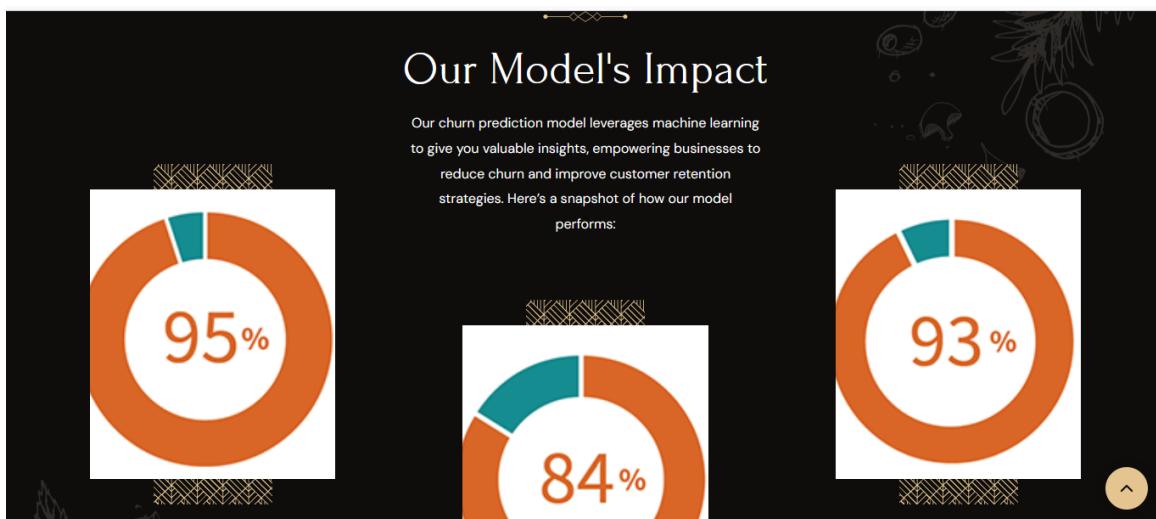
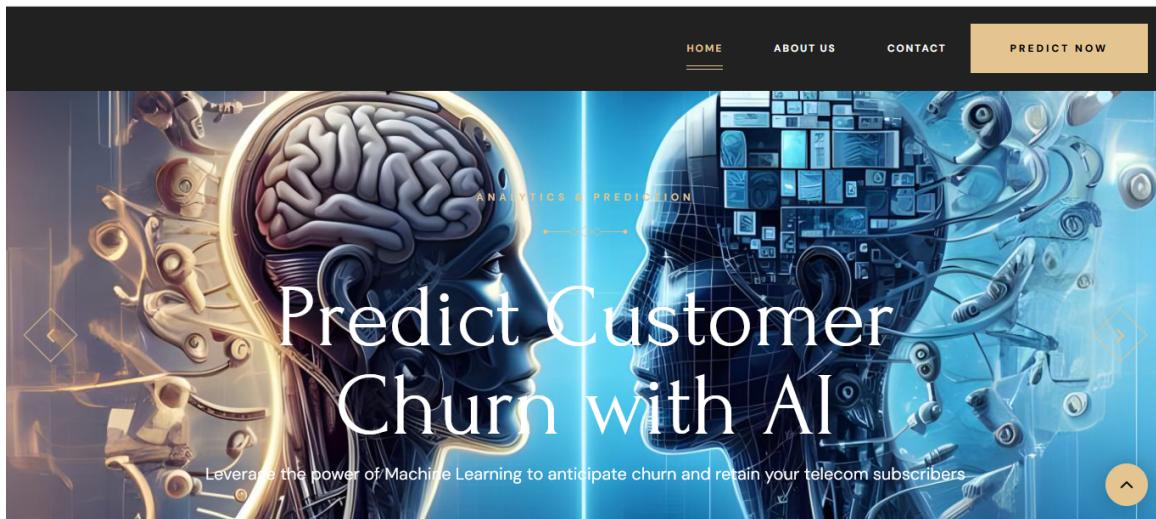
Résultat

