Rapport de Projet : Ingénierie Logicielle Appliquée (Titanic)

Groupe : 6

Membres de l'équipe : Rayan RAMI Ismaël GAHLOUZI Aziz DJERBI Asaad SAADI

Dépôt GitHub : https://github.com/RayanData/Titanic--Project-Groupe6

Date : 8 Février 2026

---

1. Introduction et Objectifs

Ce projet a pour but d'appliquer les bonnes pratiques d'ingénierie logicielle à un projet de Data Science existant (prédiction de survie sur le Titanic). Nous sommes partis d'un notebook monolithique pour construire une application modulaire, testée et automatisée, prête pour le déploiement.

Nos objectifs principaux étaient :

- **Refactoriser** le code pour le rendre maintenable et réutilisable.
- Assurer la **qualité du code** (normes PEP 8) et la fiabilité (tests unitaires).
- Mettre en place une chaîne d'intégration continue (**CI/CD**) avec GitHub Actions.
- Garantir la portabilité via la **conteneurisation** (Docker).

---

2. Architecture du Projet et Refactoring

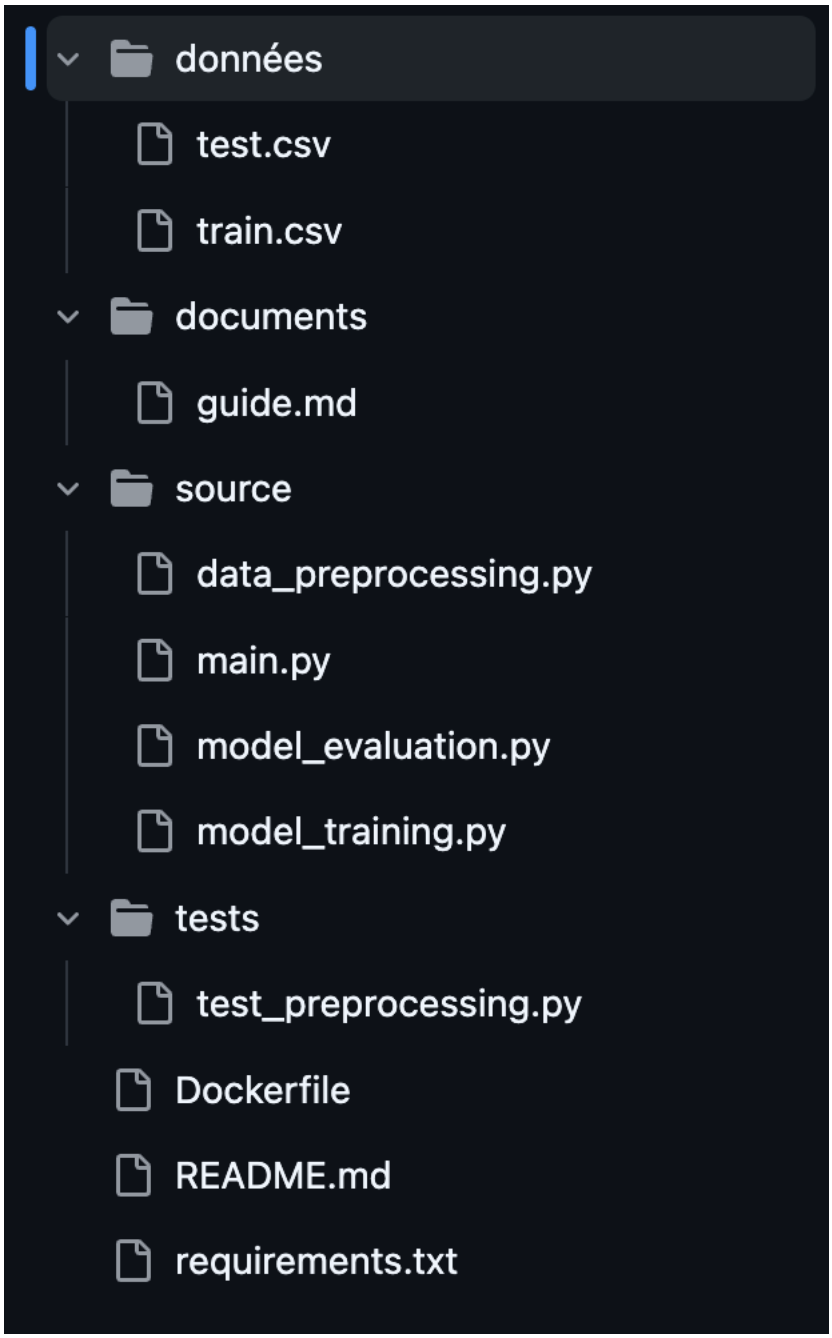
Nous avons transformé le notebook initial en une structure de projet standardisée. Le code source a été divisé en modules distincts situés dans le dossier `src/`, chacun ayant une responsabilité unique.

Structure des fichiers

Le projet suit l'arborescence suivante :

- `src/data_preprocessing.py` : Contient les fonctions de nettoyage et de préparation des données (gestion des valeurs manquantes, encodage).
- `src/model_training.py` : Gère l'entraînement du modèle (Random Forest) et sa sauvegarde au format `.pkl`.
- `src/model_evaluation.py` : Calcule et affiche les métriques de performance (Précision, F1-Score).
- `src/main.py` : Point d'entrée unique qui orchestre tout le pipeline.

Capture d'écran de l'architecture finale :



---

3. Qualité du Code et Tests Unitaires

Respect des normes (PEP 8)

Nous avons veillé à utiliser des noms de variables explicites et à respecter la convention PEP 8. Un fichier `requirements.txt` a été créé pour figer les versions des librairies et assurer la reproductibilité de l'environnement.

Tests Unitaires (Pytest)

Pour garantir la robustesse du code, nous avons implémenté des tests unitaires dans le dossier `tests/`. Nous avons utilisé le framework **pytest**.

- **Fichier de test :** `tests/test_preprocessing.py`
- **Fonction testée :** Vérification que le nettoyage des données renvoie bien un DataFrame non vide et contient les colonnes cibles.

---

4. Automatisation et CI/CD (GitHub Actions)

Nous avons mis en place un pipeline d'Intégration Continue (CI) hébergé sur GitHub via **GitHub Actions**.

À chaque `push` ou `pull_request` sur la branche principale, le workflow défini dans `.github/workflows/main.yml` s'exécute automatiquement. Il effectue les étapes suivantes :

1. Installation de l'environnement Python.
2. Installation des dépendances (`pip install -r requirements.txt`).
3. Vérification de la syntaxe (Linting avec Flake8).
4. Exécution des tests unitaires (Pytest).

Cela garantit qu'aucun code défectueux ne peut être intégré au projet.

---

5. Documentation et Collaboration

- **Documentation :** Un dossier `docs/` a été créé contenant un guide utilisateur (`guide.md`) expliquant l'installation et l'exécution du projet.
- **Collaboration :** Le projet est hébergé sur un dépôt GitHub public. Nous avons utilisé Git pour le versioning, avec des messages de commit explicites pour suivre l'évolution du développement.

---

6. Bonus : Conteneurisation (Docker)

Afin de valider les points bonus, nous avons ajouté un `Dockerfile` à la racine du projet. Ce fichier permet de construire une image Docker du projet, rendant l'application exécutable sur n'importe quelle machine sans soucis.

Conclusion

Ce projet nous a permis de passer d'une approche "Notebook exploratoire" à une approche "Ingénierie Logicielle industrielle". Le pipeline de données est maintenant robuste, testé automatiquement et documenté, prêt pour une mise en production.