Note méthodologique :

Méthode d'entraînement

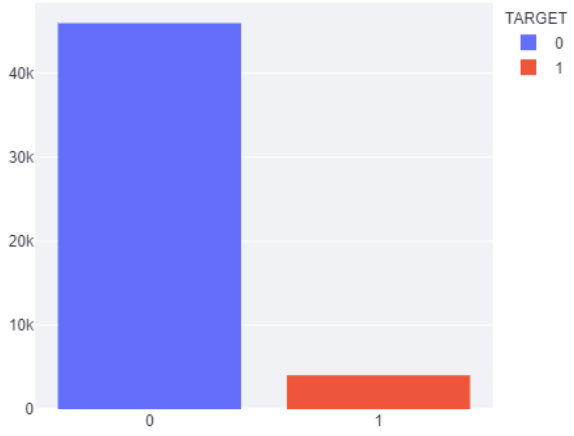
- **Prétraitement**

La première étape après avoir récupérer le jeu de données est le nettoyage de celui-ci. Nous avons enlevé les variables qui possédaient plus de 50% de données manquantes. Nous avons pu observer quelles valeurs aberrantes comme par exemple des personnes qui ont un emploi depuis plus de 100 ans.

Après le nettoyage, nous abordons l'étape de 'Feature Engineering', consistant à travailler les données existantes. Nous avons lié les différentes tables en agrégeant au niveau client ainsi qu'en reliant quelques variables entre elles. Par exemple nous avons créé des ratios concernant les revenus d'un client ainsi que le montant du prêt ou bien les échéances de celui-ci.

- **Déséquilibre des classes**

Le jeu de donnés est très déséquilibré de par la nature du problème étudié (risque de défaut de remboursement). Pour prendre en compte cet effet ainsi que l'aspect métier nous mettrons en place une fonction coût métier et nous utiliserons le module imblearn avec SMOTEENN pour aider à l'entraînement.



Nombre de clients par classe

- **SMOTEENN**

SMOTEENN est une méthode du module imblearn permettant d'aider les modèles de machine learning à apprendre malgré un déséquilibre de classe important. Il fonctionne en combinant l'undersampling (le fait de sélectionner un échantillon) ainsi que l'oversampling (celui de créer de nouveaux individus proches d'existants afin de créer de nouveaux individus pour la classe sous représentée). Le but de cette méthode est de produire un jeu de donné d'entraînement avec des classes équilibrées. La combinaison avec l'undersampling permet de nettoyer le bruit créé par les nouveaux individus.

- **Crossvalidation**

Enfin, pour entraîner le modèle nous utiliserons une cross validation avec 3 folds pour éviter d'overfitter sur le jeu d'entraînement.

- **Gridsearch**

Nous comparerons plusieurs modèles (lightgbm, régression logistique, random forest et xgboost) avec différents jeux de paramètres possibles pour déterminer la meilleure combinaison et le meilleur modèle qui sera dans notre cas XGBOOST.

Fonction coût métier et optimisation

- **Fonction coût**

Nous mettons en place une fonction coût personnalisée comparant les prédictions du modèle avec les classes réelles du jeu d'entraînement. Nous effectuons la différence en pondérant celle-ci lorsque le modèle fait une erreur de prédiction de type faux négatif (prédire un client sans défaut de paiement alors que sa classe réelle correspond à un défaut de paiement). De cette façon en cherchant à réduire cette fonction coût le modèle cherchera en priorité à éviter ces erreurs et donc à prédire le plus possible les défauts de paiement. La pondération sera à fixer avec le corps du métier. On peut la résumer par la formule suivant

diff = sum(abs(k*(y_true.iloc[i] - y_pred[i])))

