

+++

Projet : Prédiction de Survie des Passagers du Titanic



Rédigé par : Mehdi BENAYED Cheikh GUEYE Aymeric MOUTTAPA Nathanaël SLAMA

Date : 02 / 02 / 2025

Sommaire

Introduction	1
Présentation des données	2
Analyse Exploratoire des données	3
Construction des modèles	4
Comparaison des modèles	5
Optimisation des Hyperparamètres	6
Validation et tests	7
Explicabilité des modèles	8
Création du fichier submission	9
CI/CD et Automatisation GitHub + Assertions	10
Conclusion	11

Introduction

*Le projet **Titanic Survival Prediction** s'est concentré sur la prédiction de la probabilité de survie des passagers du Titanic, en utilisant des informations détaillées sur ces derniers, telles que l'âge, le sexe, la classe de billet, etc. Ce projet est basé sur un notebook Python disponible sur Kaggle, qui présente une première implémentation du modèle de prédiction.*

L'objectif a été de refactoriser ce code afin de le transformer en scripts Python modulaires, en intégrant les meilleures pratiques d'ingénierie logicielle étudiées lors de la formation. Cela a impliqué la refactorisation du code du notebook, l'ajout de tests unitaires et la mise en place d'un pipeline CI/CD pour automatiser les processus de développement, de tests et de déploiement.

Objectif du projet :

L'objectif principal était de travailler sur un projet concret de data science tout en appliquant des principes d'ingénierie logicielle. Le projet a inclus plusieurs étapes clés :

1. *Refactoriser le code : Transformation du notebook en plusieurs scripts Python modulaires :*
 - *data_preprocessing.py pour le nettoyage et l'ingénierie des données.*
 - *model_training.py pour l'entraînement et la sauvegarde des modèles.*
 - *model_evaluation.py pour l'évaluation des performances des modèles.*
2. *Appliquer les bonnes pratiques d'ingénierie logicielle :*
 - *Respect des normes de style PEP 8.*
 - *Rédaction de tests unitaires avec pytest.*
 - *Ajout de docstrings pour clarifier les fonctions et modules.*
3. *Collaborer en équipe avec GitHub pour la gestion des versions et la coordination des travaux, en utilisant des stratégies de branches.*
4. *Mettre en place un pipeline CI/CD : Utilisation de GitHub Actions pour automatiser les tests, le linting et le déploiement.*
5. *Documenter le projet avec un fichier README.md détaillant les objectifs, l'utilisation du projet et les contributions de chaque membre.*

Ressources disponibles :

- *Données : Les données du Titanic ont été téléchargées depuis Kaggle. Elles contiennent des informations telles que l'âge, le sexe, la classe de billet, le port d'embarquement et la survie des passagers.*

- *Outils utilisés :*
 - *Python 3.x pour le langage de programmation.*
 - *pandas pour le traitement des données.*
 - *scikit-learn pour la création du modèle d'apprentissage automatique.*
 - *pytest pour les tests unitaires.*
 - *GitHub pour la gestion du code et la collaboration en équipe.*
 - *GitHub Actions pour l'intégration continue (CI) et le déploiement continu (CD).*
- *Documentation : Une documentation complète a été incluse pour expliquer l'utilisation et les étapes du projet.*

Travail en équipe :

L'équipe était composée de 4 membres, chacun ayant des responsabilités spécifiques afin d'assurer la réussite du projet :

1. *Mehdi - Responsable du traitement des données :*
 - *Chargé du nettoyage des données, du prétraitement et de l'ingénierie des caractéristiques.*
 - *A refactorisé le code de prétraitement du notebook en un module Python data_preprocessing.py.*
2. *Cheikh - Responsable du modèle d'apprentissage machine :*
 - *A pris en charge l'entraînement du modèle avec scikit-learn et la sauvegarde du modèle.*
 - *A réécrit le code d'entraînement et de sauvegarde du modèle dans model_training.py.*
3. *Aymeric - Responsable de l'évaluation du modèle et des tests unitaires :*
 - *A réécrit le code d'évaluation du modèle dans model_evaluation.py.*
 - *A ajouté les tests unitaires pour assurer la fiabilité du code.*
4. *Nathanael - Responsable de l'intégration continue et de la documentation :*
 - *A mis en place le pipeline CI/CD avec GitHub Actions pour automatiser les tests, le linting et le déploiement.*
 - *A rédigé le fichier README.md et ajouté des docstrings dans les scripts pour la documentation.*

