# PROJET TITANIC

## Prédiction de Survie avec Machine Learning

Ingénierie Logicielle Appliquée

BUT VCOD - IUT Paris Cité

Année 2025-2026

**Réalisé par :**

BHUIYAN Rakib

Riade EL ATTAR

| | |
|---|---|
| **Repository GitHub:** | https://github.com/B-Rakib/titanic-survival-ml |
| **Branche principale:** | main |
| **Date de rendu:** | 11 Février 2026 |
| **Contact:** | bhu.rakib05@gmail.com |

# Table des Matières

# 1. Introduction et Contexte

Le naufrage du Titanic en 1912 demeure l'une des catastrophes maritimes les plus marquantes de l'histoire. Dans le cadre de notre formation en ingénierie logicielle, nous avons entrepris de développer une solution complète de machine learning visant à prédire la survie des passagers en analysant leurs caractéristiques démographiques et socio-économiques. Ce projet représente bien plus qu'un simple exercice de modélisation : il constitue une occasion d'appliquer rigoureusement l'ensemble des principes et bonnes pratiques de l'ingénierie logicielle moderne dans un contexte concret et mesurable.

## Objectifs Pédagogiques

L'objectif principal consistait à transformer un notebook Jupyter initial, fonctionnel mais monolithique, en une application Python professionnelle, modulaire et maintenable, respectant les standards industriels en matière de structuration du code, de tests automatisés, d'intégration continue et de documentation. Le défi ne résidait donc pas uniquement dans la création d'un modèle prédictif performant, mais plutôt dans la mise en œuvre d'une architecture logicielle robuste, facilement compréhensible, testable, déployable et maintenable par une équipe de développeurs.

Ce projet s'inscrit dans une démarche d'apprentissage complète visant à maîtriser l'ensemble du cycle de développement logiciel. Nous devions refactoriser le code existant en modules Python réutilisables, implémenter des tests unitaires avec pytest, mettre en place un pipeline CI/CD avec GitHub Actions, containeriser l'application avec Docker, et produire une documentation exhaustive accompagnant l'ensemble du projet.

# 2. Méthodologie de Développement

Notre équipe a adopté une méthodologie de développement structurée et progressive, privilégiant la qualité du code et la collaboration efficace. Le travail s'est organisé autour de plusieurs phases distinctes, chacune apportant sa contribution à l'édifice final du projet.

## Approche de Développement

Nous avons commencé par analyser en profondeur le notebook Jupyter fourni afin de comprendre la logique métier et d'identifier les différentes étapes du pipeline de machine learning. Cette phase d'analyse nous a permis de concevoir une architecture modulaire claire, où chaque composant a une responsabilité bien définie. Le développement s'est ensuite déroulé de manière itérative, en commençant par le module de prétraitement des données, puis en progressant vers l'entraînement du modèle et enfin l'évaluation.

L'utilisation de Git et GitHub a été au cœur de notre processus de développement. Chaque fonctionnalité significative a fait l'objet d'un commit avec un message descriptif suivant les conventions établies. Cette discipline nous a permis de maintenir un historique clair des modifications et de faciliter la traçabilité du projet. Le repository GitHub centralise l'ensemble du code source, des tests, de la documentation et des configurations CI/CD, garantissant ainsi une vision unifiée et accessible du projet.

# 3. Architecture Technique du Projet

L'architecture du projet repose sur une séparation claire des responsabilités, inspirée des principes SOLID et du modèle en couches. Chaque module possède une fonction spécifique et peut être testé, maintenu et amélioré indépendamment des autres composants.

## Structure Modulaire

| Module | Responsabilité | Fichier |
|---|---|---|
| Prétraitement | Nettoyage et transformation | src/data_preprocessing.py |
| Entraînement | Modélisation ML | src/model_training.py |
| Évaluation | Métriques et soumission | src/model_evaluation.py |
| Orchestration | Pipeline complet | main.py |

Le module de prétraitement gère l'ensemble du pipeline de transformation des données : chargement des fichiers CSV, traitement des valeurs manquantes, création de nouvelles features (FamilySize, IsAlone, Title), et encodage des variables catégorielles. Le module d'entraînement encapsule la logique de création et d'entraînement du modèle de régression logistique, avec validation croisée et sauvegarde du modèle. Le module d'évaluation fournit les outils de mesure de performance et de génération du fichier de soumission Kaggle. Enfin, le script principal orchestre l'exécution séquentielle de ces trois étapes.

