Projet Final : Data Engineering & Credit Scoring

Projet Final

24 novembre 2025

Table des matières

1 Objectif **2**

1.1 Votre mission 2

1.2 Synthèse des besoins et recommandations pour l'élaboration de l'outil de credit scoring . 2

2 Étapes du Projet **4**

2.1 Étape 1 : Préparer, nettoyer et enrichir les données 4

2.2 Étape : Traquer les expérimentations avec MLflow 5

2.3 Étape 3 : Modéliser et expérimenter avec plusieurs algorithmes 5

2.4 Étape 4 : Optimiser les hyperparamètres et le seuil métier 7

3 Rendu Final **8**

3.1 Structure du dépot Gihub 8

3.2 Descriptions des dossiers 8

4 Grille d'évaluation du projet **12**

1 Objectif

Vous êtes Data Scientist au sein d'une société financière spécialisée dans les **crédits à la consommation** destinés à des clients ayant peu ou pas d'historique de prêt. L'entreprise souhaite mettre en place un outil complet de **credit scoring** permettant :

- d'évaluer automatiquement la probabilité qu'un client rembourse son crédit ;
- de classer chaque demande en **crédit accordé** ou **crédit refusé** ;
- de s'appuyer sur des données variées (comportementales, financières externes, etc.).

Voici les données dont vous aurez besoin pour réaliser l'algorithme de classification. Pour plus de simplicité, vous pouvez les télécharger à cette adresse `https://www.kaggle.com/c/home-credit-default-risk/data`. Vous aurez besoin de joindre les différentes tables entre elles.

Votre mission consiste à développer un algorithme de classification, analyser les facteurs explicatifs du modèle et mettre en œuvre une démarche MLOps de bout en bout.

1.1 Votre mission

Construire et optimiser un modèle de scoring qui donnera une prédiction sur la probabilité de faillite d'un client de façon automatique.

Analyser les features qui contribuent le plus au modèle, d'une manière générale (feature importance globale) et au niveau d'un client (feature importance locale), afin, dans un souci de transparence, de permettre à un chargé d'études de mieux comprendre le score attribué par le modèle.

Mettre en œuvre une approche globale MLOps de bout en bout, du tracking des expérimentations à la pré-production du modèle.

Votre manager, vous incite à sélectionner un ou des kernels Kaggle pour vous faciliter l'analyse exploratoire, la préparation des données et le feature engineering nécessaires à l'élaboration du modèle de scoring.

Si vous le faites, vous devez analyser ce ou ces kernels et les adapter pour vous assurer qu'ils répondent aux besoins de votre mission. C'est optionnel, mais nous vous encourageons à le faire afin de vous permettre de vous focaliser sur l'élaboration du modèle, son optimisation et sa compréhension.

1.2 Synthèse des besoins et recommandations pour l'élaboration de l'outil de credit scoring

L'objectif est de développer un outil de *credit scoring* accompagné d'une démarche MLOps complète permettant d'automatiser et d'industrialiser le cycle de vie des modèles. La plateforme devra s'appuyer sur des outils open source (liste fournie).

Composante MLOps attendue

La mise en œuvre doit inclure au minimum :

- le tracking des expérimentations avec MLflow dans les notebooks d'entraînement ;
- l'utilisation de l'interface *MLflow UI* pour visualiser les runs ;
- le stockage centralisé des modèles dans un *model registry* MLflow ;
- le test du *serving* MLflow.

Enjeux métiers à intégrer

Deux éléments majeurs doivent être pris en compte dans la construction du modèle :

- le déséquilibre entre bons et mauvais clients dans le jeu de données ;
- l'asymétrie du coût métier : un faux négatif (FN) coûte environ dix fois plus qu'un faux positif (FP).

Score métier et optimisation du seuil

Un score métier doit être défini afin de comparer les modèles en minimisant le coût des erreurs FN/FP. Cette approche nécessite d'optimiser le seuil de décision plutôt que d'utiliser le seuil standard de 0.5.