Répartition des tâches :

Tâches	Membres Responsables
<i>Nettoyage et prétraitement des données</i>	<i>Mehdi</i>
<i>Entraînement du modèle</i>	<i>Cheikh</i>
<i>Évaluation du modèle et tests unitaires</i>	<i>Nathanael</i>

<i>Tâches</i>	<i>Membres Responsables</i>
<i>Mise en place du pipeline CI/CD</i>	Aymeric
<i>Documentation et README.md</i>	Aymeric

Outils de gestion de version et collaboration :

Le projet a été géré avec Git et GitHub pour faciliter la collaboration entre les membres de l'équipe. La gestion des versions a été organisée comme suit :

- *Branching Model :*
 - *main pour la version stable du projet.*
 - *develop pour la version de développement.*
 - *Chaque fonctionnalité a été développée dans une branche dédiée (par exemple, feature/data-preprocessing, feature/model-training).*
 - *Les pull requests ont été utilisées pour les revues de code avant la fusion des branches.*

CI/CD avec GitHub Actions :

Le pipeline CI/CD a été configuré avec GitHub Actions pour automatiser les processus suivants :

1. *Linting : Utilisation de flake8 pour assurer la conformité avec les bonnes pratiques de style.*
2. *Formatage : Utilisation de black pour un formatage automatique du code.*
3. *Tests unitaires : Exécution des tests avec pytest pour garantir la qualité du code.*

Livrables :

- *Dépôt GitHub : Le code source, structuré en scripts Python modulaires, avec des tests unitaires et la documentation.*
- *Documentation : Le fichier `README.md` explique le projet, son utilisation, et les contributions de chaque membre.*
- *Pipeline CI/CD : Le pipeline GitHub Actions effectue le linting, le formatage et les tests automatiquement.*

Présentation des données

Les données utilisées dans le projet proviennent de la compétition Kaggle Titanic. Elles regroupent des informations sur les passagers du Titanic, permettant de prédire leur probabilité de survie. Le jeu de données est divisé en deux parties : **train.csv** pour l'entraînement du modèle et **test.csv** pour la validation.

Pour mieux comprendre l'attendu, nous avons d'abord travaillé sur un seul script **notebook**, où toutes les étapes ont été réalisées dans un même fichier. Ce processus nous a permis d'expérimenter chaque phase, du prétraitement à l'évaluation des modèles, avant de structurer le projet en plusieurs scripts Python distincts.

Pour plus de détails et une vision complète du travail effectué, le fichier **/docs/projet_final.pdf** contient une explication détaillée de chaque étape et des choix méthodologiques.

1. Jeu de données

Les jeux de données contiennent des informations sur les passagers, telles que :

- **Pclass** : La classe de billet du passager (1, 2, 3).
- **Name** : Le nom du passager.
- **Sex** : Le sexe du passager (male, female).
- **Age** : L'âge du passager (certaines valeurs peuvent être manquantes).
- **SibSp** : Le nombre de frères et sœurs / conjoints à bord.
- **Parch** : Le nombre de parents / enfants à bord.
- **Ticket** : Le numéro de ticket.
- **Fare** : Le prix du billet.
- **Cabin** : Le numéro de cabine (souvent manquant).
- **Embarked** : Le port d'embarquement du passager (C = Cherbourg; Q = Queenstown; S = Southampton).
- **Survived** : La cible de prédiction (1 = survécu, 0 = décédé).