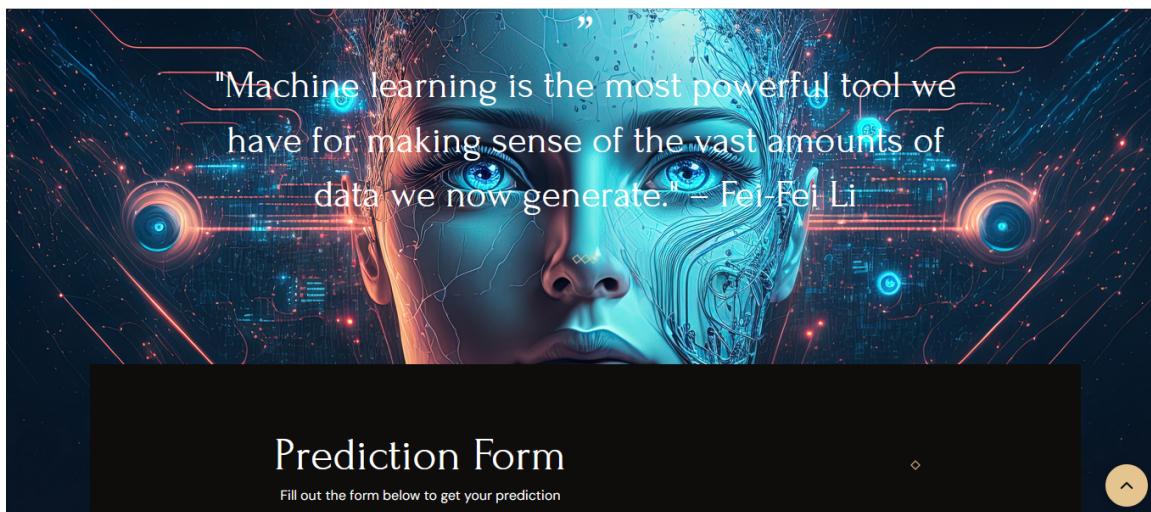
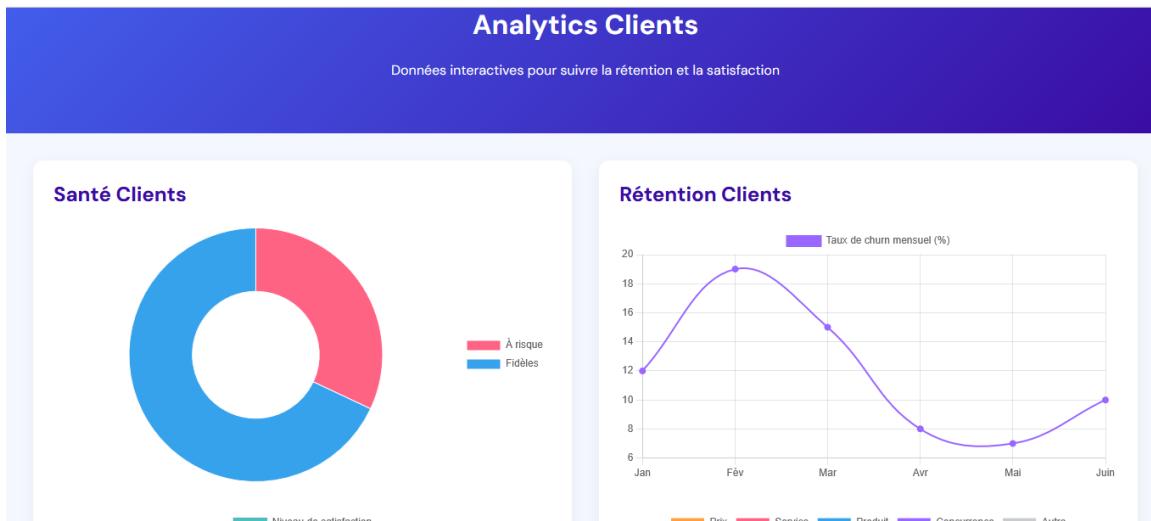
```

✓ Test de la prédiction via l'API Flask...
[Pipeline] sh
+ set -e
+ curl --request POST --url http://192.168.93.6:5001/predict --data
account_length=100&number_vmail_messages=5&total_day_calls=150&total_day_charge=55.5&total_eve_calls=200&total_eve_charge=30.7&total_night_calls=300&total_night_charge=45.2&total_intl_calls=30&total_intl_charge=15.6&customer_service_calls=1&international_plan=1&voice_mail_plan=1 --silent
[Pipeline] echo
✓ Prédiction : Churn

