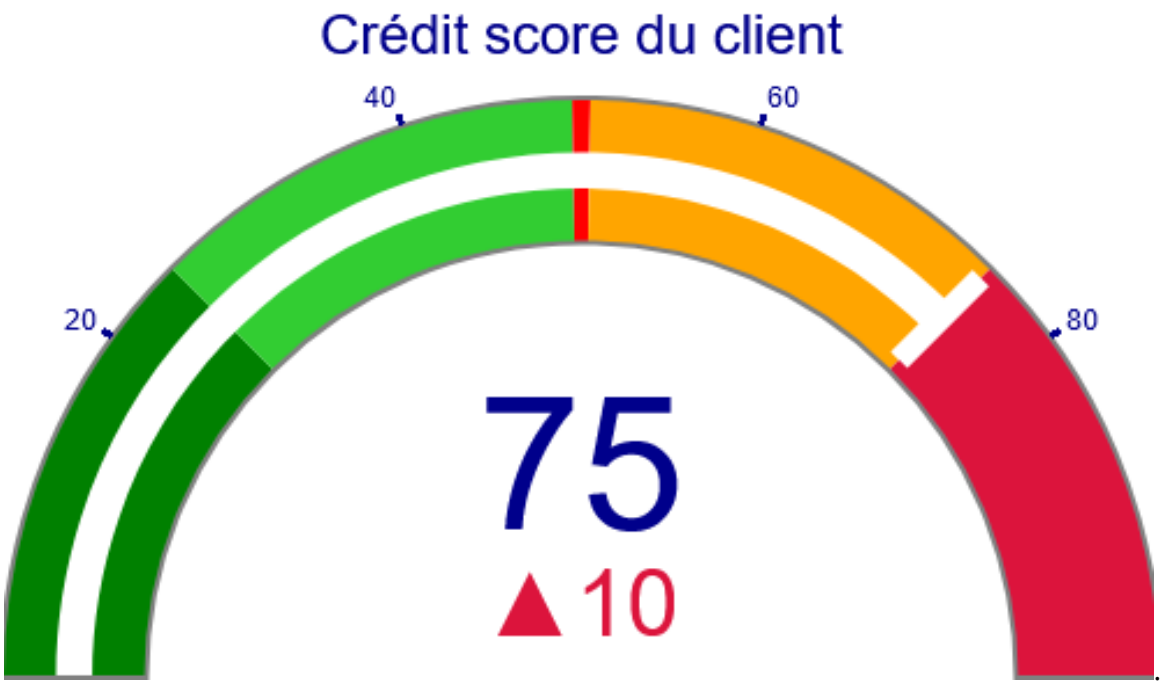
Note méthodologique :

Implémentation d'un modèle de scoring

M. Check KOUTAME
email : **chkoutam@yahoo.fr**



Juillet 2022

Table des matières

I. Introduction

La société financière d'offre de crédit à la consommation **Prêt à dépenser** souhaite mettre en œuvre un outil de **"scoring crédit"** pour calculer la probabilité qu'un client rembourse son crédit, puis classifie la demande en crédit **accordé ou refusé**. Elle souhaite donc développer un algorithme de classification en s'appuyant sur des sources de données variées (données comportementales, données provenant d'autres institutions financières, etc.).

Selon les chargés de relation client, les clients sont de plus en plus demandeurs **de transparence** vis-à-vis des décisions d'octroi de crédit. Cette demande de transparence des clients va tout à fait dans le sens des valeurs que l'entreprise veut incarner. De ce fait, Prêt à dépenser décide de développer un **Dashboard interactif** pour que les chargés de relation client puissent à la fois expliquer de façon la plus transparente possible les décisions d'octroi de crédit, mais également permettre à leurs clients de disposer de leurs informations personnelles et de les explorer facilement.

Notre **mission** sera donc de développer un **algorithme de classification** permettant de **refuser ou d'accorder un prêt** à un client en interagissant facilement avec un Dashboard. Le Dashbord sera le socle de transparence pour expliquer la décision de la société.

