Atelier 2 : Modularisation du Code

Prérequis : Script Jupyter fonctionnel.

1. **Objectifs :**
   I. Transformer le script Jupyter de votre projet en code modulaire.
   II. Créer des fonctions réutilisables pour chaque étape du pipeline ML.
2. **Étapes :**
   I. Identification des blocs logiques dans le notebook (préparation des données, entraînement, évaluation, etc.).
   II. Définir les fonctions correspondantes :
      - **prepare_data() :** Charger et prétraiter les données.
      - **train_model() :** Entraîner le modèle.
      - **evaluate_model() :** Évaluer les performances.
      - **save_model() :** Sauvegarder le modèle entraîné.
      - **load_model() :** Charger un modèle sauvegardé.
3. **Livrables :** Un fichier Python (Exemple : model_pipeline.py) contenant les fonctions modularisées et un fichier main (Exemple : main.py) pour l'exécution de ses fonctions.

Tutoriel : Modularisation d'un Script Jupyter en Code Python Structuré - Modularisation du Code Machine Learning

Introduction

Ce tutoriel guide la transformation d'un script Jupyter fonctionnel en code modulaire pour simplifier son utilisation et son intégration dans un pipeline CI/CD. Nous allons extraire des fonctions bien définies pour chaque étape du processus machine learning, en respectant les bonnes pratiques de structuration de code.

I. Étape 01 : Identifier les étapes clés dans le script Jupyter

1) Analyser le script Jupyter :

Repérer les blocs de code correspondant aux étapes suivantes :

- Chargement et prétraitement des données.
- Entraînement du modèle.
- Évaluation des performances.
- Sauvegarde et chargement du modèle.

2) Planifier les fonctions nécessaires :

Chaque étape doit être encapsulée dans une fonction claire et réutilisable :

- **prepare_data() :** Charger et prétraiter les données.
- **train_model() :** Entraîner le modèle.
- **evaluate_model() :** Évaluer les performances.
- **save_model() :** Sauvegarder le modèle entraîné avec **joblib**.
- **load_model() :** Charger un modèle sauvegardé.
- **Etc ……**

II. Étape 02 : Copier le fichier data « nom_fichier.csv » vers WSL

1) Utiliser la commande « cp » pour copier le fichier dans votre répertoire WSL :

```
sirine@DESKTOP-4U94PKC:~/ml_project$ cp /mnt/c/Users/user/Desktop/MlOps/Churn_Modelling.csv ~/ml_project/
sirine@DESKTOP-4U94PKC:~/ml_project$ ls
Churn_Modelling.csv  model_pipeline.py  requirements.txt  test_environment.py  venv
```

III. Étape 03 : Créer un fichier Python pour le pipeline

```
sirine@DESKTOP-4U94PKC:~/ml_project$ nano model_pipeline.py
```

1) Créer le fichier contenant les fonctions (model_pipeline.py dans notre cas) dans votre terminal WSL en suivant les instructions suivantes :

    a. Ajouter les imports nécessaires

    b. Implémenter les fonctions

2) Créer le fichier principal main.py (La classe main) pour l'exécution des fonctions

3) Tester les différentes fonctions (prepare, train, evaluate, etc …)

Livrables

- ✓ model_pipeline.py : Contient toutes les fonctions modularisées.
- ✓ main.py : Permet d'exécuter les différentes étapes via des arguments CLI.

Bonnes pratiques

1) Vérifier les entrées et sorties de chaque fonction.
2) Tester chaque fonction individuellement avant d'intégrer.
3) Documenter les fonctions avec des docstrings.