Projet de synthèse 2025

Session 2023-2025

Auteurs : NIDHEL WASSYL BENHACINE & AMINE AIBOUT

**Optimisation de la Fiabilité et Réduction des Pannes grâce à la Maintenance Prédictive**

Guide Complet de Réalisation d’un Projet de Maintenance Prédictive et Détection d’Anomalies avec Intégration de Modèles, API, et Interface Utilisateur

# TABLE DES MATIÈRES

Introduction ……………………………...………………………………………………. 2

Étape 1 : Analyse et préparation initiale ………………………………………………. 3

Étape 2 : Préparation et exploration des données ……………………………………. 3

Étape 3 : Développement du modèle 1 – Prédiction des pannes ……………………. 3

Étape 4 : Développement du modèle 2 – Imputation des données manquantes …… 4

Étape 5 : Enregistrement et gestion des modèles ……………………………………... 4

Étape 6 : Développement de l’API Flask ………………………………………………… 4

Étape 7 : Développement de l’interface utilisateur (Front-End) ………………………... 5

Étape 8 : Déploiement avec Docker ……………………………………………………… 5

Étape 9 : Validation finale …………………………………………………………………. 5

Résumé des outils utilisés …………………………………………………………………. 6

## INTRODUCTION

**CONTEXTE**

Les systèmes de production d'air dans les trains, notamment les compresseurs, sont essentiels au bon fonctionnement des freins et de la climatisation. Ces équipements, sensibles aux variations de conditions telles que la pression, la température et le courant moteur, voient leur fiabilité affectée par ces fluctuations. Dans l'environnement opérationnel ferroviaire, des pannes imprévues représentent des défis majeurs pour la sécurité et l’efficacité.  
Le suivi de ces équipements à l’aide de capteurs analogiques et numériques permet de mieux comprendre leur comportement dans le temps. Cette surveillance fournit des données essentielles pour anticiper les problèmes et optimiser les interventions de maintenance, réduisant ainsi les interruptions de service et améliorant la sécurité. L’analyse en temps réel devient ainsi cruciale dans le domaine ferroviaire, où l’efficacité et la sécurité sont des priorités absolues.

**OBJECTIF**

Développer un modèle de machine learning capable de prédire les pannes des compresseurs en détectant les anomalies sur la base des données temporelles collectées par des capteurs. Le modèle doit fournir des prédictions fiables pour anticiper les interventions de maintenance et réduire les risques de défaillance.

Ce document présente un guide complet pour la réalisation d’un projet de maintenance prédictive et de détection d’anomalies, incluant le développement et le déploiement des modèles, l’API backend, et une interface utilisateur front-end. Un document complémentaire, partagé sur GitHub, explique et présente en détail le jeu de données MetroPT-3 : <https://github.com/Netamine/APU_ANOMALY_DETECTION/blob/main/documentation/Presentation_du_Jeu_de_Donnees_MetroPT.docx>.

## Étape 1 : Analyse et préparation initiale

**OBJECTIF** : Comprendre les besoins et planifier le projet.

**DÉTAILS :**

* Identifier les objectifs : prédire les pannes et compléter les valeurs manquantes.
* Analyser les données disponibles (fichiers CSV des capteurs).
* Définir les outils nécessaires.

**OUTILS UTILISÉS :** Notion, MLflow.

## Étape 2 : Préparation et exploration des données

**OBJECTIF :** Nettoyer et comprendre les données pour les rendre exploitables.

**DÉTAILS :**

* Charger les données depuis les fichiers CSV avec pandas.
* Identifier et traiter les valeurs manquantes.
* Supprimer les anomalies et normaliser les données.
* Visualiser les relations entre variables (corrélations, tendances) avec des graphiques.