II. Présentation des données

Les données à notre disposition sont composées de huit fichiers au format csv téléchargeables sur kaggle. Les données contiennent des informations personnelles anonymes, mais aussi tout un historique de prêts acceptés ou refusés concernant les clients enregistrés à Homme Crédit Group et d'autres organismes financiers. Dans ces données, 307511 clients sont concernés avec 218 variables (figure 1).

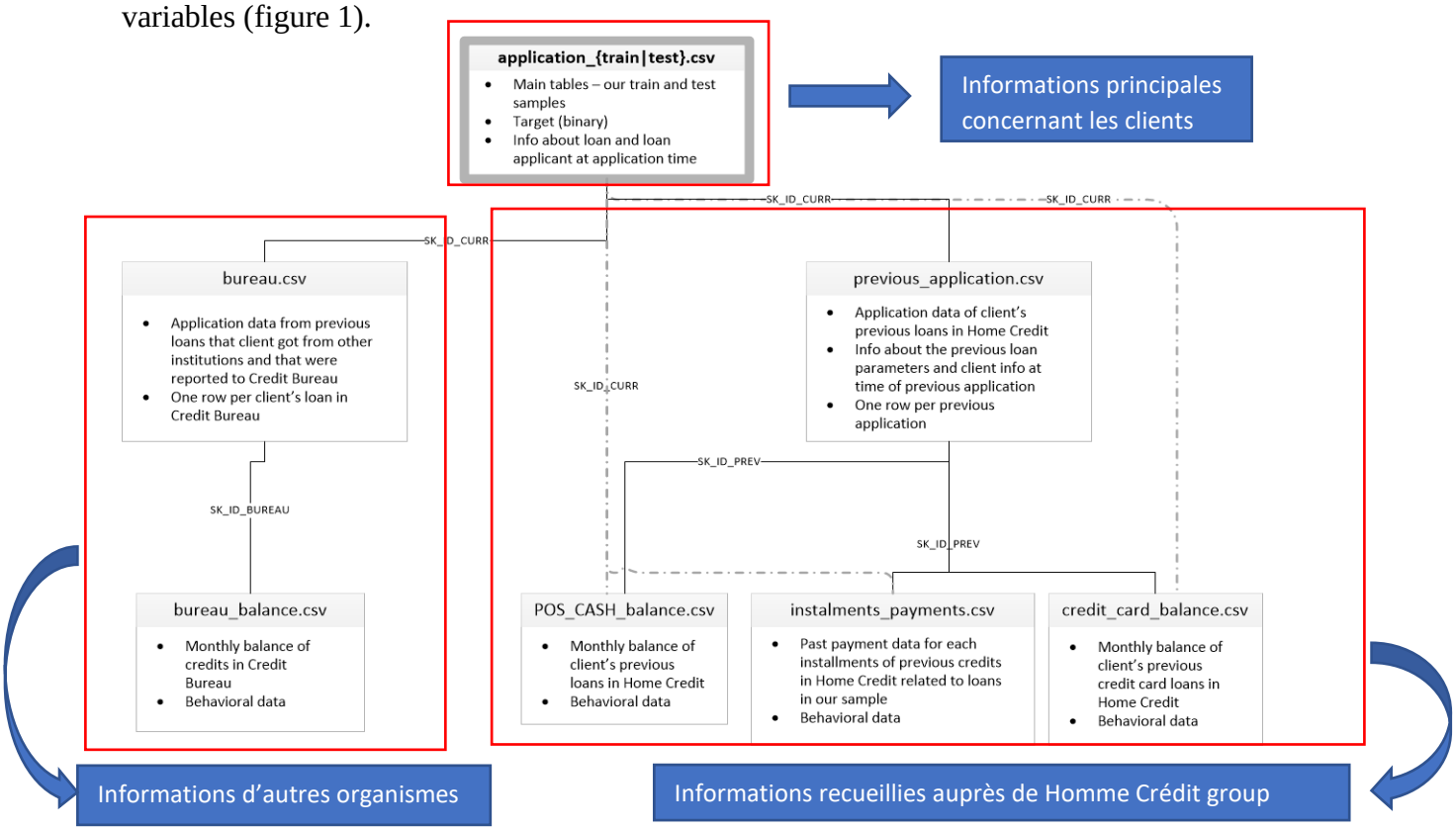
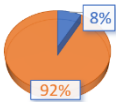