avec k=1 si y_true.iloc[i] = 0 et k=ponderation sinon

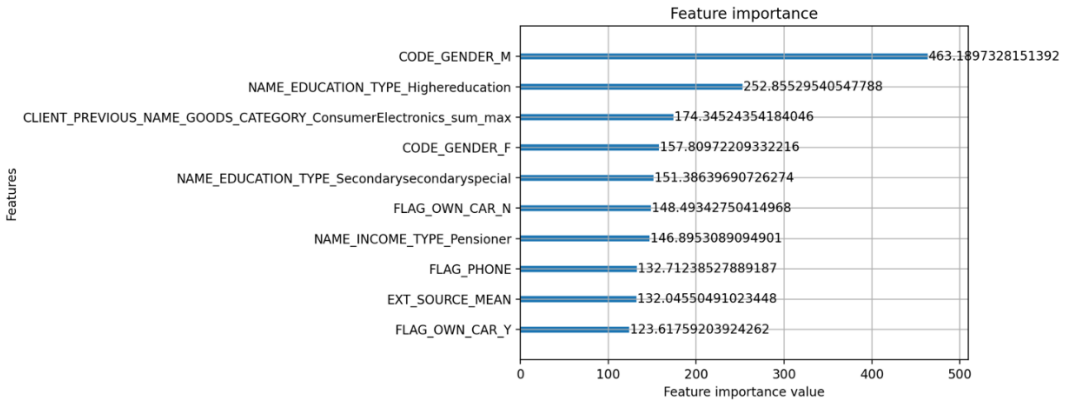
- **Optimisation**

Nous pouvons utiliser cette fonction coût afin d'optimiser la prédiction. De base le modèle renvoie une probabilité d'appartenance à la classe 0 (pas de défaut) et la classe 1 (défaut de paiement). Il attribue la classe en comparant ces prédictions et un threshold établi de base égal à 0.5. Afin d'améliorer le modèle nous pouvons jouer sur ce threshold en parcourant les probabilités d'appartenance à la classe 1. Pour chaque probabilité nous fixons le threshold comme égal à celle-ci et nous calculons la fonction coût. En prenant le résultat minimum de cette fonction nous pouvons fixer un threshold différent de celui de base. Si le client a un probabilité de défaut de paiement supérieur à celui-ci on considèrera qu'il est trop risqué de lui accorder un prêt.

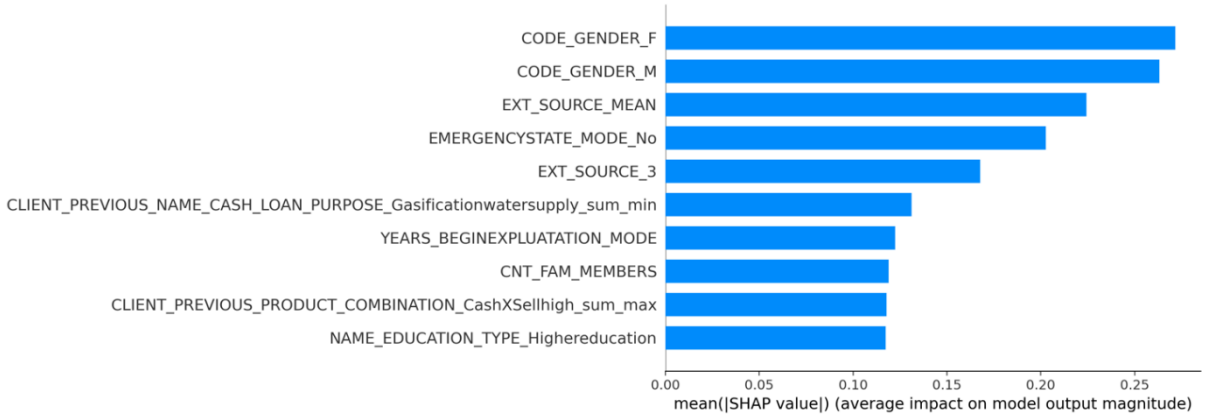
Interprétabilité du modèle

- **Globalement**

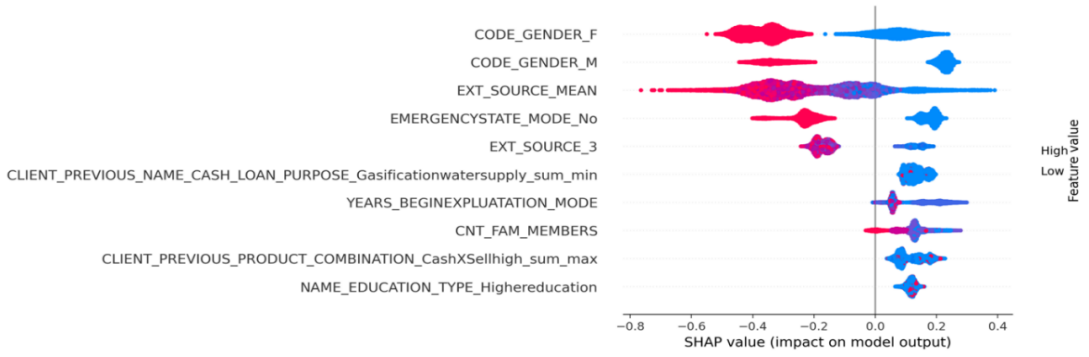
Pour interpréter le modèle nous pouvons regarder les features importance que nous propose XGBOOST. Par exemple nous pouvons regarder le gain du modèle choisi qui représente la moyenne de la réduction de la fonction coût quand la feature est utilisée au niveau d'un nœud.



