PROJET 2

Table des matières

1) Introduction

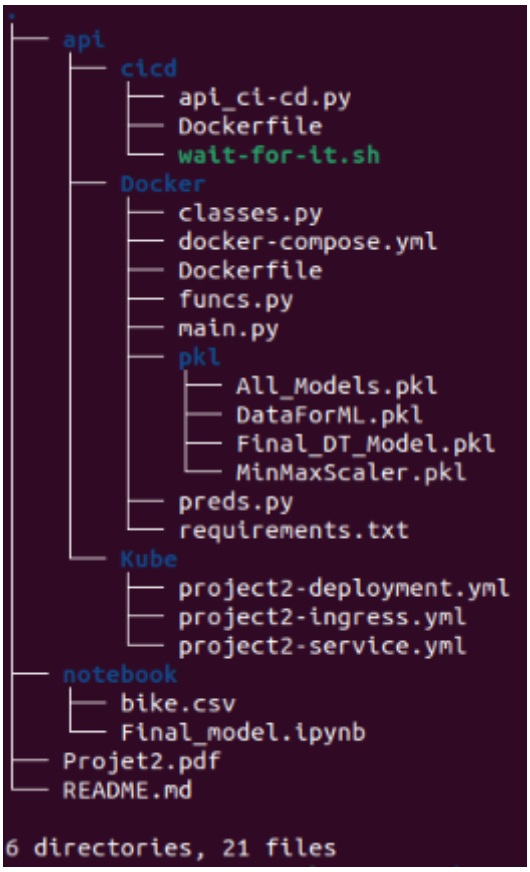
a. Contexte

L'objectif de ce projet est de déployer un modèle de Machine Learning. Le présent document a pour objectif d'expliquer les différentes étapes qui ont été nécessaires pour la réalisation de ce projet.

b. Participants

- ✓ Rémy DALLAVALLE
- ✓ Cléa MOURTZAKIS

c. Livrables



2) Analyse et Machine Learning

Le jeu de données fourni concerne des informations sur des locations de vélos pour les années 2011 et 2012. Ce jeu contient **17379 entrées**, pour **8 variables** (7 explicatives, et 1 expliquée).

Ce jeu a été audité et analysé dans le but de prédire le nombre de vélos qui seront loués en fonction de différents indicateurs, tels que le mois ou la météo.

Après normalisation et standardisation des données, plusieurs algorithmes de Machine Learning ont été évalués afin de retenir le plus performant. Dans le cas de ce projet, l'algorithme retenu est le **DecisionTreeRegressor**.

Les entraînements sur tous les algorithmes ont été sauvegardés dans un fichier **pickle** afin d'être interrogé par l'API.

Le notebook joint au projet contient les différentes étapes ainsi que des explications exhaustives. Il est à noter que les informations suivantes ont été enregistrées en pickle également afin d'être utilisées par l'API :

- Les données normalisées et standardisées (avant entraînement),
- Un scaler MinMax,
- Tous les modèles entraînés et testés
- Le modèle final

3) Développement de l'API

L'API est développée avec **FastAPI**, incluant **Swagger** pour la documentation embarquée. Un système d'authentification de type **BasicAuth** est mis en place.

a. main.py

Ce fichier contient le code principal et les routes pour l'API.

- Une route **Status** qui renvoie « **1** » si l'API fonctionne.
- Une seconde route **Predictions**, qui indiquera à l'utilisateur la prédiction quant au nombre de vélos loués pour les paramètres entrés.
- Et une route **Scores** qui interroge les différents modèles afin d'obtenir leur score respectif.

b. funcs.py

Il s'agit d'un fichier de fonctions utilisées par l'API. On y retrouve la gestion de l'authentification et des utilisateurs, ainsi qu'une fonction de « conversion » de la météo, par rapport à une entrée alphanumérique.