Figure 1 : Bases de données

III. Méthodologie d'entrainement du modèle

L'entrainement du modèle consiste à prédire 2 valeurs :



- 0 : non défaillant, c'est-à-dire que le client arrive à rembourser totalement son prêt
- 1 : défaillant, c'est-à-dire que le client n'arrive pas à rembourser son prêt.

Il s'agit d'une modélisation basée sur une classification binaire avec un score de probabilité de paiement. Cependant, il convient de bien noter que les variables cibles sont très déséquilibrées : 8% de défaillant et 92% de non-défaillant.

Plusieurs traitements ont été effectués sur les jeux de données pour arriver à l'entrainement final du modèle (figure 2).

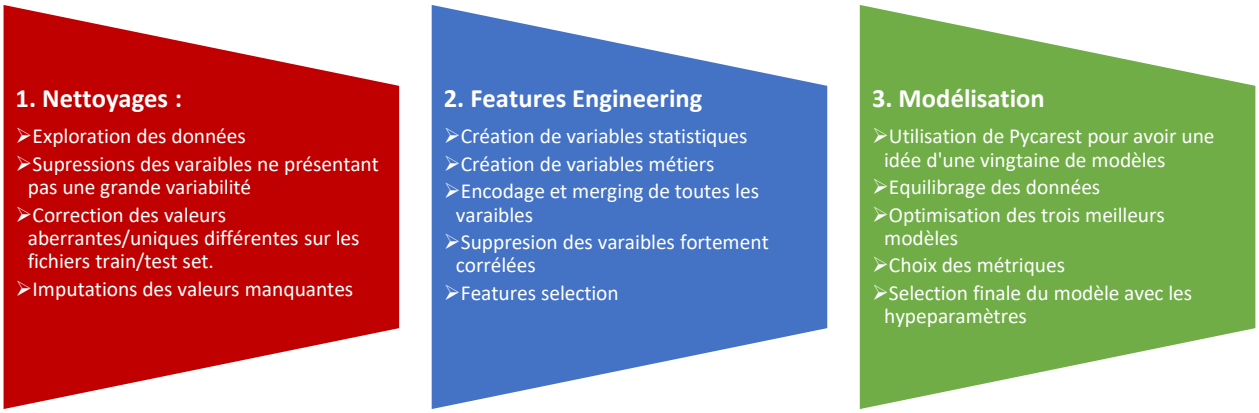


Figure 2 : Process Pycaret

La première étape consiste à faire une exploration des données pour ne garder que les variables qui présentent **une grande variabilité** entre les défaillants et les non-défaillants. Puis une transformation des types d'objet sera effectuée pour remplacer par exemple une variable de type Oui/Non (ou sexe masculin/féminin) en 0/1. **Les variables aberrantes seront corrigées**, puis toutes les variables ayant plus de 67 % de **valeurs manquantes** seront supprimées. Une fois cela achevé, nous effectuerons une **imputation des valeurs manquantes** restantes par la médiane pour les variables numériques et la mode pour les variables catégorielles.

La deuxième étape, appelée Features Engineering, consiste à créer de nouvelles variables sous deux formes :

- Une création manuelle de variables à partir des connaissances que nous avons du métier : en combinant des variables susceptibles d'améliorer notre modèle.
- Une création automatique/statistique de variables permettant d'ajouter la moyenne, le minimum, le maximum, etc…

La deuxième étape nous permet aussi **d'encoder** les variables catégorielles. Elles sont transformées en variables numériques qui sont les seules prises en compte par nos modèles. Pour ce faire, nous avons utilisé deux types d'encodage :

- **LabelEncoder** pour les variables binaires (Féminin/Masculin, Oui/Non) pour les transformer en (1/0)
- **OneHotEncoder** pour les variables contenant plusieurs valeurs différentes.

