Pipeline MLOps avec DVC, MLflow et Streamlit pour la prédiction de la demande en retail

Rapport technique

21 décembre 2025

Table des matières

Table des figures

1 Introduction

1.1 Contexte et motivation

La prédiction de la demande en retail constitue un enjeu central pour les entreprises de distribution. Elle conditionne la capacité à anticiper les ventes, à optimiser les niveaux de stock et à réduire les ruptures comme les surplus. Dans un contexte de concurrence accrue et de marges limitées, une estimation fiable des ventes hebdomadaires permet d'aligner les approvisionnements sur les besoins réels, d'améliorer la satisfaction client et de réduire les coûts logistiques.

Cependant, construire un modèle performant ne suffit pas : il faut également assurer la reproductibilité des données, le suivi des expériences, la traçabilité des décisions, ainsi que le déploiement et la maintenance des modèles en production. Ces exigences sont au cœur des pratiques MLOps (Machine Learning Operations), qui visent à industrialiser le cycle de vie des modèles.

1.2 Objectifs du projet

Le projet présenté dans ce rapport vise à construire un pipeline MLOps complet pour la prédiction de la demande en retail, en intégrant DVC pour le versioning des données et la reproductibilité, MLflow pour le tracking des expériences et le Model Registry, une API FastAPI pour le serving du modèle, une interface Streamlit pour la consommation des prédictions, et Docker pour garantir une exécution reproductible d'API. Les objectifs principaux sont :

— Intégrer DVC et MLflow dans un workflow MLOps unifié.
— Gérer le cycle de vie complet des modèles.
— Automatiser le passage de l'expérimentation à la production.
— Mettre à disposition un service de prédiction via API et interface utilisateur.

1.3 Organisation du rapport

Le rapport est structuré en neuf sections. Après une introduction générale, il présente l'architecture du système, la gestion des données avec DVC, l'entraînement et l'évaluation des modèles, le tracking avec MLflow, la gestion du cycle de vie avec le Model Registry, le déploiement via FastAPI et Docker, l'interface Streamlit, puis conclut par un bilan et des perspectives.

2 Architecture générale du système

2.1 Vue d'ensemble du pipeline MLOps

Le pipeline suit un flux logique allant des données brutes jusqu'à la consommation des prédictions par l'utilisateur. Les étapes principales sont : ingestion des données, prétraitement, entraînement et évaluation, logging dans MLflow, enregistrement dans le Model Registry, déploiement via FastAPI, et interaction via Streamlit.

Un schéma d'architecture permet de visualiser cette chaîne de valeur et de situer les responsabilités de chaque composant.

Figure : Diagramme d'architecture montrant le flux données → DVC → entraînement → évaluation → MLflow → Model Registry → API → Streamlit.