Nous pouvons aussi regarder du côté des SHAP values. On peut déterminer l'effet des différentes variables d'une prédiction pour un modèle qui explique l'écart de cette prédiction par rapport à la valeur de base. Aussi, en moyennant les valeurs absolues des valeurs de Shap pour chaque variable, nous pouvons remonter à l'importance globale des variables.

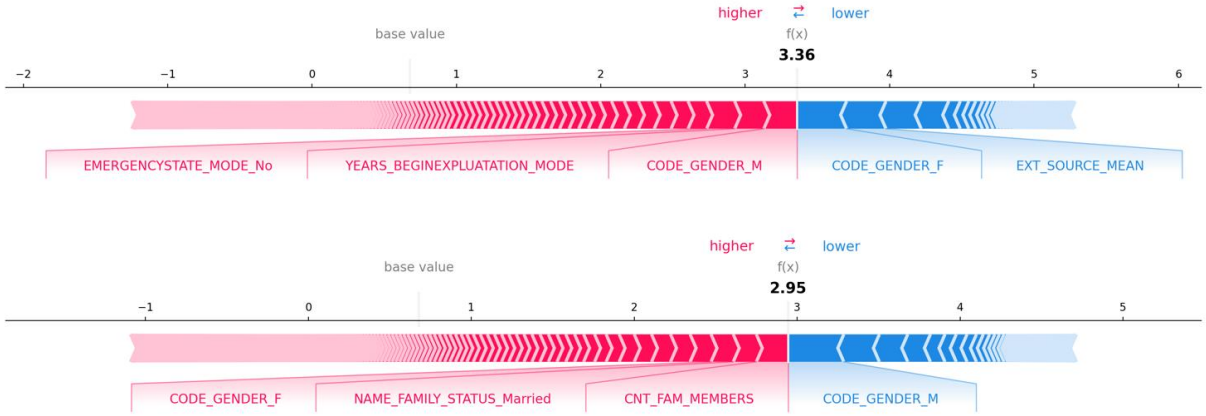


Comme les valeurs sont calculées pour chaque individu du dataset, il est possible de représenter chaque client par un point et ainsi avoir une information supplémentaire sur l'impact de la variable en fonction de sa valeur. Par exemple EXT_SOURCE_MEAN qui est une des variables les plus importantes, a un impact négatif quand la valeur de cette variable est élevé.



- **Localement**

Avec l'approche SHAP, une prédiction peut être écrite comme la somme des différents effets des variables ajoutée la valeur de base. La valeur de base étant la moyenne de toutes les prédictions. On peut voir sur deux clients différents que les variables 'importantes' sont différentes.



Limites et améliorations possibles

La première limite et amélioration disponible est le feature engineering mis en place. Nous traitons beaucoup de variables avec des transformations mathématiques ce qui génère beaucoup de variable. Nous allons créer des variables qui n'ont que peu de sens d'un point de vue métier. Pour corriger cela une simple concertation avec une équipe métier pour se concentrer sur les variables pertinentes permettrait potentiellement d'améliorer le modèle. De plus pour des raisons techniques nous avons échantillonné notre jeu de données provoquant potentiellement un overfitting sur certaines variables.

De la même façon la fonction de scoring mise en place peut être améliorée. Par exemple la pondération actuelle a été fixée arbitrairement sans avis métier. Par ailleurs nous pourrions pousser plus loin en analysant les mauvaises prédictions du modèle en cherchant à les expliquer et itérer afin de les réduire.

Enfin comme abordé plus tôt, le déséquilibre des classes inhérent au problème n'aide pas à la modélisation même si nous mettons déjà en place un moyen pour y remédier. Nous avons utilisé SMOTEENN d'imblearn mais nous pourrions tester d'utiliser un classweight sans équilibrer la distribution de la target et comparer quel modèle performe le mieux.