**Note**

**Méthodologique**



**Projet 7 Parcours Data Scientist**

**Dossier versioning :**

[**https://github.com/OG89/P7-OpenClassrooms/tree/main**](https://github.com/OG89/P7-OpenClassrooms/tree/main)

**Oumar GUEYE Juillet 2022**

***OpenClassrooms***

**1 - Prétraitement des données**

**Cette partie de preprocessing est décomposée en 2 sous-parties.**

* La première partie est une brève inspection et description des données...

Pour chaque variable numérique des jeux de données, il y a des informations concernant le pourcentage de données manquantes, la moyenne, la médiane, l'écart-type etc...

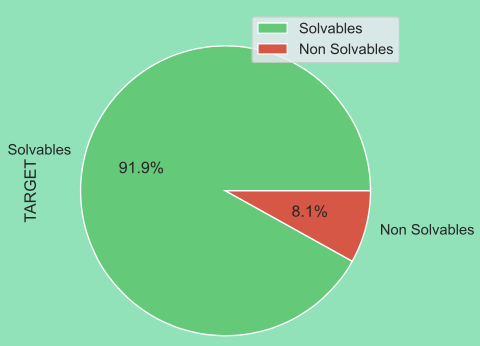
Et pour chaque variable catégorielle, il y a des informations concernant les données manquantes et pour chaque valeur prise par la variable il y a le nombre d'occurrences.

* La deuxième partie s'intéresse au nettoyage et à l'exploration de notre donnée.

Les individus dont les valeurs des variables catégorielles sont non communes aux jeux test et d'entrainement sont supprimés ainsi que les individus dont les valeurs trop peu fréquentes. Les valeurs aberrantes de certaines variables numériques sont remplacées par 'NaN' pour une imputation ultérieure.

L’inspection et la description des données nous montrent que :

* le fichier ‘**application\_test.csv’** renseigne exactement les mêmes informations que le fichier **'application\_train.csv'**, mais il détermine en plus **la probabilité de défaut de paiement pour ces nouveaux emprunts.**
* les autres **fichiers.csv**, ils nous informent sur les emprunts antérieurs qui ont été accordés par cette banque ou d'autres institutions financières.
* Le nom de la colonne dont on cherche à prédire la valeur est la variable "**TARGET**". Elle prend soit la valeur 0 quand le prêt a été remboursé sans soucis, soit la valeur 1 quand il y a eu un défaut de payement.



**Observations** : On observe que 92 % des prêts ont été remboursés et 8 % des clients ont été non solvables.

Ensuite, nous avons procédé respectivement au :

# Nettoyage des données

* **Création de dataset étendu**

Les jeux de données d'entrainement et de test contiennent les principales informations des clients nous allons les concaténer afin traiter les features de manière conjointe.

Les individus dont les valeurs des variables catégorielles sont non communes aux jeux test et d'entrainement sont supprimés ainsi que les individus dont les valeurs trop peu fréquentes. Les valeurs aberrantes de certaines variables numériques sont remplacées par 'NaN' pour une imputation ultérieure.

# Feature Engineering

## **Feature Engineering "automatisé"**

Tout d'abord, nous avons initialisé un ***EntitySet***, chargé le DataFrame des applications en tant qu'entité, changé le typage de certaines variables de type 'numeric' qui devraient être de type 'boolean'.

## **Feature engineering "Manuelle"**

Dans cette étape de 'Feature Engineering' nous allons créer de nouvelles features pertinentes issues de celles existantes pour aider le modèle.

* Le ratio temps travaillé/âge
* Le ratio crédit/revenu
* Le ratio revenu/annuité
* Le ratio revenu/annuité/âge
* Le ratio crédit/annuité
* Le ratio crédit/annuité/âge

# Traitement des valeurs manquantes

Dans cette partie, nous avons évalué le pourcentage des valeurs manquantes puis supprimer les variables qui ont plus de 90% de valeurs manquantes.

# Encodage des variables catégorielles

Les variables catégorielles doivent être encodés pour être utilisables par les modèles.