2. Types de données

Les données sont structurées comme suit :

- **Numériques** :

- Pclass, Age, SibSp, Parch, Fare : Ces variables sont de type numérique et contiennent des informations directement utilisables pour entraîner un modèle de prédiction.
- **Catégorielles :**
 - Sex, Embarked : Ce sont des variables catégorielles qui peuvent être transformées en variables numériques par encodage (par exemple, un encodage one-hot).
- **Texte :**
 - Name, Ticket, Cabin : Ces variables sont principalement textuelles. Certaines, comme Cabin, contiennent beaucoup de valeurs manquantes, tandis que Name et Ticket pourraient être utilisées pour extraire des informations supplémentaires, mais elles sont généralement exclues des modèles d'apprentissage automatique.

3. Prétraitement des données

Avant d'entraîner un modèle, plusieurs étapes de nettoyage et de transformation des données sont nécessaires :

- **Traitement des valeurs manquantes :**
 - Des variables comme Age et Cabin contiennent des valeurs manquantes, ce qui nécessite un traitement, comme l'imputation par la médiane pour Age ou la suppression de la colonne Cabin.
- **Encodage des variables catégorielles :**
 - Sex et Embarked peuvent être convertis en variables numériques à l'aide d'un encodage comme le LabelEncoder ou l'encodage one-hot.
- **Création de nouvelles variables :**
 - Des caractéristiques supplémentaires peuvent être créées, telles que le titre du passager extrait de la colonne Name (par exemple, Mr, Mrs, Miss), qui pourrait avoir une influence sur la survie.

4. Traitement des données dans le projet

Le traitement des données dans ce projet a été effectué à l'aide des scripts suivants :

- **data_loading.py** : Chargement des fichiers CSV (en particulier train.csv et test.csv) pour les jeux d'entraînement et de test.
- **data_preprocessing.py** : Traitement des valeurs manquantes et transformation des variables catégorielles en variables numériques.

- **eda_analysis.py** : Analyse exploratoire des données (EDA), où des graphiques et des statistiques sont générés pour mieux comprendre les relations entre les variables et leur impact potentiel sur la survie des passagers.

5. Modélisation et évaluation

Une fois les données prétraitées, plusieurs modèles d'apprentissage automatique sont utilisés pour prédire la probabilité de survie des passagers :

- **Modèles utilisés :**
 - **Régression logistique**
 - **Forêt aléatoire**
 - **XGBoost**
 - **Stacking des modèles** : Une approche combinant plusieurs modèles pour améliorer la performance.

Ces modèles sont entraînés et évalués dans les scripts `model_building.py` et `model_evaluation.py`. Chaque modèle est évalué à l'aide des scores de précision et de ROC, et les meilleurs modèles sont sauvegardés dans le répertoire `models/`.

6. Résumé des jeux de données

- **Ensemble d'entraînement (train.csv)** : Contient 891 lignes et 12 colonnes, dont la variable cible `Survived`, utilisée pour l'entraînement du modèle.
- **Ensemble de test (test.csv)** : Contient 418 lignes et 11 colonnes, sans la variable cible `Survived`. Cet ensemble est utilisé pour évaluer les performances du modèle.

Analyse Exploratoire des données

Dans cette première étape, les jeux de données d'entraînement et de test ont été chargés à partir de fichiers CSV.

Informations sur le jeu d'entraînement :

*Le jeu d'entraînement comporte **891 lignes** et **12 colonnes**. Les types de données sont principalement **int64**, **float64**, et **object**.*

- *La colonne `Age` contient **177 valeurs manquantes**.*
- *La colonne `Cabin` est particulièrement touchée, avec **687 valeurs manquantes**.*
- *La colonne `Embarked` présente **2 valeurs manquantes**.*

Informations sur le jeu de test :

Le jeu de test comporte **418 lignes** et **11 colonnes**. On observe plusieurs valeurs manquantes :

- **86 valeurs manquantes pour la colonne Age.**
- **1 valeur manquante pour Fare.**
- **327 valeurs manquantes pour Cabin.**

Cela met en lumière les colonnes à traiter spécifiquement lors de l'étape de nettoyage des données.

