README (Explication détaillée)

Étapes pour lancer les scripts

1. Cloner le projet :

La première étape consiste à cloner le projet sur votre machine locale. Utilisez la commande suivante (assurez-vous d’avoir git installé sur votre système) :

git clone <lien\_du\_dépôt>

2. Installer les dépendances :

Le projet utilise des bibliothèques Python externes telles que pandas, scikit-learn, Flask, et Dash. Celles-ci sont listées dans le fichier requirements.txt. Pour les installer, exécutez la commande suivante :

pip install -r requirements.txt

3. Lancer l'API Flask :

Une fois les dépendances installées, vous pouvez démarrer l’API Flask qui permet de faire des prédictions à partir de l'identifiant du client. Utilisez cette commande :

python app.py

Cela démarrera un serveur local accessible à l’adresse http://127.0.0.1:5000/predict. Vous pouvez ensuite envoyer des requêtes POST à cette URL avec les informations du client sous format JSON pour obtenir une prédiction ainsi qu'une probabilité associée.

4. Tester l'API :

Pour tester l’API, vous pouvez utiliser un outil comme Postman ou faire une requête Python en utilisant la bibliothèque requests. Voici un exemple de requête POST en Python :

import requests

client\_data = {"SK\_ID\_CURR": 123456}

response = requests.post('http://127.0.0.1:5000/predict', json=client\_data)

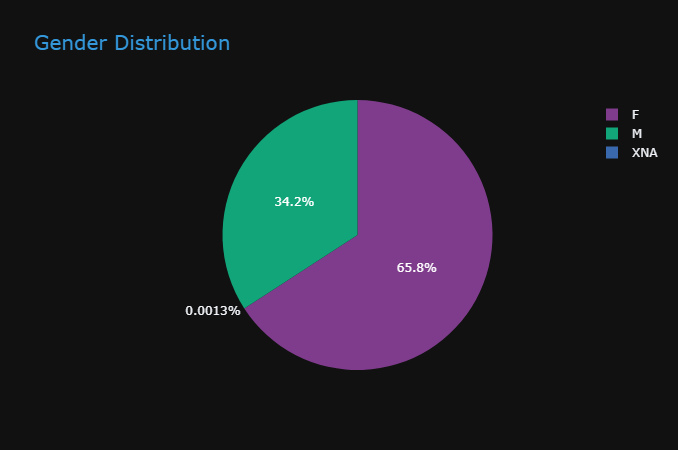
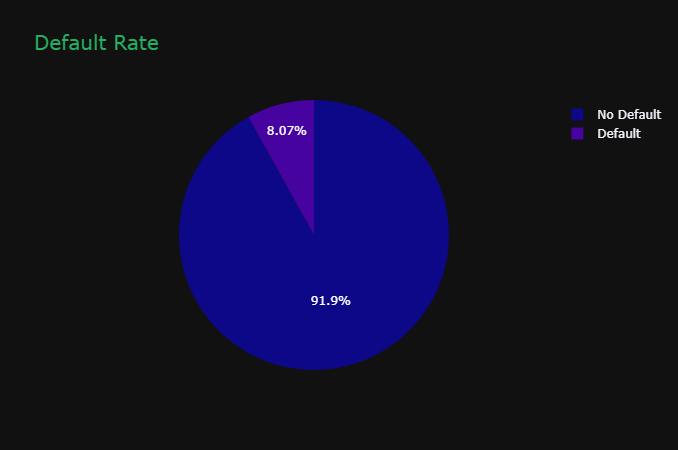
print(response.json())

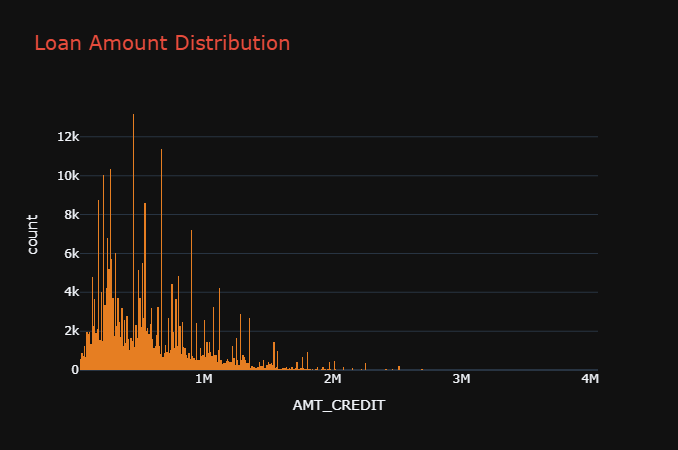
5. Lancer le tableau de bord interactif (Dash) :

Le tableau de bord interactif, construit avec Dash, permet d'explorer les données clients et de visualiser les prédictions du modèle. Pour démarrer le tableau de bord, exécutez :

python dashboard.py

Le tableau de bord sera accessible via votre navigateur à l'adresse http://127.0.0.1:8050. Il propose des visualisations interactives permettant aux conseillers de mieux comprendre les décisions du modèle et les informations associées à chaque client.