```

Interfaces de flask





✖

✖

✖

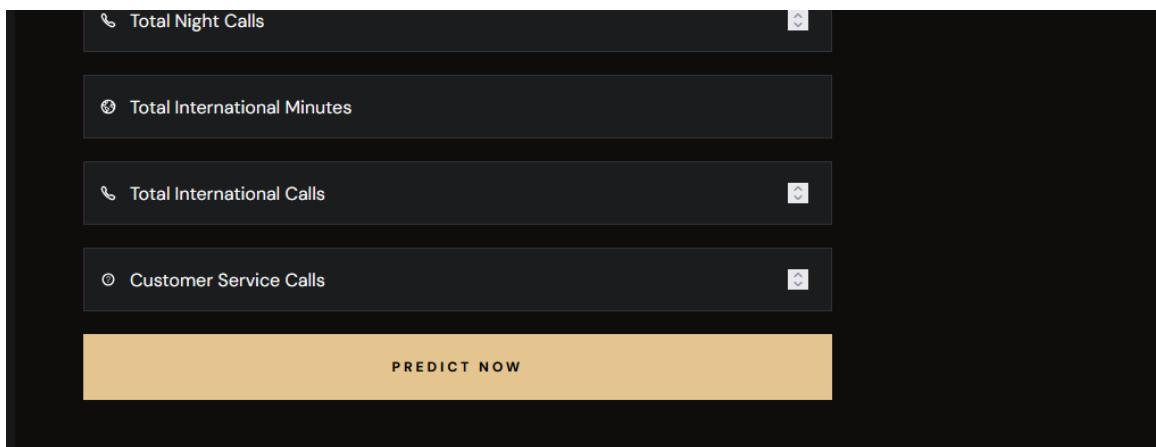
✖

✖

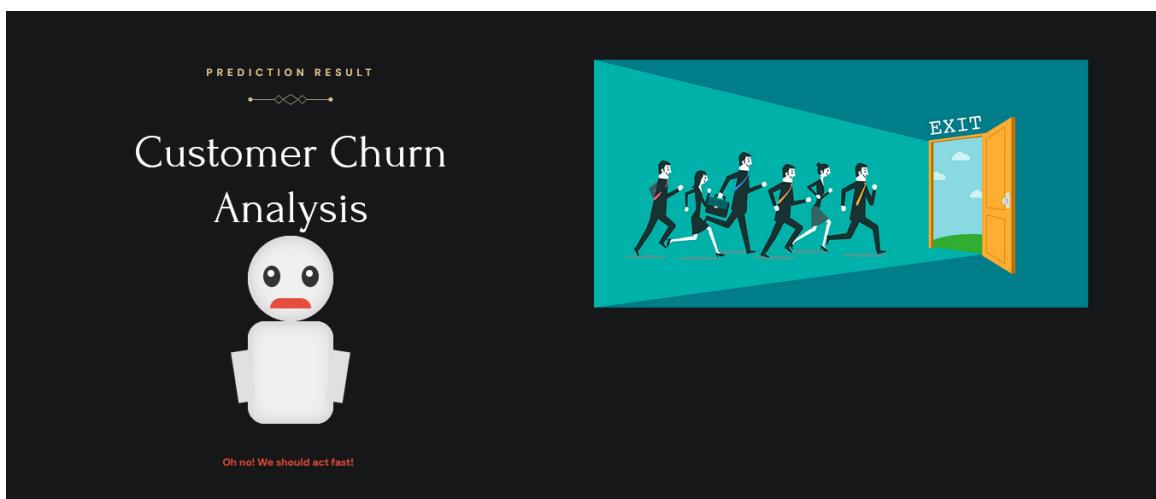
✖

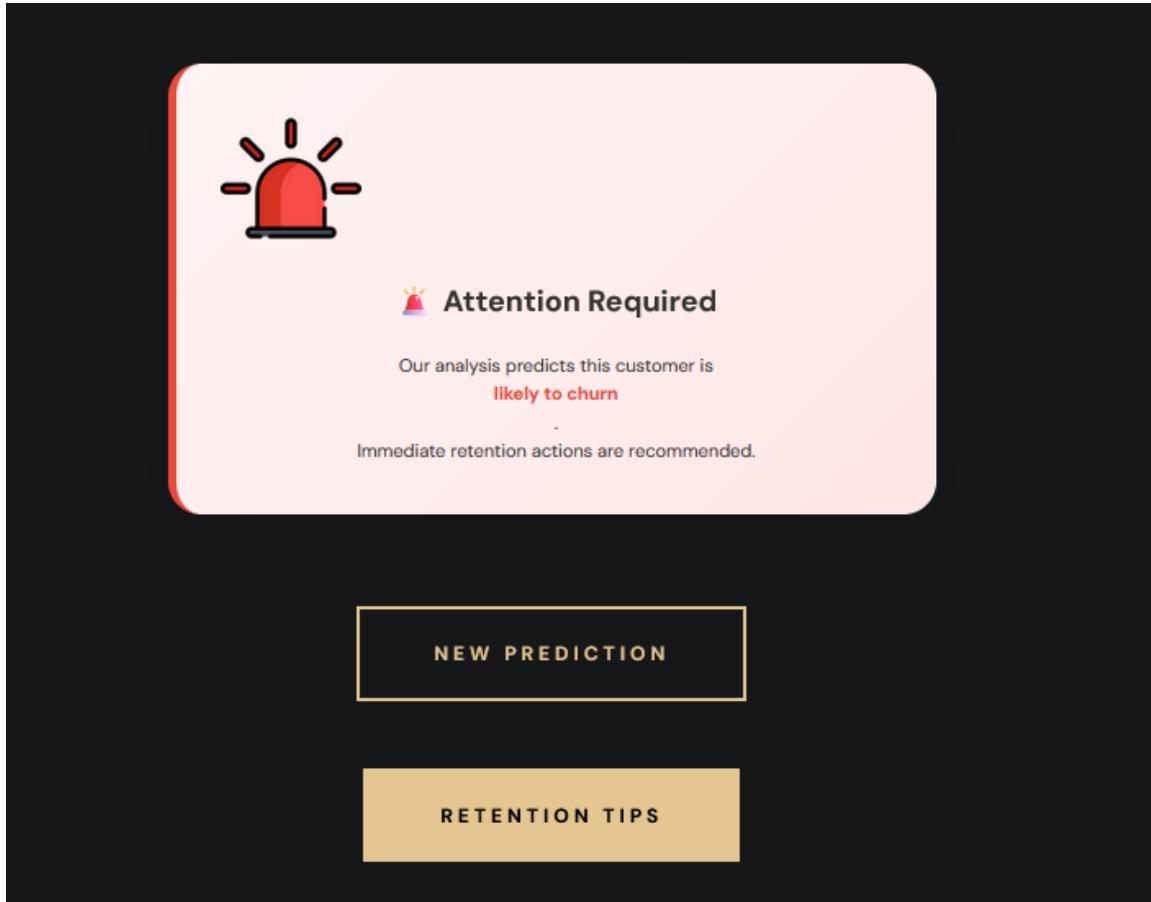
✖

✖



Si le résultat est churn :





Customer Retention Strategies

Proven techniques to prevent customer churn and rebuild loyalty

⌚ Immediate Action Recommended
Customers identified as churn risks require prompt intervention for best results



1. Personalized Retention Offer
Create customized incentives based on the customer's usage patterns and value to your business.