# 4. Implémentation et Technologies

## Choix Technologiques

| Catégorie | Technologie | Justification |
|---|---|---|
| Langage | Python 3.11 | Ecosystème data science riche |
| ML | scikit-learn | Bibliothèque standard et robuste |
| Data | pandas, numpy | Manipulation efficace des données |
| Tests | pytest | Framework de test simple et puissant |
| CI/CD | GitHub Actions | Intégration native avec GitHub |
| Container | Docker | Portabilité et reproductibilité |

## Modèle de Machine Learning

Nous avons opté pour une régression logistique plutôt qu'un modèle plus complexe comme Random Forest. Ce choix délibéré repose sur plusieurs considérations. La régression logistique offre une excellente interprétabilité, permettant de comprendre facilement l'impact de chaque variable sur la prédiction. Elle présente également un temps d'entraînement très rapide et nécessite peu de ressources computationnelles. Les features utilisées incluent la classe du billet (Pclass), le sexe (Sex), l'âge (Age), le tarif (Fare), le port d'embarquement (Embarked), ainsi que des variables dérivées comme la taille de la famille (FamilySize), le statut de solitude (IsAlone), la présence de cabine (Has_Cabin) et le titre extrait du nom (Title).

# 5. Assurance Qualité et Tests

L'assurance qualité constitue un pilier fondamental de notre démarche d'ingénierie. Nous avons développé une suite complète de tests unitaires couvrant l'ensemble des fonctionnalités critiques de l'application.

## Stratégie de Tests

| Module | Nombre de Tests | Couverture |
|---|---|---|
| data_preprocessing | 11 tests | Chargement, valeurs manquantes, features encodage |
| model_training | 5 tests | Création, entraînement, sauvegarde, prédic ion |
| model_evaluation | 3 tests | Métriques, fichier de soumission |
| TOTAL | 19 tests | 100% de réussite |

Notre suite de tests valide le bon fonctionnement de chaque étape du pipeline. Les tests du module de prétraitement vérifient le chargement correct des données, le traitement des valeurs manquantes, la création des nouvelles features et l'encodage des variables catégorielles. Les tests du module d'entraînement s'assurent que le modèle est correctement instancié, entraîné et sauvegardé. Les tests du module d'évaluation valident le calcul des métriques et la génération du fichier de soumission. L'exécution de ces tests est automatisée via le pipeline CI/CD, garantissant qu'aucune régression n'est introduite lors des modifications du code.

# 6. Intégration Continue et Déploiement

L'intégration continue transforme radicalement la façon dont les équipes développent et livrent des logiciels. En automatisant les tests et la vérification de la qualité du code, le CI/CD permet de détecter les problèmes très tôt dans le cycle de développement, lorsqu'ils sont encore faciles et peu coûteux à corriger.

## Pipeline GitHub Actions

Nous avons mis en place un pipeline complet avec GitHub Actions qui s'exécute automatiquement à chaque push sur la branche principale. Le pipeline commence par la création d'un environnement Python 3.11 isolé dans une machine virtuelle Ubuntu. Il installe ensuite toutes les dépendances listées dans le fichier requirements.txt. Une fois l'environnement prêt, le pipeline exécute l'intégralité de notre suite de tests unitaires avec pytest. Si un seul test échoue, le pipeline entier est marqué comme failed et l'équipe est immédiatement notifiée via le badge dans le README.

## Containerisation Docker

La containerisation avec Docker représente une avancée majeure pour le déploiement d'applications. Notre Dockerfile définit précisément comment construire l'image de l'application, en partant d'une image Python 3.11 minimale, en installant les dépendances, et en copiant le code source. Cette approche garantit que l'application s'exécutera de manière identique sur n'importe quel système supportant Docker, éliminant ainsi les problèmes classiques de compatibilité entre environnements de développement et de production.

# 7. Résultats et Performances

Au-delà de la qualité de l'architecture et de l'implémentation, un projet de machine learning doit produire des résultats précis et interprétables. Notre modèle de régression logistique a atteint des performances satisfaisantes tout en conservant une excellente lisibilité.

## Métriques de Performance

| Métrique | Valeur |
|---|---|
| Accuracy (validation croisée) | ~80% |
| Survivants prédits | 148/418 (35.4%) |
| Features utilisées | 8 + titres encodés |

L'analyse exploratoire des données révèle que le sexe des passagers constitue le facteur le plus déterminant pour la survie, reflétant directement l'application du protocole "femmes et enfants d'abord" lors de l'évacuation du navire. La classe du billet apparaît également comme un facteur important, les passagers de première classe ayant bénéficié d'un accès privilégié aux canots de sauvetage. Les variables familiales montrent des corrélations intéressantes : voyager avec quelques membres de sa famille semble améliorer les chances de survie, tandis que les très grandes familles ont rencontré plus de difficultés à coordonner leur évacuation dans le chaos.