Problèmes rencontrés sur le jeu de données :

1. Valeurs manquantes :

Problème : Le dataset contient de nombreuses valeurs manquantes, notamment dans les colonnes catégoriques et numériques, ce qui peut affecter la qualité des prédictions si elles ne sont pas correctement traitées.

Solution :

Pour les colonnes numériques, nous avons rempli les valeurs manquantes avec la médiane, car cette méthode est moins sensible aux valeurs aberrantes que la moyenne.

Pour les colonnes catégoriques, nous avons utilisé le mode (la valeur la plus fréquente) afin de maintenir la cohérence des données.

2. Déséquilibre des classes :

Problème : La variable cible (indiquant si un client fait défaut ou non sur son crédit) est très déséquilibrée. Il y a beaucoup plus de clients non-défaillants que de défaillants, ce qui pourrait biaiser le modèle en faveur de la classe majoritaire.

Solution envisagée : Nous avons prévu d'utiliser SMOTE (Synthetic Minority Over-sampling Technique) pour générer des exemples synthétiques de la classe minoritaire. Cependant, pour cette première version rapide du modèle, nous avons simplement utilisé une division train-test sans équilibrage pour tester le flux général.

Nettoyage des données :

1. Encodage des variables catégoriques :

Certaines variables sont catégoriques, comme CODE\_GENDER et NAME\_CONTRACT\_TYPE. Pour les transformer en données utilisables par notre modèle de machine learning, nous avons utilisé plusieurs techniques d'encodage :

Pour les variables binaires (avec seulement deux catégories, comme CODE\_GENDER), nous avons utilisé LabelEncoder, qui transforme les catégories en 0 et 1.

Pour les variables avec plus de deux catégories, nous avons appliqué la technique de OneHotEncoding, qui crée des colonnes indicatrices pour chaque catégorie avec des valeurs binaires (0 ou 1).

2. Création de nouvelles features :

Nous avons ajouté des colonnes dérivées de certaines informations financières et comportementales :

CREDIT\_INCOME\_PERCENT : rapport entre le montant du crédit et le revenu total du client.

ANNUITY\_INCOME\_PERCENT : rapport entre l'annuité du crédit et le revenu total.

CREDIT\_TERM : durée du crédit, obtenue en divisant l'annuité par le montant total du crédit.

DAYS\_EMPLOYED\_PERCENT : pourcentage de jours travaillés par rapport à l'âge du client, calculé avec DAYS\_EMPLOYED et DAYS\_BIRTH.

Modélisation (du preprocessing à la prédiction) :

Prétraitement des données :

Après le nettoyage et l’enrichissement du dataset, nous avons normalisé les variables numériques à l'aide de StandardScaler. La normalisation est essentielle pour certains algorithmes comme la régression logistique afin de s'assurer que toutes les variables sont sur une échelle similaire, ce qui améliore la performance du modèle.

Entraînement du modèle :

Nous avons utilisé un modèle de régression logistique pour cette première version du scoring de crédit. Ce modèle est simple, interprétable et rapide à entraîner.

Pour entraîner le modèle, nous avons divisé les données en un ensemble d’entraînement (80 %) et un ensemble de validation (20 %). La régression logistique a été ajustée en utilisant des hyperparamètres standards avec la bibliothèque Scikit-learn.

Évaluation du modèle :

Pour évaluer la performance du modèle, nous avons calculé le score ROC-AUC sur les données de validation. Le ROC-AUC est une mesure utile pour évaluer la capacité d'un modèle à distinguer entre les classes positives (défaillants) et négatives (non-défaillants).

Résultat initial : Un score ROC-AUC de 0.74 a été obtenu, ce qui constitue un point de départ raisonnable pour un modèle .

Enregistrement du modèle et du scaler :

Pour rendre le modèle réutilisable dans une API Flask, nous avons enregistré le modèle et le scaler avec Pickle. Cela permet de charger ces objets plus tard sans avoir à réentraîner le modèle à chaque requête.