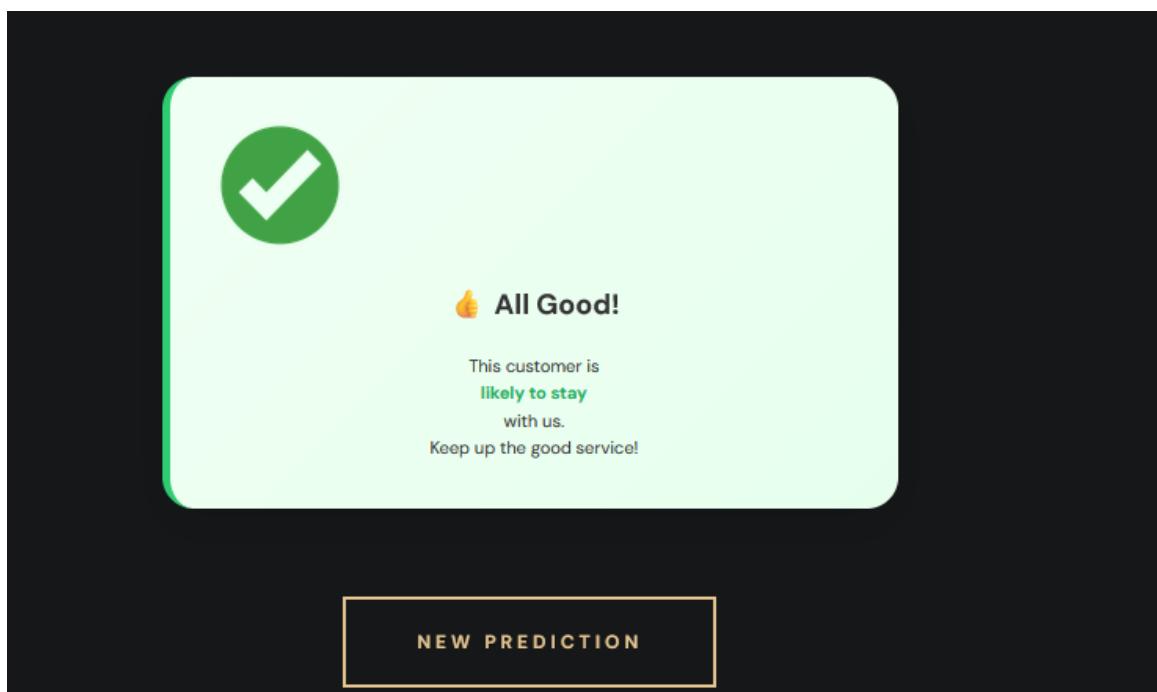
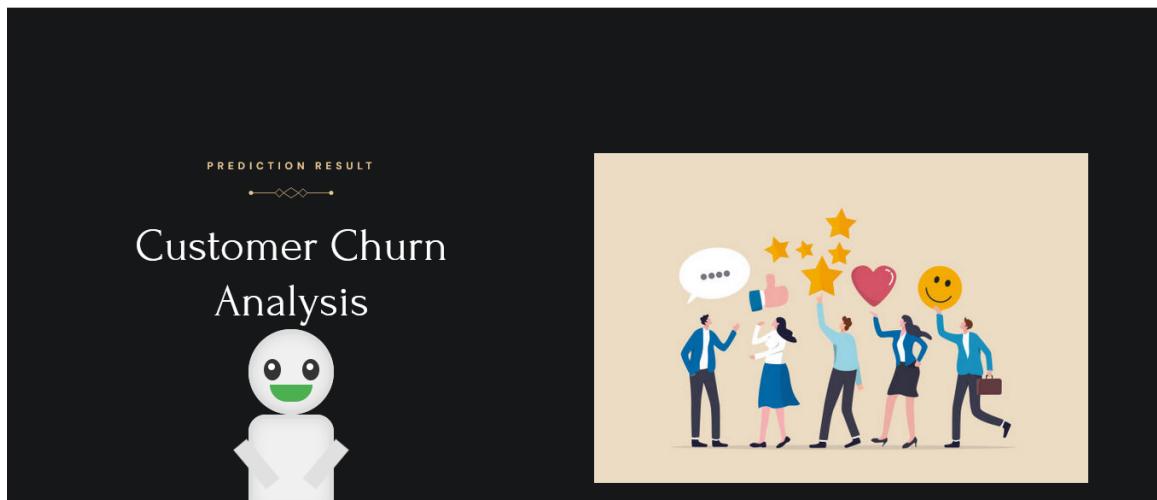