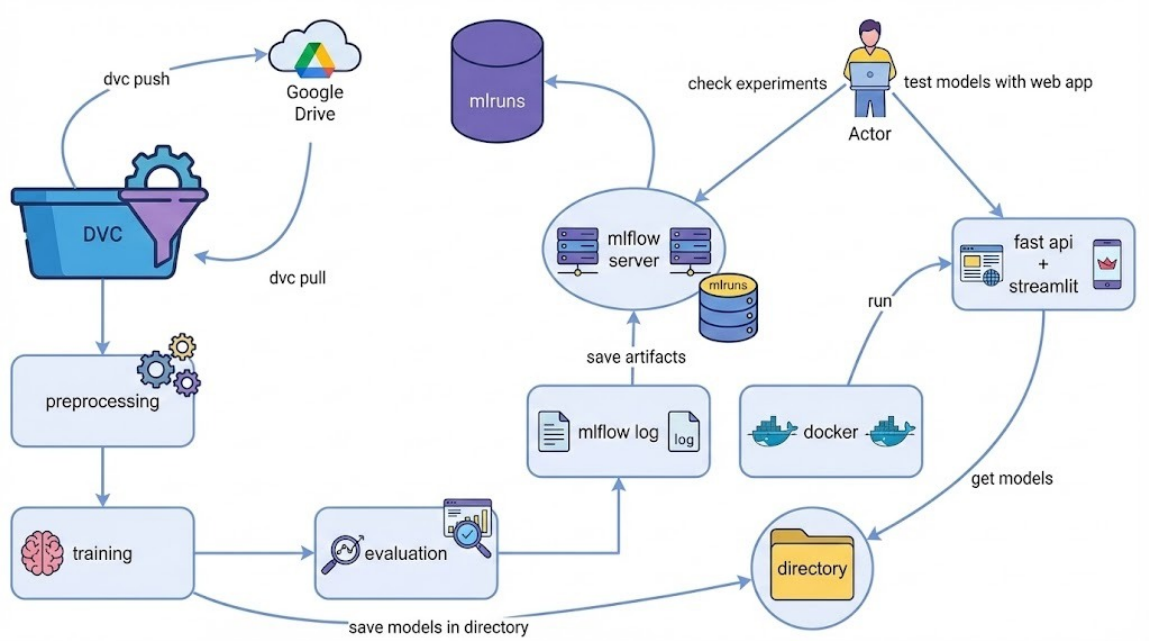

FIGURE 1 – Vue d'ensemble du pipeline MLOps

2.2 Choix technologiques

Les technologies retenues répondent à des besoins spécifiques :

— **Git / GitHub** : versioning du code et collaboration.

FIGURE 2 – Git Logo

FIGURE 3 – GitHub Logo

— **DVC** : versioning des données, reproductibilité du pipeline.

FIGURE 4 – DVC logo

— **MLflow** : tracking des expériences, stockage des artefacts, Model Registry.

FIGURE 5 – MLflow logo

— **Docker** : conteneurisation et déploiement reproductible d'API.

FIGURE 6 – Docker logo

— **FastAPI** : API de prédiction performante et documentée.

FIGURE 7 – FastAPI logo

— **Streamlit** : interface simple pour la visualisation et la prédiction.

FIGURE 8 – Streamlit logo

Ce choix vise un équilibre entre robustesse, simplicité d'intégration et adoption industrielle.

2.3 Structure du projet

Le dépôt est organisé par domaines fonctionnels : scripts de pipeline, artefacts de modèles, métriques, et services de déploiement. Cette structuration facilite la maintenance et la lecture du workflow.

```
|-- .dvc/
|   |-- config
|-- .github/
|   `-- workflows/
|       `-- cml.yaml
|-- app/
|   |-- __init__.py
|   `-- main.py
|-- data/
|   |-- train.csv.dvc
|   |-- features.csv.dvc
|   |-- stores.csv.dvc
|-- metrics/
|-- models/
|-- scripts/
|   |-- preprocessing.py
|   |-- training.py
|   |-- evaluating.py
|   `-- mlflow_log.py
|-- Dockerfile
|-- Dockerfile.streamlit
|-- docker-compose.yml
|-- dvc.yaml
|-- dvc.lock
|-- requirements.txt
|-- requirements-streamlit.txt
|-- streamlit_app.py
`-- readme.md
```

3 Gestion des données et pipeline DVC

3.1 Description des données

Les données proviennent de trois fichiers principaux :

— **train.csv** : ventes hebdomadaires par magasin et département, variable cible *Weekly_Sales*.

— **features.csv** : variables exogènes (température, prix du carburant, CPI, chômage, etc.).

— **stores.csv** : informations statiques sur les magasins (type, taille).

Les variables explicatives incluent des indicateurs temporels, économiques et structurels.

3.2 Préprocessing et feature engineering

Le prétraitement comprend :

— Fusion des tables par magasin et date.

— Gestion des valeurs manquantes (imputation, substitution par zéro).

— Encodage des variables catégorielles.

— Création de variables temporelles (année, mois, semaine).

— Séparation train/validation selon l'ordre temporel.

Cette stratégie préserve la causalité temporelle, essentielle en prévision de la demande.

3.3 Versioning des données avec DVC

DVC permet de tracker les fichiers de données lourds sans les stocker dans Git. Les fichiers '.dvc' référencent les données, tandis que le cache DVC assure la reproductibilité. Les bénéfices incluent :

— Synchronisation des données entre collaborateurs.
— Reproductibilité stricte des expériences.
— Comparaison des résultats entre versions de données.

3.4 Pipeline DVC

Le pipeline est défini en trois stages principaux : *preprocess*, *train*, *eval*. Chaque stage décrit ses dépendances et ses sorties.