Une fois l'encodage effectué, tous les fichiers sont concaténés. Il convient donc de vérifier que chaque client n'est représenté qu'une seule fois dans le nouveau fichier.

La dernière sous-étape de la Feature Engineering consiste à diminuer le nombre de variables obtenues après la fusion des fichiers. En effet, nous disposons de 615 variables, un nombre trop élevé pour nos modèles et qui nécessite donc d'être réduit. Il s'agit donc de faire une Sélection de variables, encore appelée **Feature Selection**, qui consiste à suivre suite les opérations suivantes :

- Une étape de suppression de toutes les variables fortement corrélées.
- Une méthodologie d'Embedded ou intégrée qui permet de voir quelles sont les variables qui contribuent le plus à la précision du modèle
- Des méthodologies algorithmiques qui permettent de sélectionner les meilleures variables du dataset pour la création de modèles. Il s'agit de librairies Python comme Boruta, BorutaShop, RFECV, etc…

A l'issu des différents algorithmes utilisés, nous assemblons les différentes variables qui sont les plus sélectionnées dans tous les algorithmes. Au final, l'assemblage des variables les plus fréquentes nous permet d'extraire 106 variables du train_set et 105 variables du test_set. Ainsi, ce sont ces variables qui seront utilisées pour entrainer nos différents modèles et de les optimiser.

La troisième étape consiste à créer un modèle permettant au mieux de classifier nos différents clients. Pour ce faire, il convient de passer par plusieurs sous-étapes :

- **Utilisation de Pycaret** pour choisir les modèles les plus performants pour notre jeu de données. En effet, cette librairie permet de lancer une vingtaine de modèles de classification simultanément et de les comparer entre eux. Elle nous permet par la suite de sélectionner les modèles qui admettent les meilleurs métriques (AUC, précision, F1, etc…). Les modèles ensemblistes (LightGBM, XGBoost et CatBoost) nous donnent les meilleurs résultats. Le choix sera porté sur LightGBM pour sa rapidité par rapport au deux autres même si ces derniers sont plus performants.
- **Optimisation de notre algorithme LightGBM** :
  - En rééquilibrant la variable cible par la méthode SMOTE car nous avons vu que les données sont fortement déséquilibrées dans les datasets.
  - En utilisant plusieurs algorithmes d'optimisations bayésiennes qui sont bayes_opt du MIT, skopt de scikit-learn et optuna.
- **Choix de métriques** pour juger la performance de nos différents modèles, afin de choisir au final le modèle le mieux entrainé.

IV. La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation

1. Métrique d'évaluation et fonction coût

Pour juger du meilleur modèle à utiliser, il convient de savoir quelle est la métrique utilisée, et quelle est la plus adaptée en fonction de notre problématique. Dans notre cas, nous souhaitons limiter la perte d'argent pour la société financière. Nous nous appuyons sur la matrice de confusion pour une meilleure prédiction qui maximise le gain obtenu par validation ou non des demandes de prêts bancaires. Il convient donc :

- **De ne pas prédire « défaillant » un client non défaillant** : minimiser les faux positifs (erreur de type I : prédire défaillant alors que le client est non-défaillant). Dans ce cas, la société perd seulement les intérêts qu'elle aurait pu gagner. Il s'agit donc de maximiser la métrique Précision

- **De ne pas prédire un client non-défaillant s'il est défaillant :** minimiser le nombre de faux négatifs (erreur de type II : prédire non-défaillant alors que le client est défaillant) : Dans ce cas la perte est maximale car la société perd toute la somme prêtée au client. Il s'agit donc de maximiser les métriques Recall ou FBeta 10.

.