2. Win-Back Campaign
Develop a multi-channel campaign to re-engage at-risk customers with targeted messaging.

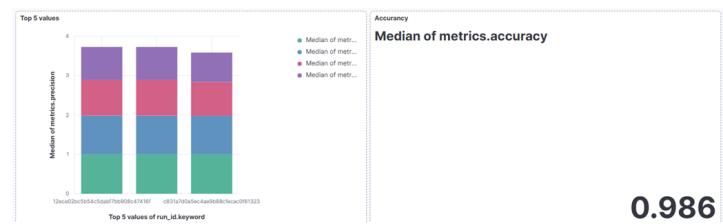


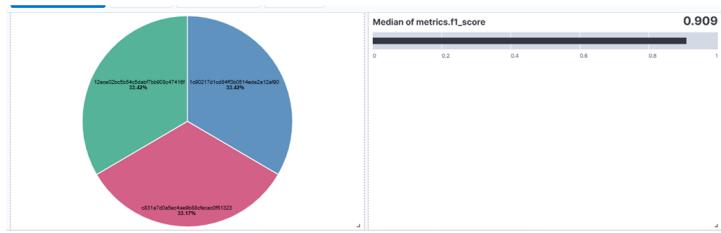
3. Dedicated Success Manager
Assign a personal contact to high-value at-risk customers to address their concerns directly.

Si le résultat n'est pas churn :



2.2.4 Kibana





Kibana est un outil open source de visualisation de données qui fonctionne en étroite collaboration avec Elasticsearch. Il permet d'explorer, d'analyser et de représenter visuellement les données stockées dans Elasticsearch à l'aide de tableaux de bord interactifs, de graphiques, de cartes et d'autres éléments visuels. Grâce à Kibana, les utilisateurs peuvent interroger les données en temps réel et obtenir des insights pertinents sans avoir besoin d'écrire des requêtes complexes. Il est largement utilisé dans les systèmes de monitoring, de logs et d'analyse de performance.

2.2.5 Post

```
post {
    always {
        echo "Nettoyage ou actions post-exécution"
        sh "rm -rf ${env.PROJECT_DIR}/venv"

        // Copier les fichiers dans le workspace
        sh "cp ${env.PROJECT_DIR}/model_card.md ."
        sh "cp ${env.PROJECT_DIR}/performance_report.md ."
        sh "cp ${env.PROJECT_DIR}/performance_report.md_confusion_matrix.png ."

        // Vérifier l'existence des fichiers dans le workspace
        sh "ls -l"

        script {
            def jobName = env.JOB_NAME
            def buildNumber = env.BUILD_NUMBER
            def pipelineStatus = currentBuild.result ?: 'UNKNOWN'
            def bannerColor = pipelineStatus.toUpperCase() == 'SUCCESS' ? 'green' : 'red'

            def body = """
<html>
<body>
<div style="border: 4px solid ${bannerColor}; padding: 10px;>
<h2>${jobName} Build ${buildNumber}</h2>
<div style="background-color: ${bannerColor}; padding: 10px;>
<h3 style="color: white;>Pipeline Status: ${pipelineStatus.toUpperCase()}</h3>
</div>
<p>Check the <a href="${env.BUILD_URL}">console output</a>.</p>
</div>
</body>
</html>"""
        }
    }
}
```

```
// Utiliser un pattern GLOB valide
emailext(
    subject: "${jobName} Build ${buildNumber} ${pipelineStatus}",
    body: body,
    to: 'meriamhfaidha@gmail.com',
    from: 'jenkins@example.com',
    replyTo: 'jenkins@example.com',
    mimeType: 'text/html',
    attachmentsPattern: "model_card.md, performance_report.md,performance_report.md_confusion_matrix.png", //
)
}

success {
    echo "✅ Pipeline exécuté avec succès!"
}

failure {
    echo "❌ Pipeline échoué."
}
```

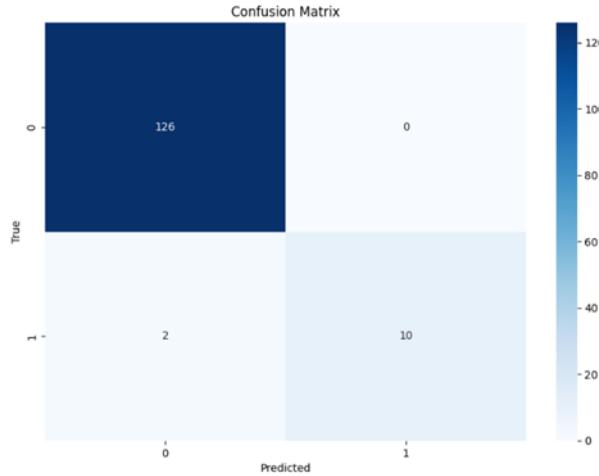
Le bloc `post` est une section importante dans un pipeline Jenkins. Il est exécuté après la fin du pipeline, que celui-ci ait réussi ou échoué. Dans notre cas, nous utilisons le bloc `post { always }` pour effectuer des actions de nettoyage et de notification, quelles que soient les conditions d'exécution.