**OUTILS UTILISÉS :** Pandas, NumPy, Matplotlib, Seaborn, MLflow.

## Étape 3 : Développement du modèle 1 – Prédiction des pannes

**OBJECTIF :** Construire un modèle pour prédire les pannes à partir des données.

**DÉTAILS :**

* Ajouter une colonne cible (0 pour normal, 1 pour panne, 2 pour panne dans 1 heure).
* Extraire des caractéristiques comme les moyennes mobiles ou les gradients.
* Entraîner un modèle de classification (Random Forest, XGBoost).
* Enregistrer les expérimentations dans MLflow :
  + Hyperparamètres (nombre d’estimations, profondeur maximale).
  + Métriques de performance (précision, rappel, F1-score).
  + Artefacts du modèle entraîné (fichiers .pkl ou .h5).

**OUTILS UTILISÉS:** Jupyter Notebook, scikit-learn, XGBoost, MLflow.

## Étape 4 : Développement du modèle 2 – Imputation des données manquantes

**OBJECTIF :** Construire un modèle pour compléter les plages de données incomplètes.

**DÉTAILS :**

* Simuler des valeurs manquantes pour entraîner le modèle.
* Entraîner un modèle de régression (Random Forest Regressor, Autoencodeur).
* Enregistrer les expérimentations dans MLflow :
  + - Métriques (MAE, RMSE).
    - Hyperparamètres (profondeur, nombre d’estimations).
    - Artefacts du modèle.

**OUTILS UTILISÉS :** Jupyter Notebook, scikit-learn, TensorFlow/Keras, MLflow.

## Étape 5 : Enregistrement et gestion des modèles

**OBJECTIF :** Centraliser la gestion des modèles avec MLflow.

**DÉTAILS :**

* Enregistrer les modèles entraînés dans le registre MLflow pour versionner et centraliser les modèles.
* Utiliser MLflow pour charger les modèles dans l’API Flask.

**OUTILS UTILISÉS :** MLflow Model Registry.

## Étape 6 : Développement de l’API Flask

**OBJECTIF :** Exposer les modèles via des endpoints RESTful.

**DÉTAILS :**

* + - Créer des endpoints RESTful :
    - POST /predict-failure : Utiliser le modèle de prédiction des pannes.
    - POST /fill-missing-data : Utiliser le modèle d’imputation des données.
    - Charger les modèles depuis MLflow.
    - Documenter l’API avec Swagger.

**OUTILS UTILISÉS :** Flask, Swagger, MLflow, Postman.

## Étape 7 : Développement de l’interface utilisateur (Front-End)

**OBJECTIF :** Permettre aux utilisateurs de saisir leurs données et visualiser les prédictions.

**DÉTAILS :**

* + - Créer une interface utilisateur avec React.js :
    - Formulaire pour saisir les données.
    - Graphiques pour afficher les résultats.
    - Connecter le front-end à l’API Flask via des appels HTTP.

**OUTILS UTILISÉS :** React.js, Axios, Chart.js, Plotly.

## Étape 8 : Déploiement avec Docker

**OBJECTIF :** Conteneuriser l’application pour simplifier le déploiement.

**DÉTAILS :**

* + - Créer un Dockerfile pour inclure Flask, MLflow et les modèles.
    - Déployer le conteneur sur une machine Linux.

**OUTILS UTILISÉS :** Docker, MLflow.

## Étape 9 : Validation finale

**OBJECTIF :** Vérifier le bon fonctionnement et la robustesse du système.

**DÉTAILS :**

* + - Tester les modèles et l’API avec des scénarios réels.
    - Valider les résultats auprès des utilisateurs finaux.
    - Analyser les performances avec les métriques enregistrées dans MLflow.