Réelles	+	**TP** : Vrais Positif	**FN** : Faux Négatifs
	-	**FP** : Faux positifs	**TN** : Vrais Négatifs
		+	-
		Prédictions	

$$Précision = \frac{TP}{TP+FP} \quad Recall = \frac{TP}{TP+FN}$$

$$\mathbf{F_\beta\text{-score}} = (1+\beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

- $\beta \geq 1$, on accorde **plus d'importance au Recall** (autrement dit aux faux négatifs).
- $\beta \leq 1$, on accorde **plus d'importance à la Precision** (autrement dit aux faux positifs).
- $\beta = 1$, on retrouve le F1-score, qui accorde autant d'importance à la précision qu'au Recall

Une fonction coût a été créée pour tenir compte de l'importance relative de chaque erreur.

$$J = TP * TP_{value} + TN * TN_{value} + FP * FP_{value} + FN * FN_{value}$$

TP, TN, FP et FN étant respectivement le nombre de True Postif, le nombre de True Negatif, le nombre de Faux Positif et le nombre de False Négatif. Les values représentent les paramètres que l'on peut modifier pour se rapprocher de ce qui nous arrange.

TP_value	0
FN_value	-10
TN_value	1
FP_value	0

Des coefficients ont été posés arbitrairement :

Ces valeurs de coefficients signifient que les Faux Négatifs engendrent des pertes 10 fois plus importantes que les gains des Vrai Négatifs. Ces poids ont été fixés arbitrairement et il est tout à fait envisageable de les modifier à la convenance de l'optique métier.

Dans le choix des différentes métriques utilisées, nous avons choisi la **F-score F10** qui donne de meilleurs résultats par rapport aux autres.

2. Choix du modèle et Algorithme d'optimisation

Le choix du modèle a été effectué à partir de Pycaret. Puis les trois modèles ensemblistes les plus performants ont été retenus, et principalement le modèle LightGBM qui est plus rapide que les deux autres (CatBoost et XGBoost). Pour améliorer ce modèle, nous avons d'abord équilibré les données cibles, puis optimisé les paramètres du modèle en fonction de la métrique F10 retenue.

a. Rééquilibrage de la variable cible

Pour une meilleure modélisation, l'ensemble des data_set a été modifié pour permettre une meilleure homogénéisation de la variable cible. Pour ce faire deux méthodes ont été testées :

- **Le sous-échantillonnage :** Parmi les individus majoritaires, on en retire une partie afin d'accorder plus d'importance aux individus minoritaires. Cette approche permet de diminuer la redondance des informations apportées par le grand nombre d'individus majoritaires.
- **Le sur-échantillonnage :** Le nombre d'individus minoritaires est augmenté pour qu'ils aient plus d'importance lors de la modélisation. Différentes solutions sont possibles, comme le "clonage" aléatoire ou le SMOTE.

Dans notre modélisation, nous avons utilisé d'une part SMOTE ainsi que ses dérivés BorderLineSMOTE et ADASYN, et d'autre part l'hyperparamètre class_weight du modèle LightGBM.

b. Choix du modèle et hyperparamétrisations :

L'optimisation s'est effectuée en utilisant la technique Bayésienne ainsi que 3 de ces librairies (bayes_opt du MIT, skopt de scikit-learn et optuna). L'optimisation bayésienne fonctionne en construisant une distribution postérieure de fonctions (processus gaussien) qui décrit au mieux la fonction que l'on veut optimiser. Au fur et à mesure que le nombre d'observations augmente, la distribution postérieure s'améliore, et l'algorithme devient plus certain des régions de l'espace des paramètres qui méritent d'être explorées et de celles qui ne le méritent pas.

L'optimisation a été effectuée sur différentes métriques (Roc Auc, PR Auc, F10, Recall et la métrique métier) pour différents jeux de données (rééquilibrés avec smote, hyperparamètre class_weight de Lightgbm ou non équilibrés, standardisés ou non…). Le but est de minimiser le nombre de faux négatifs tout en prédisant le plus de vrais positifs possibles et en limitant le nombre de faux positifs. Le modèle LightGBM avec les paramètres de base sert de comparatif.