c. preds.py

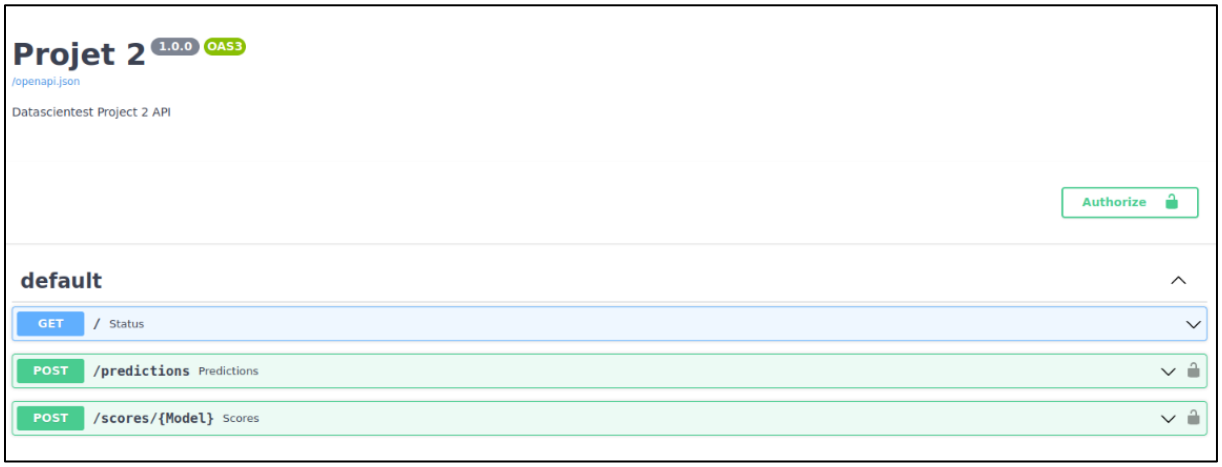
Les actions sur les modèles sont effectuées par ce fichier. Les routes de l'API appellent les fonctions développées ici pour obtenir les résultats souhaités.

d. classes.py

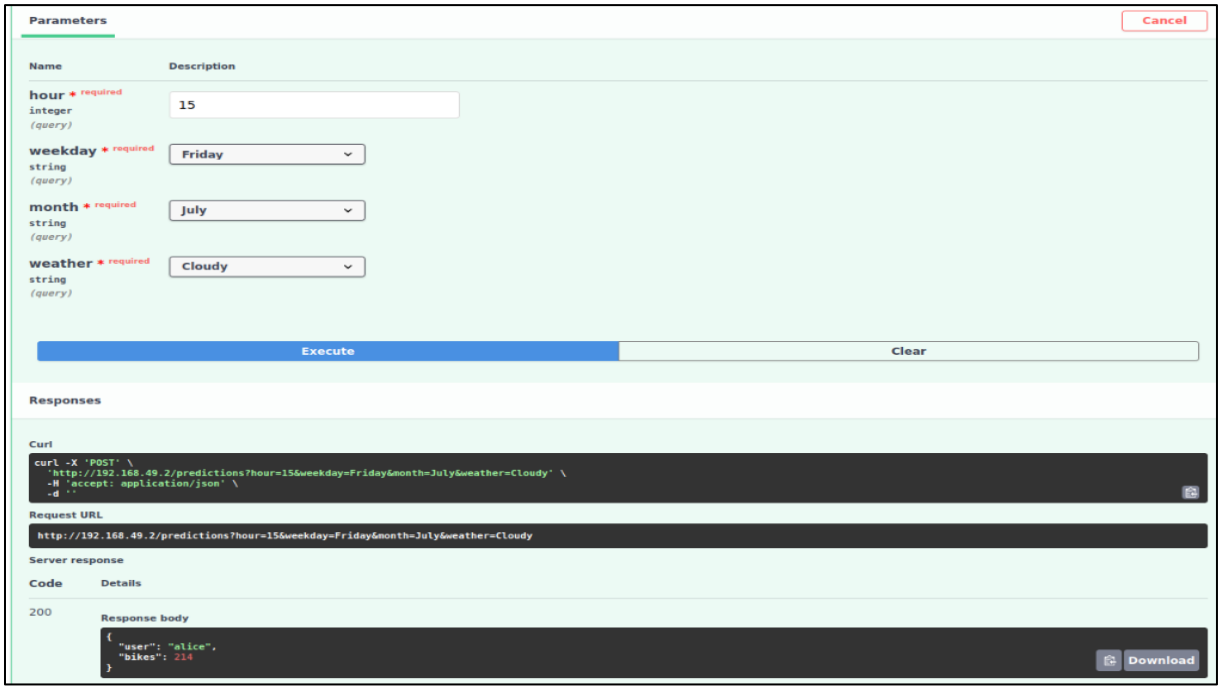
Contient la déclaration des classes utilisées par l'API, notamment pour les champs.

