Lugat Gaspard
Tivrier Baptiste
Chen Patrick
Sebag Dan

# PROJET D'INGENIERIE LOGICIELLE

# Table des matières

# 1. Introduction :

Ce projet vise à appliquer les bonnes pratiques d'ingénierie logicielle à un projet existant sous forme de notebook Python. Le projet choisi est Titanic Survival Prediction, proposé sur Kaggle.

Le projet Titanic Survival Prediction se décompose en plusieurs étapes. Tout d'abord, il s'agit de nettoyer, préparer et analyser les données. Ensuite, une analyse exploratoire des données (EDA) est réalisée pour mieux comprendre les caractéristiques et tendances du dataset. Enfin, un modèle d'apprentissage automatique (ML) est développé pour prédire les probabilités de survie des passagers du Titanic.

Parallèlement à ces étapes, les bonnes pratiques d'ingénierie logicielle enseignées en cours sont mises en œuvre, afin d'assurer une structure, une qualité et une maintenabilité optimales du projet.

Pour garantir un code propre, lisible et conforme aux bonnes pratiques, nous suivrons les conventions établies par PEP 8, qui favorisent une meilleure compréhension et une maintenance simplifiée du code. Afin de s'assurer que le programme fonctionne correctement, même en cas d'évolutions futures, et de renforcer sa robustesse, nous mettrons en place des tests unitaires. Ces tests permettront d'identifier rapidement d'éventuelles régressions ou erreurs.

Pour une collaboration efficace au sein de l'équipe, nous utiliserons GitHub, un outil essentiel pour le suivi des versions, la gestion des branches et le partage du code. GitHub permettra également une gestion claire des contributions et des revues de code, favorisant ainsi la qualité globale du projet.

Enfin, pour automatiser les étapes clés du développement, nous configurerons un pipeline d'intégration et de déploiement continu (CI/CD) à l'aide de GitHub Actions. Ce pipeline permettra d'exécuter automatiquement les tests, de vérifier la conformité du code grâce au linting, et, si nécessaire, de déployer les nouvelles versions de manière rapide et fiable. Cette approche garantit un processus de développement fluide, réduit les erreurs humaines et accélère la livraison des fonctionnalités.

# 2. Refactorisation et structuration du projet

## 2.1 Division en modules Python

Le notebook initial, contenant l'ensemble des étapes du projet Titanic Survival Prediction, a été refactorisé en plusieurs scripts Python modulaires. Cette refactorisation visait à améliorer la clarté, la maintenabilité et la réutilisabilité du code. Chaque script a été conçu pour remplir un rôle précis dans le pipeline du projet :

- **data_preprocessing.py :** Ce module est dédié au nettoyage des données et à leur préparation. Il contient les fonctions permettant de charger les données brutes, de transformer les variables catégoriques en numériques (comme le genre des passagers), et de sélectionner les colonnes pertinentes pour la prédiction.
- **model_training.py :** Ce script prend en charge l'entraînement du modèle de machine learning. Il utilise les données préparées pour entraîner un modèle de type Random Forest, qui est ensuite sauvegardé pour être réutilisé.
- **model_evaluation.py :** Ce module gère l'évaluation du modèle. Il recharge le modèle entraîné et l'utilise pour faire des prédictions sur les données de test. Les résultats sont exportés sous forme d'un fichier CSV, conforme aux exigences de Kaggle.

## 2.2 Gestion des dépendances et qualité du code

Pour garantir la reproductibilité et la qualité du projet, un environnement virtuel a été mis en place à l'aide de conda ou poetry. Cela a permis de centraliser les bibliothèques nécessaires, comme pandas et scikit-learn, et d'assurer que le projet est facilement exécutable sur n'importe quelle machine via un fichier de configuration.

Afin d'améliorer la qualité du code, plusieurs mesures ont été prises.

L'application des conventions PEP 8 : Tous les scripts Python ont été alignés sur les standards de style définis par PEP 8. Cela inclut l'utilisation de noms explicites pour les variables et fonctions, une indentation correcte et une organisation claire du code.

Import d'outils d'analyse statique : Des outils comme flake8 ou black ont été utilisés pour vérifier et corriger automatiquement les éventuels écarts par rapport aux standards de codage. Cela garantit que le code est non seulement conforme mais également lisible pour les autres développeurs.

Noms explicites et structuration : Les fonctions et modules ont été nommés de manière descriptive pour refléter leur rôle précis dans le pipeline du projet.