Exploration des premières lignes du jeu d'entraînement :

Les cinq premières lignes du jeu d'entraînement montrent les variables suivantes :

- *PassengerId, Survived, Pclass, et d'autres caractéristiques des passagers. La colonne Survived indique si le passager a survécu (1) ou non (0).*

Analyse des distributions des variables :

1. Répartition de la variable Survived :

- *Le graphique montre que le nombre de passagers ayant survécu est bien inférieur à celui des passagers non survivants, ce qui reflète un déséquilibre dans les données.*

2. Répartition des classes de passagers (Pclass) :

- *La majorité des passagers étaient dans la troisième classe, suivis de ceux de la première classe. La deuxième classe est représentée par un nombre relativement faible de passagers.*

3. Distribution de l'âge en fonction de la survie :

- *Les passagers ayant survécu sont principalement âgés de 20 à 40 ans, tandis que les non-survivants présentent une distribution d'âge plus variée, avec une concentration notable parmi les jeunes adultes.*

4. Matrice de corrélation :

- *Pclass est faiblement corrélée avec Fare, suggérant que les passagers des classes inférieures payaient moins cher.*
- *La colonne Age montre une faible corrélation avec Survived, indiquant qu'il n'existe pas de lien évident entre l'âge et la probabilité de survie.*
- *La colonne Cabin affiche une corrélation faible avec les autres variables, en grande partie à cause des nombreuses valeurs manquantes.*

Construction des modèles

Étape 1 : Gestion des valeurs manquantes

- **Âge (Age)** → Remplacé par la médiane de la classe (Pclass) pour refléter le statut socio-économique.
- **Groupe d'âge (AgeGroup)** → Catégorisation en 5 classes : Enfants (0-12), Adolescents (13-18), Jeunes adultes (19-35), Adultes (36-60), Seniors (61-80).
- **Port d'embarquement (Embarked)** → Valeurs manquantes remplies avec la valeur la plus fréquente.
- **Tarif (Fare)** → Remplacé par la médiane de Pclass dans test_df, puis regroupé en 4 catégories (FareBin) : Bas (0), Moyen (1), Élevé (2), Très élevé (3).

Étape 2 : Création de nouvelles caractéristiques

- **Interaction Age * Pclass** → Capture l'effet combiné de l'âge et du statut social.
- **Taille de la famille (FamilySize)** → Addition de SibSp et Parch + 1 (passager lui-même).
- **Voyage en solitaire (IsAlone)** → 1 si seul, 0 sinon (les personnes seules survivaient moins).
- **Titre (Title)** → Extrait du nom (Name), titres rares regroupés en "Rare", harmonisation (Mlle → Miss, Mme → Mrs).

Étape 3 : Encodage des variables

- **Taux de survie familial (FamilySurvival)** → Ajout du taux moyen de survie par FamilySize.

Encodage des variables catégoriques :

- **Sexe (Sex)** → Homme (0) / Femme (1)
- **Port (Embarked)** → S (0) / C (1) / Q (2)
- **Titre (Title)** → Converti en valeurs numériques
Pont (Deck) → Extrait de Cabin, valeurs manquantes remplacées par "M", puis converti en numérique.

Suppression des colonnes inutiles

- **PassengerId, Name, Ticket, Cabin** → Supprimés car non pertinents pour la modélisation.
- **Résumé** : Toutes ces étapes optimisent la qualité des données pour améliorer la précision du modèle .

Résultats

◊ Régression Logistique

- **Précision** : 80.45%
- **Interprétation** : Bonne performance globale, mais la classe 1 (Survived) est légèrement sous-prévue par rapport à la classe 0.
- **Matrice de confusion** :
 - 87 passagers bien classés comme **non survivants**.
 - 57 passagers bien prédits comme **survivants**.
 - 17 erreurs de prédition pour la classe 1 (survivants).