e. L'API

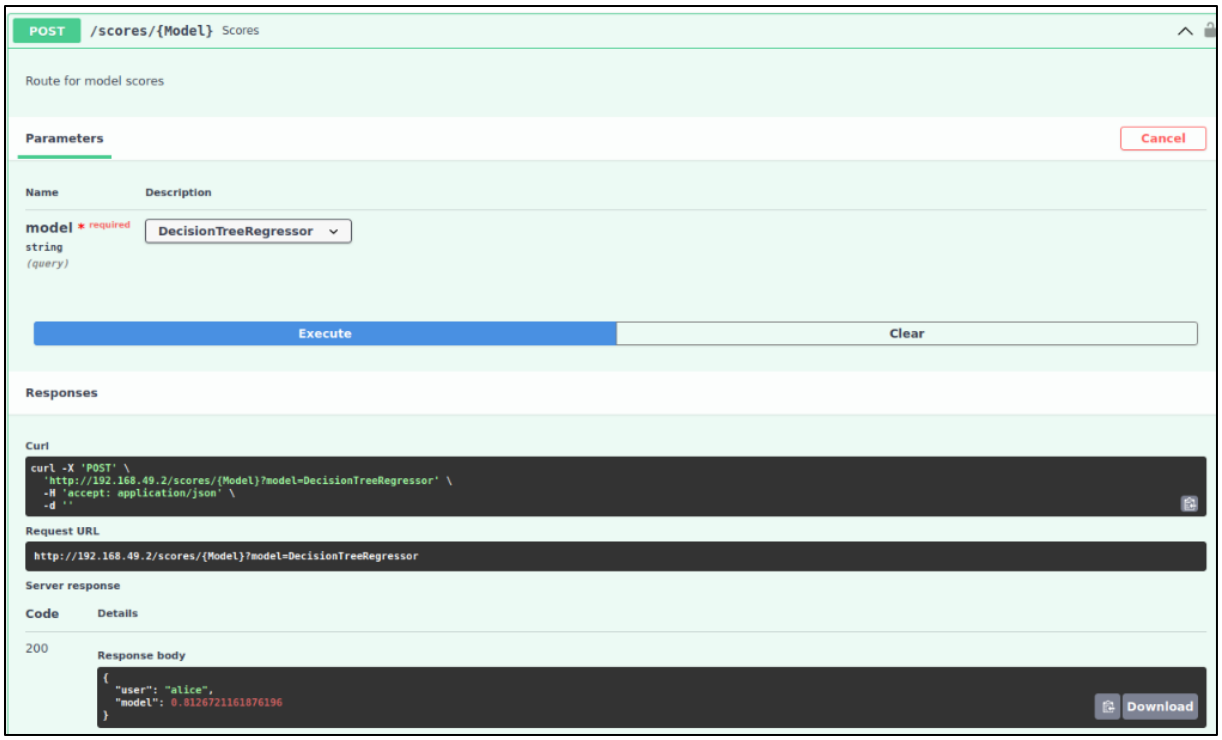
En se rendant vers le endpoint **/docs** on obtient l'interface suivante.



Pour les prédictions, plusieurs champs sont à remplir avant d'exécuter la requête (seuls des utilisateurs authentifiés peuvent interroger l'API)



Concernant les scores, une liste propose les modèles pris en compte par l'API (ici aussi, seuls des utilisateurs authentifiés peuvent interroger cette route)


POST
/scores/{Model}
Scores
Route for model scores
Parameters
Cancel
Name
Description
model
required
string
(query)
DecisionTreeRegressor
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://192.168.49.2/scores/{Model}?model=DecisionTreeRegressor' \
-H 'accept: application/json' \
-d ''
Request URL
http://192.168.49.2/scores/{Model}?model=DecisionTreeRegressor
Server response
Code
Details
200
Response body
{
"user": "alice",
"model": 0.8126721161876196
}
Download

4) Containers Docker

a. API

Le container créé pour le projet utilise une image de base **Python 3.9.11**. Les fichiers développés pour l'API sont positionnés dans un dossier **/code/** et les pickles dans un dossier **/pkl/**.

Au lieu d'installer les librairies indépendamment, celles-ci ont été enregistrées dans un fichier **requirements.txt**, lequel est utilisé à la création du container pour tout installer selon le besoin. Ces librairies sont les suivantes :

- FastAPI 0.78.0
- Pandas 1.4.2
- passLib 1.7.4
- pydantic 1.9.0
- scikit-learn 1.1.1
- uvicorn 0.15.0

