**Rapport Projet 2**

[Introduction 1](#_Toc1)

[Partie API 2](#_Toc2)

[Entraînement des modèles 2](#_Toc3)

[API 2](#_Toc4)

[Authentification 2](#_Toc5)

[Enpoints 3](#_Toc6)

[/ 3](#_Toc7)

[/docs 3](#_Toc8)

[/info 3](#_Toc9)

[/score 3](#_Toc10)

[/predict/{model} 3](#_Toc11)

[Tests 3](#_Toc12)

# **Introduction**

Ce projet fait suite à un premier projet dans lequel des modèles de machine learning ont été entraîné sur des données préalablement préparées. Ces procédures de traitements et d’entraînement ainsi que les modèles sauvegardés sont réutilisés pour ce projet afin de déployer une API permettant d’interroger ces modèles.

Le projet est disponible sur Github à cette [adresse](https://github.com/seilebost/mai21_cde_rains)

# Partie API

L’API permet d’interroger des modèles de machine learning préalablement entraînés pour obtenir des prédictions et le score des modèles sur les données de tests.

## Entraînement des modèles

La première étape a été l’entraînement des différents modèles. Pour ce faire un script **train.py** a été développé. Il récupère le fichier *rain.csv* qui contient les données météo brute, les nettoyer puis entraîner les différentes modèles avec ces données.

Lors du nettoyage certaines lignes non pertinentes son retirées notamment celles contenant des valeurs nulles pour la variable cible. Le script va ensuite créé un pipeline de pre-processing pour remplacer les valeurs nulles des variables catégorielle et quantitative, standardiser les données et encoder les variables catégorielle.

Ce pipeline est sauvegardé dans un fichier *preprocessor* qui est utilisé dans l’API pour appliquer les même traitements aux données fournies par l’utilisateur lors des prédictions.

Les données sont séparées en jeu d’entraînement et de test, les modèles sont entraîné puis sauvegardé dans un fichier pour être utilisé par l'API.

## API

L'API est dévelopée avec Python 3 et le framework FastAPI

Une fois le projet récupérer et les paquets du fichier *requirements.txt* installés, il est possible de tester l'API via la commande `python3 api\_models.py` qui est alors accessible à l’adresse *http://localhost:8000*

### Authentification

Hormis pour les endpoints **/** et **/docs** il est nécessaire de s'authentifier via le header **authorization-header**. Le header doit être de cette forme : *Basic b64string*

b64string est une chaîne de caractère en base64 correspond correspondant à un couple *user:password*

Par exemple on peut s'authentifier comme **authorization-header** : *Basic YWxpY2U6d29uZGVybGFuZA==* avec *YWxpY2U6d29uZGVybGFuZA==* étant la chaîne de caractères *alice:wonderland* encodée en base64.

Les différents couples d’utilisateur et mot de passe autorisés se trouve le fichier de l’API. Cette méthode d’authentification reste basique et pourrait être améliorée par exemple en stockant les identifiants dans une base données et en demandant l’authentification via un formulaire.

### Endpoints

Cette partie liste les différents endpoints de l’API et leur description.

#### /

Cet endpoint permet de contrôler si l'API est en fonctionnement. Il renvoie un objet JSON indiquant que l’API est en fonctionnement et peut notamment être utilisé par des tâches automatisées pour vérifier que l’API est toujours disponible

#### /docs

Fastapi fournit cet endpoint qui permet d'accéder à une documentation de l'API et de tester les différents endpoints. On peut y trouver une description des différents endpoints et des schemas et il est possible d’effectuer des requêtes depuis l’interface et d’obtenir le retour des endpoints.

#### /info

Renvoie la liste des modèles entraînés qui sont disponibles dans l’API.

#### /score

Cet endpoint nécessite de sélectionner le nom d’un modèle et de l’inclure dans le corps de la requête. Le résultat renvoyé est le score obtenu par ce modèle sur les données de test.

#### /predict/{model}

Ici le routage dynamique de FastAPI permet de générer un endpoint pour chaque modèle défini dans la classe **ModelsName.** Il est donc possible d’ajouter de nouveau modèle dans l’API qui seront directement disponible via de nouveaux endpoints.

L’API attend l’envoie d'un fichier CSV contenant les données pour lesquels on veut obtenir une prédiction. Ces données seront nettoyées et standardisées de la même manière que les données d’entraînement et de tests grâce au pipeline enregistré par le script **train.py**.

Le résultat renvoyé est une liste contenant la prédiction obtenue pour chaque entrée présente dans le fichier.

### Tests

Des tests développés avec le framework Pytest sont présents dans le dossier **tests** du dépôt.

Ces tests ont servis pour le développement de l’API afin d’éviter d’éventuelles régressions.