Des métriques techniques comme l'AUC et l'accuracy doivent être conservées à titre de contrôle.

Méthodologie de modélisation

L'élaboration du modèle doit inclure :

- une validation croisée ;
- une optimisation des hyperparamètres (via `GridSearchCV` ou équivalent).

Une vigilance particulière doit être apportée au risque d'overfitting, notamment si les performances dépassent trop nettement les résultats des meilleurs modèles Kaggle (AUC > 0.82).

Utilisation de MLflow

MLflow est utilisé pour :

- le suivi et le tracking des expériences ;
- la visualisation des résultats (*MLflow UI*) ;
- la gestion centralisée des modèles via un *model registry* ;
- le test de la mise en production via le *serving*.

2 Étapes du Projet

2.1 Étape 1 : Préparer, nettoyer et enrichir les données

Vous allez découvrir et préparer les données nécessaires à la construction de votre modèle de scoring. Cela inclut le nettoyage, la fusion des différentes sources, l'encodage des variables et la création de nouvelles features pertinentes. L'objectif est de constituer un dataset propre et enrichi, prêt pour l'entraînement. Vous devrez également analyser la qualité de vos variables et les déséquilibres dans les classes.

Prérequis

- Avoir exploré les données brutes fournies.
- Avoir vérifié les formats et les valeurs manquantes.
- Avoir identifié les colonnes clés pour les jointures.
- Avoir pris en compte les enjeux métiers (par exemple : déséquilibre des classes).

Résultat attendu

Un jeu de données propre, fusionné et enrichi, prêt à être utilisé pour l'entraînement.

Recommandations

- Charger chaque fichier séparément et inspecter ses colonnes.
- Utiliser `pandas` pour fusionner les jeux de données.
- Visualiser la distribution des classes cibles.
- Créer de nouvelles features à partir des variables existantes si nécessaire.
- Éviter de supprimer trop rapidement les colonnes avec des valeurs manquantes : explorer les possibilités d'imputation.

Points de vigilance

- Oublier de vérifier les doublons.
- Supprimer des colonnes sans analyser leur importance métier.
- Imputer sans documenter ni justifier.
- Fusionner sans gérer les duplications ou pertes de lignes.
- Encoder sans tenir compte du type de modèle prévu (ordinal vs nominal).

Outils

- `pandas`
- `matplotlib` et `seaborn` pour la visualisation
- `scikit-learn` pour le preprocessing
- `missingno` pour visualiser les valeurs manquantes

2.2 Étape : Traquer les expérimentations avec MLflow

Vous allez tracer vos expériences de modélisation avec MLflow : métriques, hyperparamètres, versions de modèles, etc. Vous utiliserez l'interface web pour visualiser vos runs et comparer les modèles.

Prérequis

— Avoir installé MLflow.
— Avoir configuré votre projet localement.

Résultat attendu

Des runs visibles dans l'UI MLflow avec les paramètres testés et les scores obtenus.

Recommandations

— Commencer par intégrer `mlflow.start_run()` dans vos notebooks.
— Logger les métriques et les paramètres principaux.
— Utiliser `mlflow.autolog()` si vous utilisez des modèles compatibles.
— Activer l'interface UI avec `mlflow ui` pour visualiser les résultats.

Points de vigilance

— Lancer MLflow sans environnement isolé peut créer des conflits de versions. Utiliser un environnement virtuel.
— Oublier d'annoter les expériences (tags, noms, commentaires) complique l'analyse dans l'interface MLflow.
— Ne pas versionner les modèles enregistrés empêche de reproduire les résultats et de gérer leur cycle de vie.
— Sauvegarder des fichiers inutiles ou trop volumineux dans MLflow encombre le système et ralentit l'interface.

Outils

— `MLflow`

2.3 Étape 3 : Modéliser et expérimenter avec plusieurs algorithmes

Vous allez entraîner différents modèles de classification et comparer leurs performances sur des métriques métiers et classiques. L'objectif est de tester plusieurs familles de modèles (forêts, boosting, MLP, etc.) et de construire une première version de votre pipeline d'apprentissage. Vous devez aussi intégrer une validation croisée pour évaluer leur robustesse.