Voici les principales actions effectuées :

- **Nettoyage du projet** : suppression de l'environnement virtuel `venv` pour libérer de l'espace

disque.

- **Copie de fichiers** : les fichiers de résultats (comme `model_card.md` ou les rapports de performance) sont copiés dans le workspace Jenkins afin qu'ils puissent être archivés ou envoyés.
- **Notification par email** : un message HTML personnalisé est généré contenant le nom du job, le numéro de build, le statut de l'exécution, et un lien vers les logs de Jenkins. Le code utilise la couleur verte pour un succès et rouge en cas d'échec.



```
# performance_report.Rmd > R rapport de Performance > R ## Rapport de Classification
1  # Rapport de Performance
2
3  ## Matrice de Confusion
4  ! [Confusion Matrix] (/home/meriam/meriam-hfaidhia-4DS4-mlops_project/performance_report.md_confusion_matrix.png)
5
6  ## Rapport de classification
7  | precision | recall | f1-score | support |
8  |:-----:|:-----:|:-----:|:-----:|
9  | 0 | 0.98 | 1.00 | 0.99 | 126
10 | 1 | 1.00 | 0.83 | 0.91 | 12
11
12 | accuracy | | | | 138
13 | macro avg | 0.99 | 0.92 | 0.95 | 138
14 | weighted avg | 0.99 | 0.99 | 0.98 | 138
15
```

```
# model_card.md > R Fiche de Modèle > R ## Métriques de Performance
1  # Fiche de Modèle
2
3  ## Hyperparamètres
4  {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None,
5   'max_features': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0,
6   'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'monotonic_cst': None,
7   'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start': False}
8
9  ## Métriques de Performance
10 | | 0 |
11 |:-----:|:-----:|
12 | accuracy | 0.985507 |
13 | precision | 1 |
14 | recall | 0.833333 |
15 | f1_score | 0.900091 |
```

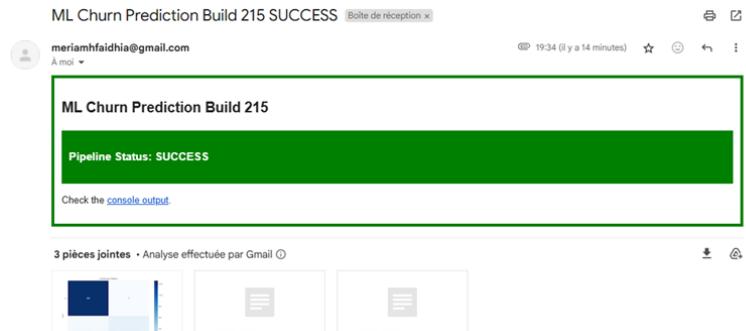
En cas d'échec



En cas d'interruption



En cas de succès :



Conclusion

Ce chapitre a présenté les mécanismes clés du pipeline MLOps, démontrant comment l'intégration continue et le déploiement continu assurent la robustesse et l'efficacité du cycle de vie des modèles. Ces processus automatisés constituent le socle essentiel pour des déploiements fiables et itératifs en environnement de production

Conclusion générale

Ce projet a permis de mettre en place un pipeline MLOps complet pour la prédiction du churn des clients télécoms, en intégrant les meilleures pratiques d'automatisation, de déploiement et de monitoring. Grâce à l'approche CI/CD , nous avons pu garantir une mise en production fluide, une mise à jour sécurisée des modèles et une surveillance proactive des performances.

L'utilisation de l'intelligence artificielle, couplée à une infrastructure scalable et reproductible, offre aux entreprises de télécommunications un avantage concurrentiel en leur permettant d'anticiper les désabonnements et d'adapter leurs stratégies de fidélisation. Cependant, plusieurs défis ont émergé, notamment la gestion des données en temps réel, la détection des biais dans les prédictions et l'optimisation des coûts infrastructurels.

En conclusion, ce pipeline MLOps constitue une solution robuste et évolutive pour la prédiction du churn, démontrant que l'alliance entre Machine Learning et automatisation opérationnelle est un levier clé pour la transformation data-driven des entreprises.