Random Forest

- **Précision** : 82.68%
- **Interprétation** : Meilleure performance que la régression logistique, avec un meilleur équilibre entre classes.
- **Matrice de confusion** :
 - 91 non survivants bien classés.
 - 57 survivants bien prédits.
 - Une légère amélioration dans la détection des survivants par rapport à la régression logistique.

Multi-Layer Perceptron (MLP)

- **Précision** : 79.89%
- **Interprétation** : Comparable à la régression logistique mais avec une détection des survivants un peu plus faible.
- **Matrice de confusion** :
 - 91 non survivants bien classés.
 - 52 survivants bien prédis, mais 22 erreurs dans la détection des survivants.
 - Sensibilité plus faible sur la classe 1 (survivants).

XGBoost

- **Précision** : 82.12%
- **Interprétation** : Très proche de Random Forest, il capture bien les relations complexes entre les variables.
- **Matrice de confusion** :
 - 91 non survivants bien classés.
 - 56 survivants bien prédis.
 - Erreurs de classification légèrement plus élevées que Random Forest.

Comparaison des modèles

◊ **Régression Logistique**

1. Meilleure précision moyenne (82.17%)

Faible variation ($\pm 1.77\%$), indiquant un modèle stable

Modèle linéaire, ce qui limite ses performances sur des relations complexes

Random Forest

2. Bonne précision (80.33%), légèrement inférieure à la régression logistique

Variation un peu plus élevée ($\pm 1.97\%$), ce qui signifie qu'il est un peu plus sensible aux données

Interprétation moyenne mais améliorable avec l'importance des features

XGBoost

3. Précision légèrement plus faible (79.07%)

Variation la plus élevée ($\pm 2.35\%$), indiquant une sensibilité plus marquée aux changements de données

Optimisation des Hyperparamètres

Régression Logistique

- Les paramètres les plus adaptés sont :
 - **C fixé à 1**, ce qui équilibre bien la régularisation.
 - **200 itérations** pour la convergence.
 - **Liblinear** comme solveur, recommandé pour les petits ensembles de données.
 - Un avertissement indique que le solveur **n'a pas convergé** complètement, ce qui suggère d'augmenter le nombre d'itérations ou de normaliser les données pour améliorer la stabilité de l'entraînement.
 - Le modèle est enregistré pour une utilisation future.
-

Forêt d'arbres décisionnels

- Après des essais successifs, les paramètres retenus sont :
 - **200 arbres** pour assurer un bon compromis entre performance et temps d'exécution.
 - **Profondeur limitée à 10**, ce qui évite un modèle trop complexe.
 - **Seuil minimal de division fixé à 2 et nombre minimal d'échantillons par feuille à 5**, réduisant le risque de sur-ajustement.
 - **Utilisation du bootstrap**, renforçant la robustesse du modèle.
 - La recherche des réglages a été optimisée grâce à une approche plus rapide qui évalue un sous-échantillon des données.
 - Le modèle final a été sauvegardé pour les prochaines étapes.
-

XGBoost

- La configuration optimale inclut :
 - **100 arbres**, suffisant pour un bon équilibre entre précision et rapidité.
 - **Profondeur des arbres fixée à 3**, empêchant le modèle de devenir trop complexe.
 - **Un taux d'apprentissage de 0.01**, permettant une meilleure stabilité.
 - **Sous-échantillonnage des données à 80%**, évitant un ajustement excessif aux données d'entraînement.
 - Les tests sur plusieurs divisions du jeu de données ont donné un **taux de réussite moyen de 83,43%** et une **précision finale de 81,01%** sur un jeu de validation indépendant.
 - Le modèle est maintenant prêt à être utilisé.
-

Bilan

- Les trois modèles ont été ajustés pour tirer le meilleur parti des données.
- Les résultats obtenus montrent un bon équilibre entre simplicité et performance.