* *Les variables cycliques comme les jours de la semaine sont encodées en deux dimensions.*
* *Les variables contenant uniquement 2 classes sont assignées à des 0 ou et des 1.*
* *Pour les variables à plus de 2 classes, un OneHotEncoder est appliqué.*

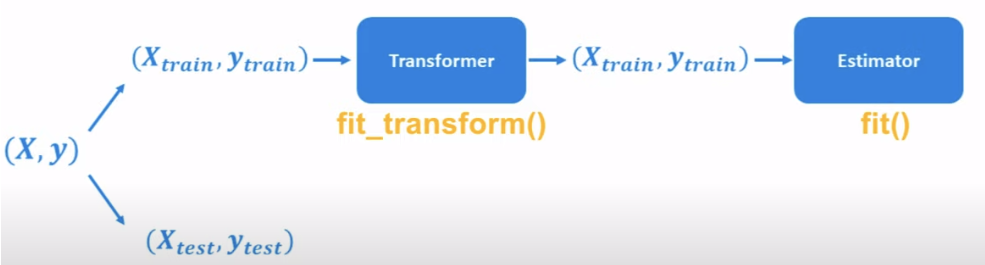
**2 - Modélisation**

L’entrainement d’un modèle nécessite une étape transitoire entre le preprocessing et la modélisation. Ce modèle de prédiction peut être représenté́ par une fonction qui prend des données en entrée et une décision en sortie. Dans le cas d’*apprentissage supervisé* où nous souhaitons expliquer une variable de sortie, les données sont classiquement subdivisées en 2 parties :

- Le jeu de données d’apprentissage correspond à l’échantillon principal où sont appliquées les méthodes, sur lesquelles les algorithmes apprennent. Il sert à ajuster le modèle, et représenté dans notre cas 80% des données.

- Le jeu de données test est utilisé́ pour évaluer le modèle optimal (au sens du résultat de la validation croisée). Il n’a donc pas été utilisé pour l’apprentissage, ce qui fait que le modèle sélectionné́ est indépendant de cet échantillon test. L’idée est de simuler la réception de nouvelles données (entrée) afin de prédire la variable à expliquer à partir du modèle final et de les comparer aux « vraies » valeurs de la variable à expliquer. Cet échantillon permet d’évaluer objectivement l’erreur réelle. L’échantillon test représente donc 20% des données.

* **Pour entrainer notre modèle :**



* **Pour évaluer notre modèle :**

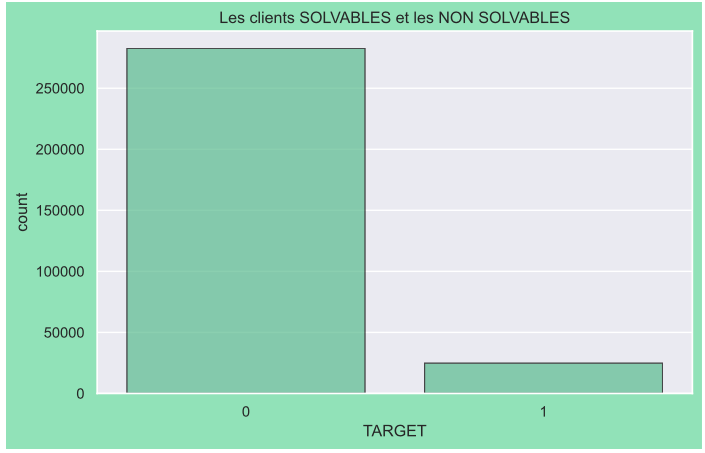
Une image contenant texte

Description générée automatiquement

Ce découpage de données dans un projet de *Machine Learning* est une étape très importante qu’il ne faut pas négliger. La fonction *train\_test\_split ()* de Scikit-Learn permet d’obtenir facilement une répartition aléatoire des individus en base d’apprentissage et de test. Il existe en effet un risque de surévaluer du modèle appelé : **over-fitting** ou le contraire appelé : **under-fitting**.