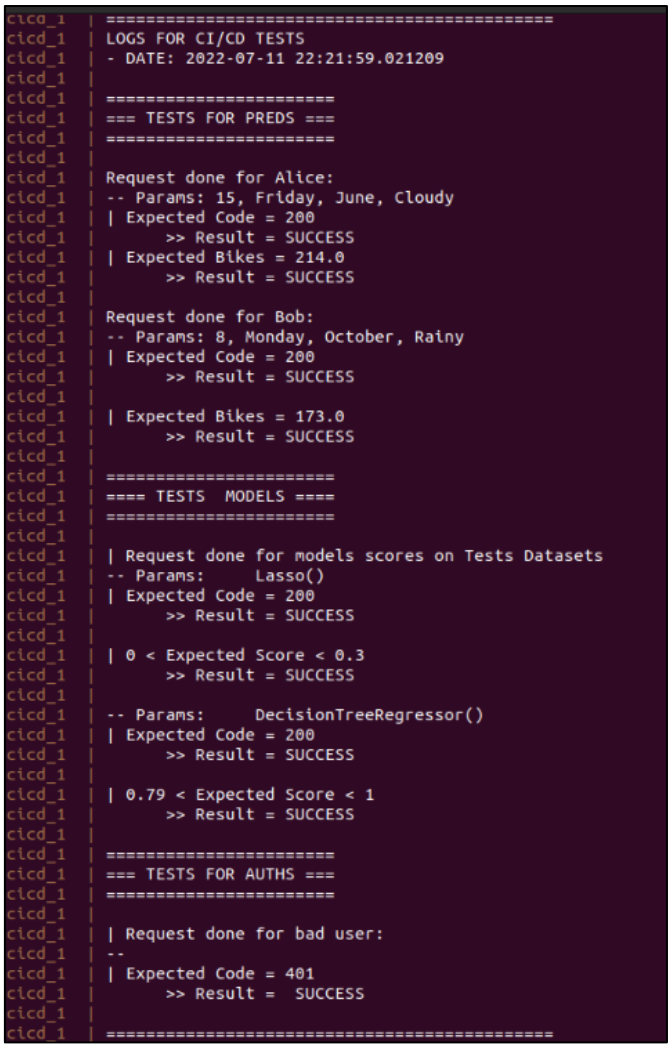
Une fois exécuté, ce container expose le port 8000, et il est possible d'accéder à l'interface de l'API.

b. Les Tests (CI/CD)

Des jeux de tests ont été développés afin de vérifier les résultats attendus. Les tests valident les points suivants :

- Exécution des prédictions pour 2 utilisateurs, avec des requêtes différentes
    - Alice : Code retour attendu 200 ; locations attendues 214
    - Bob : Code retour attendu 200 ; locations attendues 173
- Tests des modèles
    - Lasso : Code retour attendu 200 ; résultat attendu compris entre 0 et 0.3
    - DecisionTreeRegressor : Code retour attendu 200, résultat attendu supérieur à 0.79
- Connection avec un mauvais utilisateur
    - Code retour attendu 401

L'image ci-dessous montre le résultat des tests effectués.



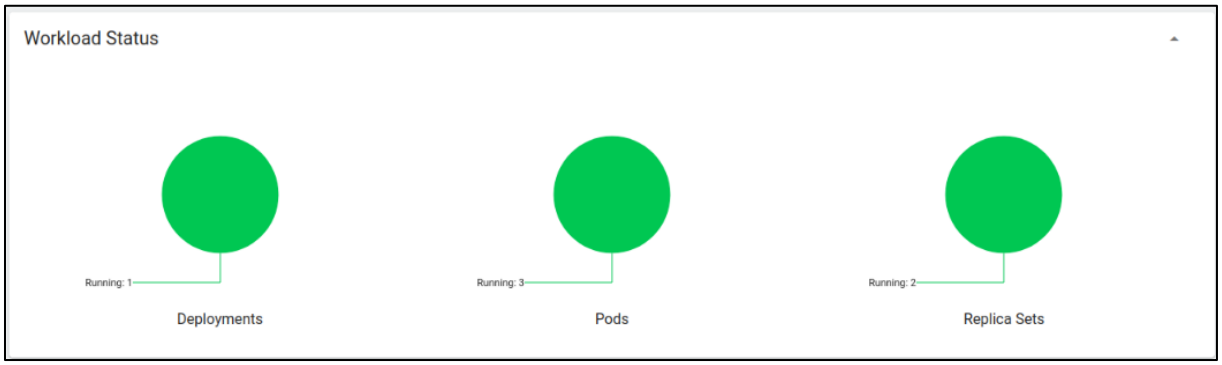
c. Docker-compose

Les 2 containers (API et CICD) ont été publiés respectivement à ces endroits : **rdallavalle/project2_api** et **rdallavalle/project2_cicd** sur **DockerHub**.

Afin que les containers s'exécutent dans le bon ordre, il a été fait le choix d'utiliser un script annexe qui vérifiera que l'API soit en cours d'exécution avant d'exécuter le container des tests. Ce script est exécuté dans le container principal.

5) Kubernetes

Le déploiement de l'API s'effectue sur 3 pods.



L'accès au endpoint peut s'effectuer directement depuis l'interface, ou en se rendant à l'adresse indiquée.