- L'étape suivante consiste à charger ces modèles et les appliquer aux nouvelles données

Validation et tests

Lors du chargement des modèles, la régression logistique, RandomForest et XGBoost sont récupérés à partir des fichiers enregistrés. Cette étape est cruciale car elle garantit que les modèles préalablement entraînés sont bien disponibles et prêts à être utilisés sans nécessiter un nouvel apprentissage.

Une fois ces modèles chargés, on passe au stacking. Chaque modèle effectue d'abord des prédictions sur les données d'entraînement. Ces prédictions sont ensuite combinées dans un nouveau jeu de données qui servira d'entrée à un modèle méta, en l'occurrence une régression logistique. Ce modèle méta apprend à exploiter les forces de chaque modèle de base pour améliorer les performances globales.

*Ensuite, on applique le même processus sur l'ensemble de validation : les prédictions des modèles de base sont générées et utilisées comme entrée du modèle méta, qui effectue sa propre prédition finale. Les résultats obtenus montrent une **précision de 81%**, avec une bonne capacité à distinguer les passagers ayant survécu de ceux qui n'ont pas survécu.*

L'analyse de la matrice de confusion révèle que le modèle empilé est plus performant pour identifier les survivants que les modèles individuels.

Enfin, le modèle méta est sauvegardé afin de pouvoir être réutilisé plus tard sans avoir à refaire tout le processus. Cette approche assure une meilleure robustesse et exploite la complémentarité des modèles pour maximiser la précision de la prédition.

Explicabilité des modèles

L'analyse des modèles montre comment différentes caractéristiques influencent les prédictions de survie.

La régression logistique met en avant le sexe comme le facteur prédominant : être une femme augmente considérablement les chances de survie. Ensuite, le taux de survie familial joue un rôle important, suggérant que les passagers issus de familles ayant survécu ont eux-mêmes de meilleures chances. La classe du billet (Pclass) suit, confirmant que les passagers des classes supérieures étaient favorisés. D'autres variables comme le titre, la solitude (IsAlone) et l'âge interviennent également, bien que dans une moindre mesure.

*Le modèle RandomForest, lui, attribue plus d'importance au tarif du billet et à l'âge, indiquant que ces variables sont plus déterminantes dans ses décisions. Le titre du passager et son sexe restent parmi les facteurs influents, mais des interactions comme Age*Pclass et le pont (Deck) apparaissent comme des éléments structurants du modèle.*

L'importance des caractéristiques par permutation renforce ces observations. Le sexe reste la variable la plus déterminante, suivi de l'interaction âge-classe qui reflète bien la hiérarchie sociale à bord du Titanic. L'embarquement, le tarif et la taille de la famille influencent également la décision du modèle.

Enfin, l'explication SHAP d'une prédiction montre visuellement comment chaque variable influe sur la probabilité de survie pour un individu donné. Le sexe et le pont du navire modifient le plus fortement cette probabilité, tandis que d'autres caractéristiques comme la classe et le titre ajustent légèrement l'évaluation finale du modèle.

L'ensemble de ces analyses permet de mieux comprendre comment chaque modèle prend ses décisions et quelles variables impactent réellement les prévisions de survie.

Création du fichier submission

Le fichier de soumission est généré en utilisant les prédictions des modèles préalablement entraînés. Avant tout, les caractéristiques du jeu de test sont alignées avec celles du jeu d'entraînement, garantissant que le modèle traite des données au format attendu. Chaque modèle de base – régression logistique, RandomForest et XGBoost – produit ses propres prédictions sur ces données. Ces résultats sont ensuite regroupés dans un nouveau jeu de données qui sert d'entrée à un modèle méta basé sur la régression logistique. Ce dernier affine les décisions en combinant les forces de chaque modèle de base pour améliorer la précision des prédictions.

Une fois les résultats obtenus, le fichier de soumission est construit en associant les identifiants des passagers aux prédictions finales. Le fichier est ensuite sauvegardé sous le format requis, prêt à être soumis. Cette approche garantit une exploitation optimale des modèles tout en maintenant la cohérence entre l'entraînement et la phase finale de prédiction.