# 3. Mise en œuvre des bonnes pratiques

## 3.1 Ajout de tests unitaires

Dans ce projet, les tests unitaires ont été utilisés pour vérifier plusieurs aspects clés. Par exemple, des tests ont été écrits pour s'assurer que la fonction de prétraitement des données convertit correctement les variables catégoriques, comme le sexe des passagers, en variables numériques, et que seules les colonnes pertinentes sont sélectionnées pour la prédiction. De même, les fonctions d'entraînement du modèle ont été testées pour valider que le modèle Random Forest s'entraîne correctement avec les données et génère des prédictions fiables. Enfin, des tests ont été réalisés pour garantir que le fichier de soumission généré respecte le format requis, avec les colonnes attendues par la plateforme Kaggle.

Le framework pytest a été utilisé pour automatiser l'exécution de ces tests. Ce choix a été motivé par sa simplicité d'utilisation et sa capacité à générer des rapports détaillés lorsque des erreurs sont détectées. Cela permet non seulement de valider les composants du projet de manière régulière mais aussi de détecter rapidement toute régression ou anomalie introduite par des modifications ultérieures.

## 3.2 Documentation

Pour garantir une bonne compréhension et faciliter l'utilisation du projet, nous avons mis en place une documentation claire et détaillée. Nous avons concentré nos efforts de documentation sur deux aspects principaux : les docstrings dans le code et le fichier README.md.

Tout d'abord, nous avons ajouté des docstrings à chaque fonction et module du projet. Ces docstrings décrivent le rôle de chaque fonction, les paramètres d'entrée et les résultats attendus. Cela permet à chaque membre de l'équipe, mais aussi à toute personne qui rejoindrait le projet, de comprendre rapidement la logique de chaque fonction sans avoir à passer par tout le code. Cela rend le projet beaucoup plus accessible et facilite le travail collaboratif.

Ensuite, nous avons créé un fichier README.md pour fournir une vue d'ensemble complète du projet.

Ce fichier inclut plusieurs sections importantes :

- Objectif du projet : Présentation du projet avec contexte et objectif
- Organisation du projet : Organisation du projet sur GitHub
- Instructions d'installation : Les bibliothèques utilisées ainsi que les dépendances

## 4. Collaboration avec GitHub

Pour assurer une collaboration efficace, nous avons utilisé Git pour le contrôle de version et GitHub pour centraliser le code. Chaque membre a travaillé sur des branches locales pour développer des fonctionnalités sans affecter la branche principale (main). Les pull requests ont été utilisées pour fusionner les modifications après revue, garantissant ainsi la qualité du code.

Les commits réguliers ont permis de documenter les changements et de suivre l'évolution du projet. Cette organisation a facilité le travail en équipe, tout en assurant une gestion claire des versions et des tâches à accomplir.

## 5. Mise en place du pipeline CI/CD

Nous avons mis en place un pipeline CI/CD pour automatiser des processus essentiels et garantir la qualité du code tout au long du développement. Ce pipeline inclut des outils comme **flake8** et **black** pour vérifier que le code respecte les normes de style, ainsi que l'exécution automatique des tests unitaires à chaque modification. Cela nous permet de nous assurer qu'aucune régression n'est introduite. Grâce à cette automatisation, nous avons pu détecter rapidement les erreurs, maintenir un code stable et réduire les tâches manuelles.

## 6. Conclusion

Ce projet a permis de mettre en pratique des bonnes pratiques d'ingénierie logicielle en refactorisant le notebook Python en scripts modulaires, améliorant ainsi la lisibilité, la maintenabilité et la réutilisabilité du code. L'application des conventions **PEP 8** et l'utilisation d'outils d'analyse statique comme **flake8** et **black** ont permis de garantir un code propre et qualitatif.
Les tests unitaires avec **pytest** ont renforcé la robustesse du projet, tandis que la documentation, incluant des docstrings et un fichier **README.md**, a facilité la compréhension et l'utilisation du projet.
L'utilisation de **GitHub** a permis une gestion claire du versioning et une collaboration efficace au sein de l'équipe grâce à des branches et des pull requests.
Enfin, la mise en place d'un pipeline **CI/CD** a automatisé des tâches essentielles, telles que le linting et l'exécution des tests, garantissant ainsi la qualité et la stabilité du code tout au long du développement, tout en réduisant les erreurs humaines et en améliorant l'efficacité du processus de développement.