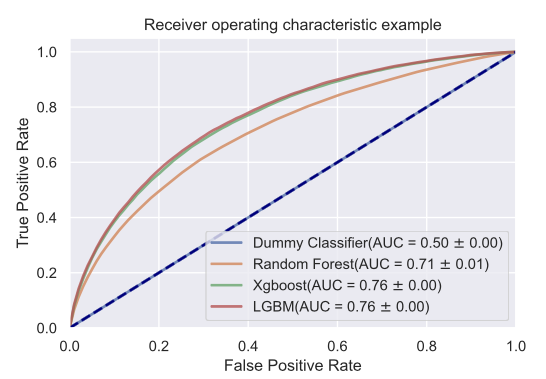
Le problème de *Machine Learning* à résoudre est un problème de *classification binaire*, il faut s’assurer que chaque côté́ contient une proportion raisonnable des classes 0 et 1.

L’Analyse Exploratoire des données a permis d’identifier un fort déséquilibre entre la précision trouvée entre la classe 0 et la classe 1. L'échantillon de travail est déséquilibré́, 92% en classe 0 vs 8% en classe 1. Le traitement *Oversampling* (ou suréchantillonnage en français) permet d’ajuster la distribution de classe de manière à avoir une répartition plus égalitaire.



## 2.1 : Liste des classifieurs utilisés pour construire le modèle de scoring

Toute la force du *Machine Learning* réside dans la diversité́ des approches utilisés. Plus le nombre de modèles testés est élevé́, plus il sera possible de trouver le meilleur algorithme permettant de répondre à la problématique, ici le modèle de scoring. Contrairement aux modèles statistiques classiques qui doivent vérifier certaines hypothèses sur la distribution des données, le *Machine Learning* a de fortes capacités prédictives compte tenu de son approche non paramétrique et de la faculté́ de ses algorithmes à apprendre, à partir des données. Ainsi, nous avons testé trois algorithmes RandomForestClassifier () XGBClassifier () et LGBMClassifier (). Une Baseline a également été fixée par *DummyClassifier ()*, un modèle statistique classique.



#### Le meilleur modèle est le LGBMClassifier (Light Gradient Boosting Machine)

* LightGBM est un framework de renforcement de gradient basé sur des arbres de décision pour augmenter l'efficacité du modèle et réduire l'utilisation de la mémoire.
* Il utilise deux nouvelles techniques : l'échantillonnage unilatéral basé sur le gradient et le groupement exclusif de fonctionnalités (EFB) qui répond aux limites de l'algorithme basé sur l'histogramme qui est principalement utilisé dans tous les cadres GBDT (Gradient Boosting Decision Tree).

## 2.2 Optimisation des hyperparamètres

Nous avons utilisé le packages hyperopt pour la séléction d'hyperparamètres. La mise en œuvre de *LightGBM* est facile, la seule chose compliquée est le réglage des hyperparamètres. *LightGBM* couvre plus de 100 hyperparamètres. Dans le contexte du projet l’idée est de pouvoir optimiser quelques hyperparamètres via Hyperopt.

En amont, il est nécessaire d’identifier des hyperparamètres pouvant avoir un impact dans l’amélioration de la métrique d’évaluation.

**● colsample\_bytree** : fraction d'observations à sélectionner pour chaque arbre  
**● learning\_rate** : détermine l'impact de chaque arbre sur le résultat final.  
● **max\_depth** : profondeur maximale d'un arbre.  
● **n\_estimators** : nombre d'arbres séquentiels.  
**● colsample\_by\_tree** : fraction d'observations à sélectionner pour chaque arbre.

Une image contenant texte

Description générée automatiquement

Le meilleur seuil de solvabilité est de 0.25, autrement dit que, tous les clients dont les valeurs de prédictions sont inférieures à 0.25 ne sont pas solvables, c’est à dire que ces clients ne pourront rembourser leurs prêts bancaires.

On trouve ci-dessous, graphiquement, que le meilleur seuil de solvabilité est de 0.1 contrairement à celui trouvé par l'optimisation via hyperopt qui est de 0.25.