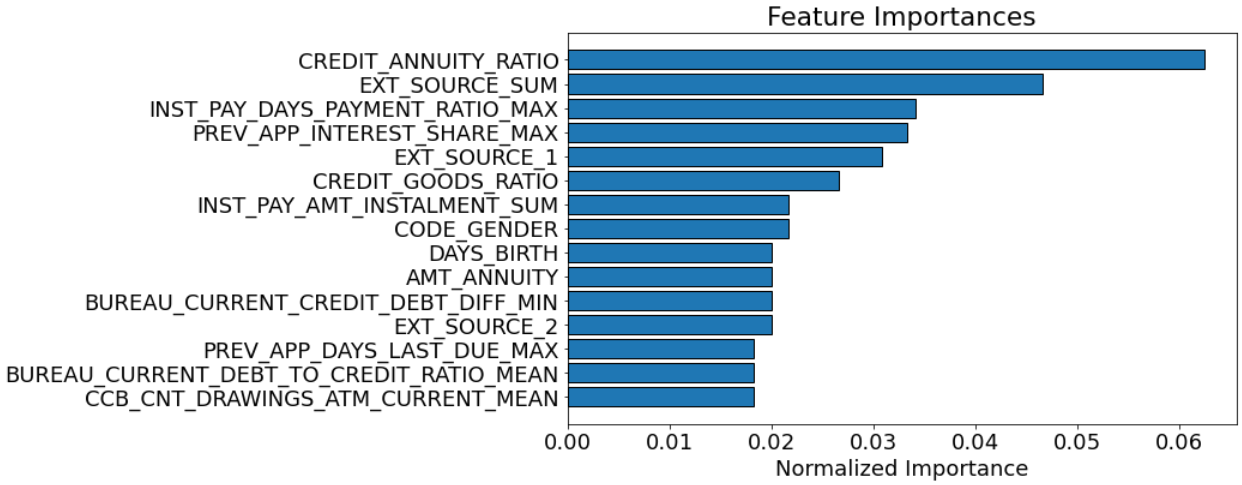
c. Choix du modèle et hyperparamétrisations :

Le modèle le plus performant est le modèle optimisé avec **la méthode bayésienne Optuna**, avec un rééquilibrage interne (**hyperparamètre class_weight='balanced'**) et la **métrique F10**. Il détecte le moins de faux négatifs, le plus de vrais positifs mais un taux plus élevé de faux positifs. Un compromis est à faire entre le taux de faux négatifs et le taux de faux positifs, une augmentation de l'un entraînant une diminution de l'autre.

V. L'interprétabilité globale et locale du modèle

L'importance des variables peut être affichée en utilisant le modèle lui-même pendant sa prédiction

Ainsi, après normalisation, nous pouvons comparer l'importance relative de chacune des variables et par un simple tri, afficher les 15 premières variables les plus importantes.



Parmi ces variables nous retrouvons les variables les plus corrélées avec la variable cible détectée lors de l'EDA :

- **Les informations bancaires** en particulier CREDIT_ANNUITY_RATIO (ratio du montant du crédit du prêt sur l'annuité de prêt), CREDIT_GOODS_RATIO (ratio du montant du prêt sur le prix réel du bien),

BUREAU_CURRENT_CREDIT_DEBT_TO_CREDIT_RATIO_MEAN (le cumul des autres prêts en cours) …,