Prérequis

- Avoir préparé un dataset propre et prêt à l'entraînement.
- Avoir compris la nature déséquilibrée du jeu de données.
- Avoir identifié les variables cibles et explicatives.
- Avoir installé les bibliothèques de machine learning nécessaires.
- Avoir paramétré MLflow.

Résultat attendu

Un ou plusieurs modèles entraînés, avec validation croisée et premières métriques d'évaluation.

Recommandations

- Commencer par tester des modèles simples (*Logistic Regression*, *Random Forest*).
- Comparer ensuite avec des modèles plus puissants (*XGBoost*, *LightGBM*, *MLP*).
- Utiliser `StratifiedKFold` pour conserver la distribution des classes et garantir une évaluation robuste.
- Entraîner les modèles dans des notebooks clairs et documentés.
- Stocker les scores et les hyperparamètres testés.

Points de vigilance

- **Ne pas tester sans validation croisée.** Une évaluation basée uniquement sur un split *train/test* unique peut produire des résultats très variables selon le hasard du découpage, et conduire à des conclusions erronées sur la performance réelle d'un modèle.
- **Ne pas comparer les modèles avec des métriques inadaptées.** Privilégier des métriques pertinentes, telles que :
  - AUC-ROC,
  - Recall sur la classe minoritaire,
  - F1-score,
  - Coût métier personnalisé ($FN \gg FP$).
- **Ne pas oublier la stratification.** Sans stratification, certains algorithmes peuvent se biaisent vers la classe majoritaire si le dataset contient beaucoup plus de bons que de mauvais clients.
- **Ne pas ignorer le déséquilibre des classes.** Utiliser un `class_weight` adapté ou du sur-échantillonnage (SMOTE, etc.) pour éviter de biaiser l'apprentissage.

Outils

- `scikit-learn`
- `XGBoost`
- `LightGBM`

2.4 Étape 4 : Optimiser les hyperparamètres et le seuil métier

Vous allez optimiser les hyperparamètres des modèles pour maximiser leurs performances selon des critères métier. Vous définirez également un seuil de décision optimal basé sur le coût des erreurs. L'objectif est de minimiser le coût métier total (avec un poids plus fort sur les faux négatifs que sur les faux positifs).

Prérequis

- Avoir entraîné plusieurs modèles.
- Avoir comparé leurs performances de base.
- Avoir compris la notion de coût d'erreur.
- Avoir défini une fonction de coût métier.

Résultat attendu

Un modèle avec hyperparamètres optimisés et un seuil métier ajusté.

Recommandations

- Utiliser `GridSearchCV` ou `Optuna` pour l'optimisation.
- Définir une fonction de coût pondérant les erreurs FN et FP.
- Tester différents seuils de classification (par exemple de 0.1 à 0.9).
- Tracer la courbe *coût vs. seuil* pour identifier la meilleure décision.

Points de vigilance

- Garder le seuil par défaut (0.5) sans justification. Ce seuil ne reflète pas nécessairement les enjeux métiers : il doit être optimisé selon le ratio FN/FP.
- Oublier de tracer le score métier en fonction du seuil : cela empêche d'identifier la meilleure décision.
- Optimiser uniquement sur l'AUC ou l'accuracy : ces métriques ne reflètent pas toujours les pertes métier.
- Oublier d'adapter les métriques aux besoins business.
- Choisir un modèle sans tester sa robustesse.

Outils

- `scikit-learn` (`GridSearchCV`)
- `Optuna`

Résultat attendu : un modèle final optimisé et justifié.