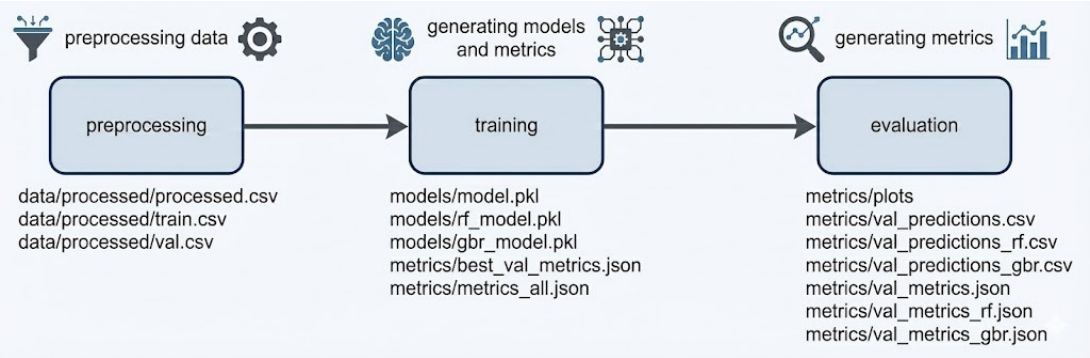

FIGURE 9 – Pipeline DVC et dépendances

Les commandes clés incluent :
— `dvc repro` : exécution du pipeline complet.
— `dvc status` : vérification des changements.
— `dvc metrics show` : comparaison des métriques.

4 Entraînement et évaluation des modèles

4.1 Modèles implémentés

Deux modèles sont comparés :
— **Random Forest Regressor** : robuste aux non-linéarités et interactions.
— **Gradient Boosting Regressor** : combinaison séquentielle d'arbres faibles.

4.2 Pipeline de machine learning

Le pipeline utilise `sklearn.Pipeline` pour intégrer prétraitement et modèle, garantissant la cohérence entre entraînement et prédiction :
— Imputation numérique (médiane).
— Encodage catégoriel (One-Hot).
— Modèle régressif.

4.3 Métriques d'évaluation

Trois métriques principales sont utilisées :
— **RMSE** : sensibilité aux grandes erreurs.
— **MAE** : erreur moyenne absolue.
— **R^2** : proportion de variance expliquée.

5 Tracking des expériences avec MLflow (3 pages)

5.1 Rôle de MLflow dans le pipeline

DVC assure la reproductibilité, tandis que MLflow fournit une couche d'analyse des runs, de comparaison et de stockage des artefacts. MLflow joue donc un rôle complémentaire en matière de gouvernance des expériences.

5.2 Configuration de MLflow

Le tracking URI peut être local (fichier) ou distant. Les runs sont regroupés par expérience et stockent :

— Paramètres du modèle.
— Métriques.
— Artefacts (modèles, métriques, graphiques).

5.3 Logging des expériences

Chaque modèle (Random Forest, Gradient Boosting) est loggé dans un run distinct. Les artefacts incluent :

— Modèles sérialisés.
— Fichiers de métriques.
— Prédictions de validation.
— Graphiques d'évaluation.

5.4 Analyse et comparaison des runs

L'interface MLflow permet de comparer les runs et de sélectionner le meilleur modèle. Les critères de comparaison incluent les métriques et la cohérence des performances.