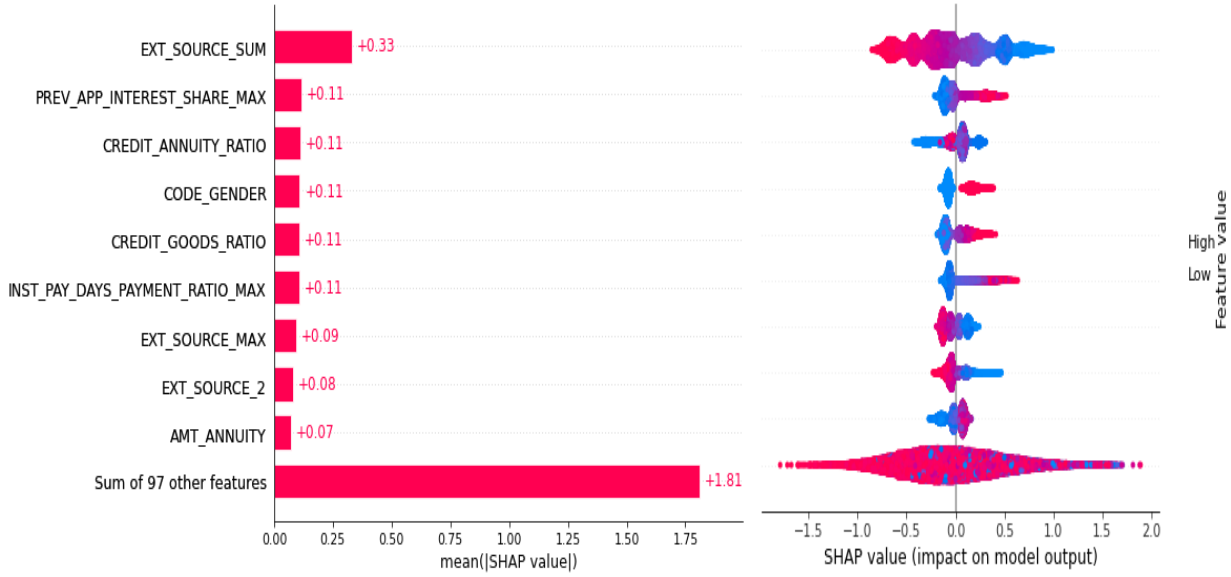
- **Les données externes** : EXT_SOURCE_SUM, EXT_SOURCE_1 et EXT_SOURCE_2,
- **Les informations personnelles** : CODE_GENDER (sexe du client), DAYS_BORTH (âge du client).

Interprétabilité : SHAP

Pour l'interprétabilité, nous avons utilisé SHAP.

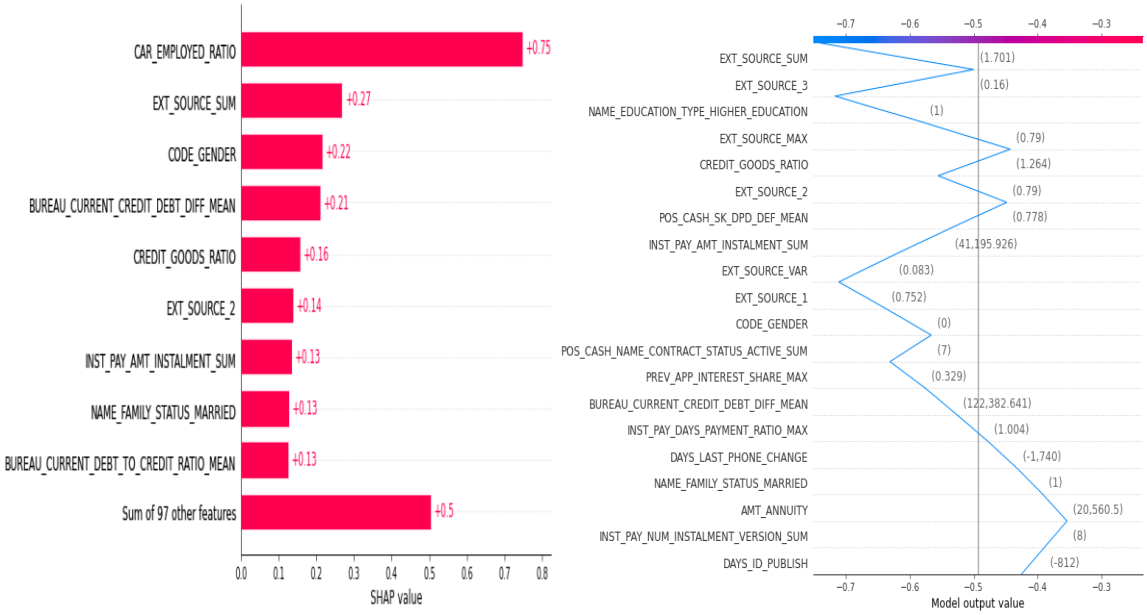
- La méthode SHAP (SHapley Additive exPlanations) consiste à calculer la valeur de Shapley pour toutes les variables de tous les individus c'est-à-dire la moyenne de l'impact d'une variable (sur la sortie, donc la prédiction) pour toutes les combinaisons de variables possibles.
- La somme des effets de chaque variable expliquera la prédiction