# 8. Répartition des Tâches

La réussite du projet repose sur une collaboration efficace et une répartition claire des responsabilités entre les membres de l'équipe. Chaque membre a apporté ses compétences spécifiques tout en contribuant à la vision globale du projet.

| Membre | Contributions Principales |
| --- | --- |
| BHUIYAN Rakib | Développement complet du projet : architecture modulaire, impl mentation des mo |
| Riade EL ATTAR | Rédaction du README.md complet avec instructions d'installati n et utilisation, |

La collaboration s'est articulée autour de l'utilisation intensive de Git et GitHub pour le versionnement du code. Les commits réguliers avec des messages explicites ont permis de maintenir un historique clair des modifications. Le pipeline CI/CD automatisé a servi de garde-fou, garantissant que chaque modification passait avec succès l'ensemble des tests avant d'être intégrée. Cette discipline collective a considérablement facilité la coordination du travail et la qualité finale du projet.

# 9. Conclusion et Perspectives

Ce projet nous a permis d'appliquer concrètement les principes d'ingénierie logicielle à un cas réel de machine learning. En transformant un notebook Jupyter initial en une application Python professionnelle, nous avons acquis une compréhension approfondie de l'importance de la modularité, des tests automatisés, de l'intégration continue et de la containerisation. Le projet a atteint l'ensemble de ses objectifs : une architecture modulaire claire, une suite de tests complète avec 19 tests validés, un pipeline CI/CD fonctionnel, une containerisation Docker opérationnelle, et une documentation exhaustive.

## Compétences Acquises

Au-delà des aspects techniques, ce projet nous a enseigné l'importance d'une communication claire et régulière au sein d'une équipe. La gestion de versions avec Git, initialement intimidante, est devenue naturelle au fil du projet et nous comprenons maintenant pourquoi elle est universellement adoptée dans l'industrie. L'expérience du CI/CD nous a montré comment l'automatisation transforme le processus de développement en détectant immédiatement les problèmes et en maintenant un standard de qualité constant.

## Perspectives d'Amélioration

• Enrichir le modèle avec l'optimisation des hyperparamètres via GridSearchCV

• Expérimenter avec d'autres algorithmes (XGBoost, Random Forest) et comparer les performances

• Implémenter une API REST avec FastAPI pour servir le modèle en production

• Développer une interface web interactive pour les prédictions

• Étendre la suite de tests pour atteindre une couverture de code de 100%

Ce projet Titanic a été une expérience formatrice qui a considérablement élargi notre compréhension de ce qu'implique réellement le développement logiciel professionnel. Les compétences techniques acquises constituent un socle solide pour nos futures carrières, mais peut-être plus important encore, nous avons développé une méthodologie de travail structurée et une capacité à collaborer efficacement qui seront précieuses bien au-delà de ce projet spécifique.

# Annexes

## Structure Complète du Repository

```
titanic-survival-ml/
■■■ src/ (Code source modulaire)
■  ■■■ data_preprocessing.py (Prétraitement des données)
■  ■■■ model_training.py (Entraînement du modèle)
■  ■■■ model_evaluation.py (Évaluation et soumission)
■■■ tests/ (Tests unitaires - 19 tests)
■  ■■■ test_data_preprocessing.py
■  ■■■ test_model_training.py
■  ■■■ test_model_evaluation.py
■■■ data/
■  ■■■ raw/ (Données Titanic brutes)
■  ■■■ processed/ (Données traitées et soumission)
■■■ models/ (Modèles ML sauvegardés)
■■■ notebooks/ (Notebook Jupyter de départ)
■■■ .github/workflows/ (Pipeline CI/CD)
■  ■■■ ci.yml
■■■ Dockerfile (Containerisation)
■■■ .dockerignore
■■■ .flake8 (Configuration linting)
■■■ main.py (Script principal)
■■■ requirements.txt (Dépendances Python)
■■■ README.md (Documentation complète)
■■■ .gitignore
```

## Informations de Restitution

**Repository GitHub :** https://github.com/B-Rakib/titanic-survival-ml
**Branche principale :** main
**Badge CI/CD :** Visible dans le README.md
**Date de rendu :** 11 Février 2026

**Projet réalisé par :**
BHUIYAN Rakib - Riade EL ATTAR

**Dans le cadre du cours d'Ingénierie Logicielle**
BUT VCOD - IUT Paris Cité
Année universitaire 2025-2026