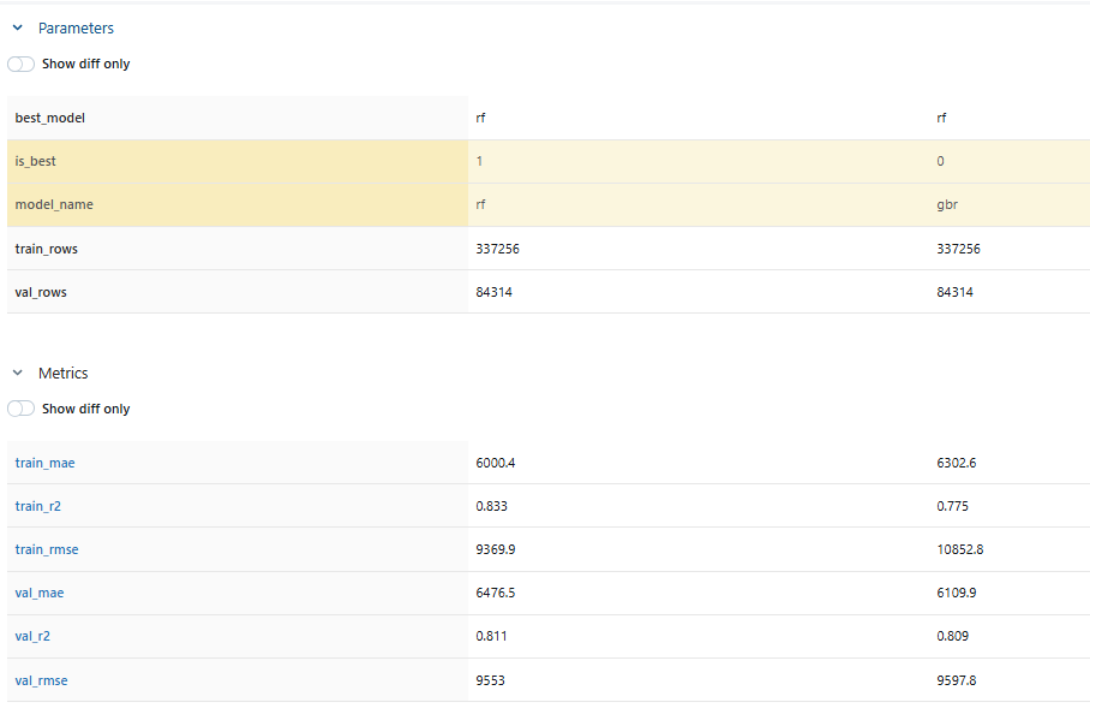

FIGURE 10 – Interface MLflow pour la comparaison des runs

6 Model Registry et gestion du cycle de vie

6.1 Problématique du cycle de vie des modèles

Un modèle traverse plusieurs étapes : expérimentation, validation, mise en production. Le Model Registry formalise ce cycle afin d'assurer traçabilité et gouvernance.

6.2 Model Registry MLflow

MLflow Registry permet :
— d'enregistrer des versions de modèles,
— d'attribuer des statuts (Staging, Production),
— de conserver l'historique complet.

6.3 Promotion automatique des modèles

Un mécanisme de promotion automatique peut être défini selon des seuils de performance (ex. RMSE $<$ seuil). Cela permet d'automatiser le passage en production lorsque les critères sont satisfaits.

7 Déploiement et serving du modèle

7.1 API de prédiction

L'API est construite avec FastAPI. Elle expose un endpoint principal `/predict` recevant un enregistrement ou un batch de données, et retournant les prédictions.

7.2 Conteneurisation avec Docker

Docker garantit un environnement cohérent pour l'exécution de l'API. Un Dockerfile décrit l'installation des dépendances et le lancement du serveur.

```
services:
    api:
        build: .
        ports:
            - "8000:8000"
        environment:
            MODEL_PATH: /app/models/model.pkl
            MODELS_DIR: /app/models
            DEFAULT_MODEL_NAME: model
        volumes:
            - ./models:/app/models
    ui:
        build:
            context: .
            dockerfile: Dockerfile.streamlit
        ports:
            - "8501:8501"
        environment:
            FASTAPI_URL: http://api:8000
            FASTAPI_TIMEOUT: "120"
            FASTAPI_RETRIES: "1"
        depends_on:
            - api
```

8 Interface utilisateur avec Streamlit

8.1 Rôle de Streamlit dans le système

Streamlit sert d'interface utilisateur pour les profils non techniques. Il permet de saisir des données et de visualiser les prédictions.

8.2 Fonctionnalités de l'application

L'application propose :
— Saisie manuelle d'un enregistrement.
— Import CSV pour prédictions batch.
— Affichage des résultats et métriques.

8.3 Intégration avec l'API

Streamlit communique avec FastAPI via des requêtes HTTP. Les prédictions sont affichées en temps réel, avec possibilité de rafraîchir les résultats.