CI/CD et Automatisation GitHub + Assertions

L'ensemble des assertions effectuées dans cette partie visait à garantir que les données et les modèles étaient correctement structurés et opérationnels. L'objectif était de s'assurer que les fichiers de données et de modèles existaient bien, que les variables étaient correctement nommées et que les transformations appliquées respectaient les attentes initiales.

En réalisant ces tests, on a pu vérifier que les fichiers étaient bien chargés, que les variables essentielles étaient présentes et que les transformations sur les données

correspondaient aux étapes précédemment définies. Ces vérifications ont confirmé que le prétraitement des données et l'entraînement des modèles ont bien fonctionné, en validant les mêmes résultats que ceux obtenus auparavant.

L'ensemble du processus montre que les modèles sont bien formés et enregistrés correctement. Le pipeline d'entraînement et de prédiction est fonctionnel, avec des performances alignées sur les attentes. Cela confirme la cohérence des étapes mises en place et la fiabilité du modèle utilisé.

L'architecture du projet est organisée de façon claire et modulaire pour assurer une bonne gestion du code, de la documentation et des modèles.

- *Le dossier **/docs** contient toute la documentation essentielle, avec un **README.md** expliquant le projet, un **INSTALLATION.md** pour l'installation et un **CONTRIBUTING.md** pour les contributions. Le fichier **projet_final.pdf** est le rapport détaillé pour la restitution.*
- *Le dossier **/models** stocke les modèles entraînés : **logistic_regression_model.pkl**, **random_forest_model.pkl**, **xgboost_model.pkl** et **stacking_meta_model.pkl**. Il contient aussi **submission.csv**, utilisé pour évaluer le modèle sur Kaggle.*
- *Le dossier **/src** regroupe les scripts du projet*
- *Le dossier **/tests** sert à la validation et l'optimisation des modèles*
- *Enfin, d'autres fichiers sont présents à la racine du projet : **LICENSE** pour la licence, **requirements.txt** pour les dépendances et divers fichiers CSV contenant les données brutes et nettoyées. Cette organisation permet une exécution fluide et structurée du projet.*

Voici l'architecture :

```
titanic-survival-predict/
|
|   |-- .github/
|   |   |-- workflows/
|   |   |   `-- ci.yml      # Pipeline CI/CD avec GitHub Actions
|
|   |-- docs/          # Documentation
|   |   |-- README.md      # Explication du projet
|   |   |-- INSTALLATION.md    # Instructions pour installer et exécuter le projet
|   |   |-- CONTRIBUTING.md    # Guide de contribution pour les membres
|   |   |-- projet_final.pdf    # Rapport détaillé pour la restitution
|
|   |-- models/         # Modèles sauvegardés
|   |   |-- logistic_regression_model.pkl
|   |   |-- random_forest_model.pkl
|   |   |-- xgboost_model.pkl
```


Conclusion

Après avoir refactorisé et structuré le projet Titanic Survival Prediction en scripts modulaires, nous avons appliqué les bonnes pratiques d'ingénierie logicielle : modularisation du code, tests unitaires, documentation claire et mise en place d'un pipeline CI/CD. L'utilisation de GitHub et de GitHub Actions a permis une gestion efficace du projet en équipe, avec un suivi rigoureux des évolutions.

Les modèles développés ont été validés à plusieurs niveaux, en passant par des tests sur les données, des comparaisons de performances et une interprétabilité approfondie grâce à l'IA explicable (XAI). Les prédictions finales, soumises sur [Kaggle](#), ont confirmé la robustesse de notre approche avec un score avoisinant les 80 % de précision, validant ainsi nos choix méthodologiques.

Ce projet nous a permis d'intégrer des compétences essentielles en ingénierie logicielle appliquée à la data science, en mettant en place une méthodologie reproductive et rigoureuse, adaptée aux standards professionnels. L'expérience acquise en structuration de projet, collaboration via Git et automatisation des tests sera précieuse pour nos futurs travaux dans le domaine de la data et du développement logiciel.