Pour l'interprétabilité, les variables globales les plus importantes sont obtenues en faisant une moyenne de l'ensemble des variables importantes permettant une meilleure prédiction.



Sur l'image de gauche, l'importance des variables est calculée en moyennant la valeur absolue des valeurs de Shap. Sur la droite, les valeurs de Shap sont représentées pour chaque variable dans leur ordre d'importance. Chaque point représente une valeur de Shap (pour un exemple), les points rouges représentent des valeurs élevées de la variable et les points bleus des valeurs basses de la variable.

Pour l'interprétabilité locale, on utilise les mêmes graphiques, mais aussi avec la décision ou non de donner un prêt pour un seul client.



A gauche : les variables les plus importantes pour le **client 100001**. A droite : Impact de toutes les variables sur la prédiction pour le **client 100001**

VI. Les limites et les améliorations possibles

Pour classifier défaillant ou non-défaillant à partir des 8 fichiers fournis par Prêt à dépenser, de nombreuses techniques de Machine Learning ont été nécessaires. Il faut d'abord comprendre qu'il s'agit d'une classification binaire, suivie d'un certain nombre de méthodologies à suivre pour mener à bien l'étude portée par la société Prêt à Porter : rééquilibrage des classes, création de nouvelles variables explicables, sélection des variables pour rendre le modèle moins complexe, choix de métriques adaptées à notre problématique métier, réflexion sur le compromis taux de faux négatifs et taux de faux positifs et sur le réglage du seuil de décision. Les experts métiers pourraient nous aider à créer une métrique bancaire plus efficace et adaptée et pourraient nous donner leur avis sur l'intérêt des nouvelles variables créées et même nous indiquer de nouvelles variables. Ainsi, pendant la « Feature Engineering », ces experts pourront nous aider à déterminer quelles compositions de variables sont les plus pertinentes au vu de leur expérience dans leur choix d'accorder un prêt ou non à un client donné. Une explication des données externes serait un plus puisqu'il est difficile d'être transparent en utilisant ces variables importantes pour le modèle mais inexplicables pour le client. Le Dashboard pourra également être évalué par le client et les remarques prises en compte, des améliorations techniques pourront être apportées (notamment sur des différents graphiques affichés, choisir ceux qui sont les plus parlant pour le client, …etc).

Par ailleurs, nous pourrions envisager de diviser le jeu de données en n partitions puis d'entrainer un modèle sur chacune d'entre elles, puis d'utiliser un « blender » sur les n modèles obtenus. Un « blender » consiste à agréger les prédictions de chacun des classificateurs et de prédire la classe qui obtient le plus grand nombre de votes. Ce classificateur par vote majoritaire obtient souvent une exactitude plus élevée que le meilleur classificateur de l'ensemble.

Nous pourrions aussi utiliser un serveur plus conséquent en taille mémoire et en CPU, ce qui permettrait d'afficher plus de contenu sur le Dashboard. Eventuellement, l'ajout de GPU permettrait de tester les modèles « CatBoost » et « XGBoost » qui sont très prometteurs, notamment pour le traitement des variables catégorielles.