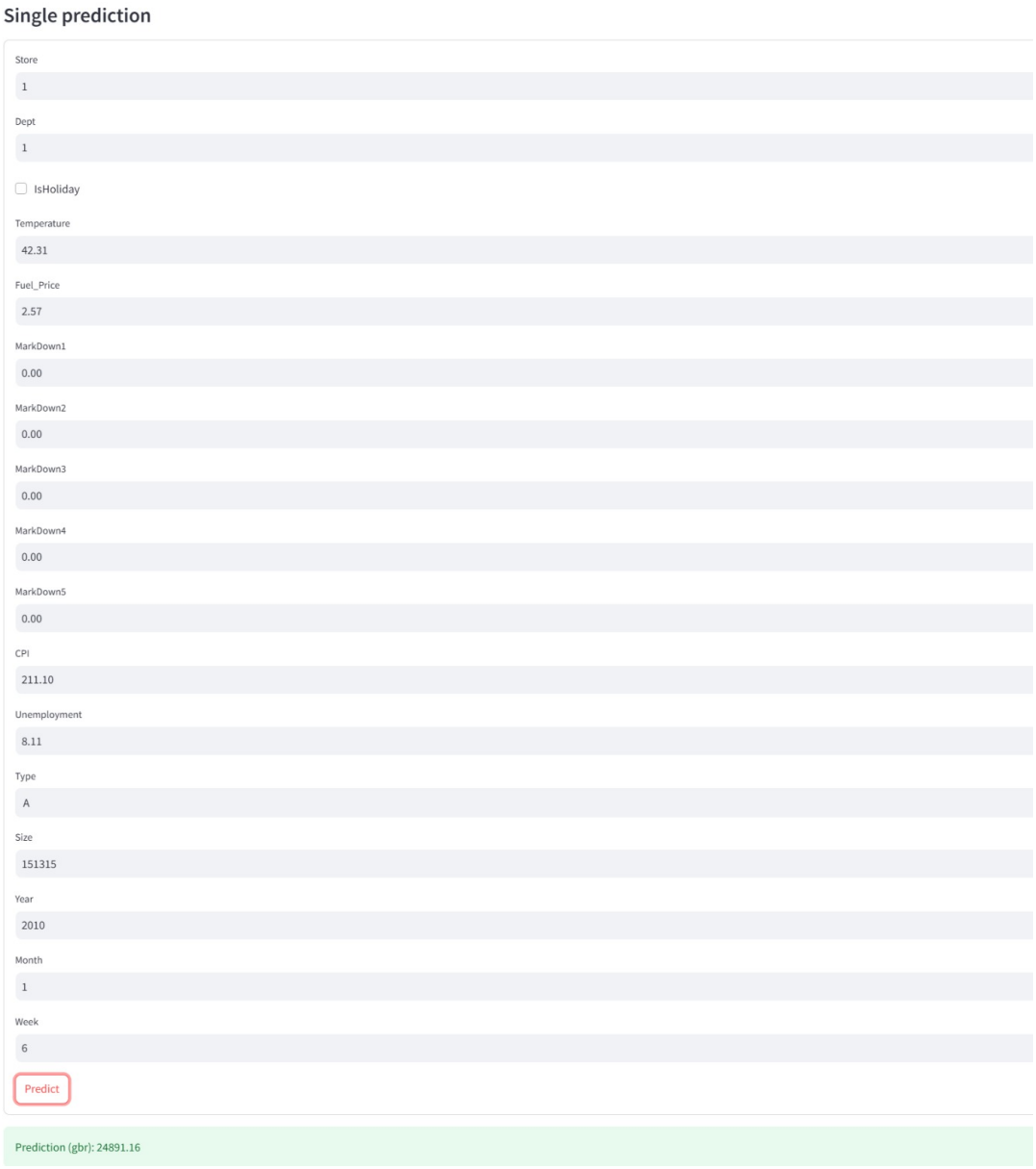

FIGURE 11 – Interface Streamlit pour la prédiction de la demande

9 Conclusion et perspectives

9.1 Bilan du projet

Le projet démontre la mise en place d'un pipeline MLOps complet, de la donnée au déploiement. Les objectifs initiaux sont atteints : intégration DVC/MLflow, traçabilité, serving via API, et interface utilisateur.

9.2 Limites et améliorations futures

Plusieurs axes d'amélioration sont envisageables :
— Monitoring avancé des modèles en production.
— Intégration d'un Feature Store.
— Déploiement cloud scalable (Kubernetes).