3 Rendu Final

Votre rendu final est un dépot github avec la structure suivante. Voici la structure attendue du projet :

3.1 Structure du dépot Gihub

```
credit-scoring/
|
|-- README.md
|-- requirements.txt
|-- Dockerfile
|-- .gitignore
|
|-- notebooks/
|   |-- 01_data_preparation.ipynb
|   |-- 02_model_training.ipynb
|   |-- 03_explainability.ipynb
|   `-- 04_mlflow_serving_test.ipynb
|
|-- src/
|   |-- data_prep.py          # Fonctions de nettoyage, encodage, jointures
|   |-- model_utils.py        # Fonctions d'entraînement, split, sauvegarde modèle
|   |-- metrics.py            # Fonctions de calcul des métriques (AUC, coût, recall)
|   `-- explainability.py     # Fonctions pour SHAP global/local
|
|-- model/
|   |-- MLmodel
|   |-- conda.yaml
|   `-- model.pkl
|
|-- reports/
|   |-- rapport_credit_scoring.pdf
|   `-- figures/
|       |-- shap_global.png
|       |-- shap_local.png
|       `-- courbe_cout_vs_seuil.png
|
`-- mlruns/                   # (facultatif) Runs MLflow
```

3.2 Descriptions des dossiers

README.md Fichier de documentation *obligatoire*. Il doit contenir :

— un résumé du projet et de l'objectif métier (scoring crédit, coût FN/FP) ;

- les commandes pour lancer le projet :
  - construction de l'image Docker ;
  - lancement du conteneur (serveur de modèle) ;
- la commande `curl` complète pour tester l'API d'inférence (`/invocations`) ;
- la description du seuil métier choisi (ex. : seuil = 0,37, décision acceptée/refusée) ;
- un rappel de la structure du dépôt.

`requirements.txt` Liste des dépendances Python (avec versions si nécessaire). Ce fichier doit permettre d'installer l'environnement d'exécution avec `pip install -r requirements.txt`.

`Dockerfile` Fichier de configuration Docker permettant de :

- installer les dépendances listées dans `requirements.txt` ;
- copier le modèle exporté depuis MLflow (dossier `model/`) ;
- exposer le port 1234 ;
- démarrer le serving du modèle, par exemple via :

Le conteneur Docker ne doit pas réentraîner le modèle ni lancer le tracking MLflow : il sert uniquement le modèle final.

`.gitignore` Fichier permettant d'exclure du dépôt les fichiers inutiles ou volumineux, par exemple :

- environnements virtuels : `venv/` ;
- fichiers temporaires Python : `__pycache__/`, `*.pyc` ;
- checkpoints de notebooks : `.ipynb_checkpoints/` ;
- données brutes : `data/`, `*.csv`, `*.parquet` ;
- dossiers `mlruns/` locaux (tracking MLflow).

`notebooks/` : démarche et expérimentation

`01_data_preparation.ipynb` Notebook dédié à la préparation des données :

- chargement et fusion des différentes sources ;
- nettoyage (doublons, valeurs manquantes, outliers) ;
- encodage des variables catégorielles ;
- séparation train/test (stratifiée si possible) ;
- analyse du déséquilibre des classes.

`02_model_training.ipynb` Notebook d'entraînement des modèles. Il doit contenir :

- la définition de plusieurs modèles (baseline + modèles plus avancés) ;
- la gestion du déséquilibre (pondération de classes, sur-échantillonnage, etc.) ;
- la validation croisée et le calcul de métriques (AUC, recall, coût métier, etc.) ;
- le **tracking MLflow obligatoire** :
  - `mlflow.log_param` pour les hyperparamètres ;
  - `mlflow.log_metric` pour les métriques ;
  - `mlflow.log_artifact` pour les graphiques ;
  - enregistrement du modèle dans le registry MLflow (par exemple `credit_scoring_model`).
- l'export du modèle final vers le dossier `model/`.

`03_explainability.ipynb` Notebook consacré à l'explicabilité du modèle :

- calcul et visualisation de l'importance globale des variables (SHAP global) ;
- explication locale pour un ou plusieurs clients (SHAP local) ;
- export des figures vers `reports/figures/`.

