# **Documentation et Explication du Code**

## **Description Générale**

Ce script définit une application web développée avec **FastAPI**. L'application permet de télécharger un fichier, d'effectuer un traitement asynchrone des données, de prédire des valeurs sur une plage de dates donnée, et de suivre l'état des tâches en temps réel via une interface web.

## **Composants Clés**

### **1. Imports**

* **FastAPI** : Framework utilisé pour créer l'application web.
* **Jinja2Templates** : Gère la génération de templates HTML pour l'affichage des pages web.
* **Pandas** : Utilisé pour manipuler les données, notamment les séries temporelles.
* **Matplotlib** : Utilisé pour créer des graphiques.
* **BytesIO** et **StringIO** : Manipulent les flux de données en mémoire.
* **uuid** : Génère des identifiants uniques pour chaque tâche.
* **BaseModel** de **Pydantic** : Permet de définir des modèles de données pour les requêtes.
* **Warnings** : Pour ignorer les avertissements futurs qui pourraient polluer la sortie.

### **2. FastAPI App**

* **app** : Instance de l'application FastAPI.
* **templates** : Instance de Jinja2Templates pour gérer les templates HTML.

### **3. Endpoints**

* **get\_form** (GET /) : Affiche le formulaire de soumission de fichier et des options. Utilise le template home.html.
* **handle\_form** (POST /) : Gère la soumission du formulaire. Crée une nouvelle tâche avec un identifiant unique, planifie son exécution via le scheduler, et redirige l'utilisateur vers une page d'attente (waiting.html).
* **get\_status** (GET /status/{task\_id}) : Vérifie l'état d'une tâche spécifique. Retourne l'état sous forme de réponse JSON.
* **handle\_predict** (POST /predict) : Gère la requête de prédiction pour une plage de dates spécifiée. Exécute la fonction de prédiction et retourne les résultats sous forme de tableau HTML.
* **get\_results** (GET /results/{task\_id}) : Affiche les résultats d'une tâche terminée en utilisant le template result.html.

### **4. Planification des Tâches**

* **schedule\_task** : Planifie l'exécution asynchrone de la tâche de traitement des données, en utilisant la fonction importée depuis scheduler.py. La tâche est planifiée pour s'exécuter avec un léger retard pour permettre une meilleure gestion asynchrone.

### **5. Traitement et Prédiction**

* Le traitement et la prédiction sont gérés dans des fonctions distinctes, importées depuis d'autres modules (pipeline.py et predict.py). Les résultats du traitement sont stockés dans un dictionnaire global global\_store et associés à l'identifiant de tâche pour être récupérés plus tard.

### **6. Démarrage de l'Application**

* Le script utilise uvicorn pour démarrer l'application FastAPI. Le serveur est lancé sur 127.0.0.1 au port 8000 pour une exécution locale.

## **Résumé**

Ce script FastAPI met en place un système complet pour le traitement asynchrone de données. Il inclut des fonctionnalités pour télécharger des fichiers, exécuter des tâches de traitement de données en arrière-plan, suivre l'état des tâches en temps réel, et afficher les résultats de manière dynamique via une interface web. L'intégration avec des templates Jinja2 permet de personnaliser l'affichage et de fournir une expérience utilisateur fluide et interactive.