**OUTILS UTILISÉS :** Postman, Navigateur Web, MLflow UI.

## Résumé des outils utilisés

|  |  |  |  |
| --- | --- | --- | --- |
| **Nom de l'outil** | **Étape dans le projet** | **Description** | **Lien pour documentation** |
| **Notion** | Analyse et préparation initiale | Outil de gestion des tâches et de la documentation. | [Notion Documentation](https://www.notion.so/help) |
| **MLflow** | Gestion des modèles | Plateforme open source pour le suivi des expérimentations, la gestion et le déploiement des modèles. | [MLflow Documentation](https://mlflow.org/docs/latest/index.html) |
| **MLflow Model Registry** | Enregistrement et gestion des modèles | Outil de MLflow pour centraliser et gérer les versions des modèles. | [MLflow Model Registry](https://mlflow.org/docs/latest/model-registry.html) |
| **pandas** | Exploration et préparation des données | Librairie Python pour la manipulation et l’analyse des données structurées. | [pandas Documentation](https://pandas.pydata.org/docs/) |
| **NumPy** | Exploration et préparation des données | Librairie Python pour les calculs numériques avancés. | [NumPy Documentation](https://numpy.org/doc/) |
| **Matplotlib** | Exploration et préparation des données | Librairie Python pour la visualisation de données sous forme de graphiques. | [Matplotlib Documentation](https://matplotlib.org/stable/contents.html) |
| **Seaborn** | Exploration et préparation des données | Librairie Python pour la visualisation statistique. | [Seaborn Documentation](https://seaborn.pydata.org/) |
| **Jupyter Notebook** | Développement des modèles | Environnement interactif pour écrire et exécuter du code Python. | [Jupyter Documentation](https://jupyter.org/documentation) |
| **scikit-learn** | Développement des modèles | Librairie Python pour l’apprentissage automatique supervisé et non supervisé. | [scikit-learn Documentation](https://scikit-learn.org/stable/documentation.html) |
| **XGBoost** | Développement des modèles | Librairie Python optimisée pour les algorithmes de gradient boosting. | [XGBoost Documentation](https://xgboost.readthedocs.io/) |
| **TensorFlow/Keras** | Développement des modèles | Librairie pour le deep learning, utile pour les autoencodeurs et autres réseaux neuronaux. | [TensorFlow Documentation](https://www.tensorflow.org/learn) |
| **Flask** | Développement de l’API | Microframework Python pour créer des API et applications web. | [Flask Documentation](https://flask.palletsprojects.com/en/2.3.x/) |
| **Swagger** | Développement de l’API | Outil pour documenter les API de manière interactive. | [Swagger Documentation](https://swagger.io/docs/) |
| **Postman** | Développement et test de l’API | Application pour tester et automatiser les requêtes API. | [Postman Documentation](https://learning.postman.com/docs/getting-started/introduction/) |
| **React.js** | Développement de l’interface utilisateur (front-end) | Librairie JavaScript pour construire des interfaces utilisateur interactives. | [React.js Documentation](https://reactjs.org/docs/getting-started.html) |
| **Axios** | Développement de l’interface utilisateur (front-end) | Librairie JavaScript pour effectuer des appels HTTP. | [Axios Documentation](https://axios-http.com/docs/intro) |
| **Chart.js** | Développement de l’interface utilisateur (front-end) | Librairie JavaScript pour créer des graphiques interactifs. | [Chart.js Documentation](https://www.chartjs.org/docs/latest/) |
| **Plotly** | Développement de l’interface utilisateur (front-end) | Librairie pour la création de graphiques interactifs avancés en Python ou JavaScript. | [Plotly Documentation](https://plotly.com/python/) |
| **Docker** | Déploiement | Outil de conteneurisation pour packager et déployer les applications. | [Docker Documentation](https://docs.docker.com/) |
| **FileZilla** | Déploiement | Outil pour transférer des fichiers entre machines via FTP/SFTP. | [FileZilla Documentation](https://filezilla-project.org/documentation.php) |
| **PuTTY** | Déploiement | Client SSH pour accéder à des serveurs distants. | [PuTTY Documentation](https://www.putty.org/) |
| **MLflow UI** | Validation finale | Interface utilisateur de MLflow pour visualiser les expérimentations et métriques. | [MLflow UI Documentation](https://mlflow.org/docs/latest/tracking.html#mlflow-tracking-ui) |