04_mlflow_serving_test.ipynb Notebook de test du modèle servi :
- appel de l'API de prédiction (via `requests` ou `curl`) ;
- vérification de la cohérence des prédictions sur un échantillon du jeu de test ;
- calcul d'une métrique simple (AUC ou coût métier) à partir des réponses de l'API.

src/ : code Python réutilisable

Le dossier `src/` contient les scripts `.py` factorisant le code réutilisable. Les notebooks doivent importer ces fonctions.

data_prep.py Fonctions de préparation des données (chargement, jointure, nettoyage, encodage).

model_utils.py Fonctions d'entraînement et de validation des modèles (split train/test, entraînement, sauvegarde).

metrics.py Fonctions de calcul des métriques :
- métriques techniques : AUC, précision, rappel, F1 ;
- métrique métier : coût pondéré des faux positifs et faux négatifs.

explainability.py Fonctions dédiées à l'explicabilité (calcul SHAP, graphiques globaux et locaux).

model/ : modèle exporté

Le dossier `model/` contient le modèle final exporté depuis MLflow (artefact) :
- `MLmodel` : métadonnées du modèle (format MLflow) ;
- `conda.yaml` : description de l'environnement utilisé lors de l'entraînement ;
- `model.pkl` : fichier binaire du modèle.

Ce dossier est utilisé par le `Dockerfile` pour servir le modèle via `mlflow models serve`.

reports/ : rapport et figures

rapport_credit_scoring.pdf Rapport final (2 à 3 pages maximum) présentant :
- la démarche de préparation des données et de modélisation ;
- les résultats principaux (AUC, seuil optimal, coût métier) ;
- une interprétation des variables importantes ;
- une capture d'écran MLflow montrant les runs et le modèle choisi.

figures/ Dossier contenant les figures utilisées dans le rapport :
- importance globale des variables ;
- `courbe_cout_vs_seuil.png` : optimisation du seuil métier.

mlruns/ : tracking MLflow (facultatif dans Git)

Le dossier `mlruns/` contient les logs locaux MLflow (runs, métriques, artefacts). Il n'est pas nécessaire de le versionner dans Git, mais :
- le tracking MLflow est **obligatoire** dans le notebook d'entraînement ;

— une capture d'écran de l'interface MLflow (comparaison de plusieurs runs, modèle final dans le registry) doit apparaître dans le rapport.

4 Grille d'évaluation du projet

Décision finale lors d'un oral de 5 minutes

1. Conditions matérielles et techniques

- L'étudiant se présente avec son propre ordinateur, prêt à l'emploi.
- Le conteneur Docker du modèle doit être lancé et opérationnel.
- Le serveur MLflow (tracking + UI) doit être démarré et accessible.
- Le serveur de prédiction (serving du modèle via MLflow) doit également être actif.

2. Déroulement de l'oral

- L'étudiant réalise une démonstration du modèle en serving via Docker sur un échantillon du jeu de test (appel API, vérification de la réponse).
- Le jury évalue la maîtrise technique : compréhension du pipeline, cohérence des résultats et capacité à expliquer les choix.

3. Décision

- **Oral validé** : la note obtenue sur le dépôt GitHub (sur 20) est confirmée.
- **Oral non validé** : la note peut être réduite ou invalidée selon l'appréciation du jury.

TABLE 1 – Grille d'évaluation du dépôt GitHub (Total : 20 points)

Catégorie	**Critères évalués**	**Points**
Structure du dépôt	Arborescence conforme, organisation claire, fichiers obligatoires présents.	2
README.md	Contexte métier, installation, commandes Docker, test API (curl), seuil métier, structure du projet.	3
Notebooks	4 notebooks complets : data prep, entraînement, explicabilité, test du serving MLflow. Documentation claire, reproductibilité.	4
Code Python dans `src/`	Modularité, fonctions propres et réutilisables, séparation data/model/metrics.	3
Tracking MLflow	Runs complets (paramètres, métriques, artefacts), modèle enregistré dans le registry, versionnage.	3
Modèle final + Docker	Présence de MLmodel, conda.yaml, model.pkl ; Dockerfile fonctionnel servant le modèle via MLflow.	3
Rapport PDF	Synthèse claire (2–3 pages), résultats, interprétabilité, figures cohérentes.	2