# 

Le gain obtenu avec un seuil de 0.1 est plus conséquent que pour un seuil par défaut de 0.25 fixé par LGBM.

# 2.3 Fonction coût

L’entreprise "Prêt à dépenser" lutte contre les défauts de paiement des clients, les pertes financières ne sont en effet pas souhaitables (frais de recouvrement, pertes, ...).

Le modèle ne permettra pas d’éviter totalement ce risque, à titre d’exemple une erreur de prédiction aura pour conséquence soit un défaut de paiement du client, soit un refus de crédit à un client qui pourrait rembourser sa dette sans aucune défaillance. Les erreurs de prédiction doivent être minimisées, dans cette logique une fonction coût ayant pour objectif de pénaliser les Faux Positifs et les Faux Negatifs a été implémentée.

***Terminologie*** :

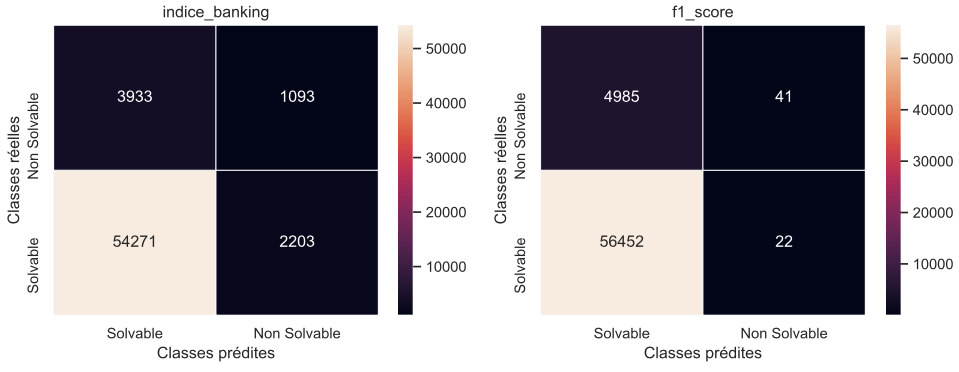
* ***-  FP (Faux Positifs)*** : les cas où la prédiction est positive, mais où la valeur réelle est négative. Perte d'opportunité́ si le crédit client est refusé à tort, alors qu'il aurait été en mesure d'être remboursé.
* -  ***FN (Faux Negatifs)*** : les cas où la prédiction est négative, mais où la valeur réelle est positive. Perte réelle si le crédit client accepté se transforme en défaut de paiement.
* -  ***TP (Vrais Positifs)*** : les cas d’acceptation, le crédit client sera remboursé.
* ***-  TN (Vrais Negatifs)*** : les cas de refus, le crédit client ne pourra pas être remboursé.

Une image contenant table

Description générée automatiquement

Prédiction sur les données de validation pour le modèle optimisé avec la métrique banking

et avec la métrique technique f1\_score



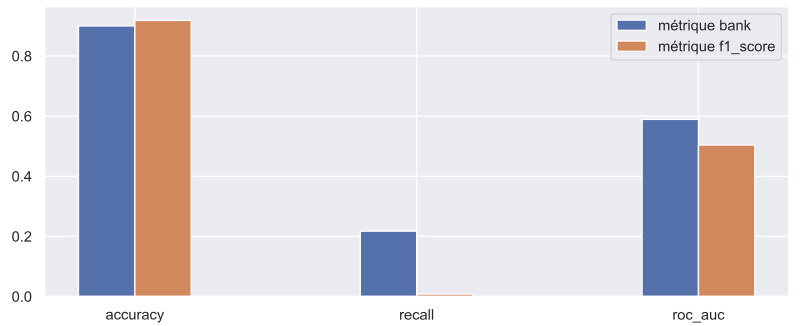
La comparaison des deux matrices de confusion pour la métrique spécifique métier et la métrique technique, montre qu'il y a moins de clients 'True negatifs' mais également moins de 'False negatifs'. C'est à dire que plus de clients ne se sont pas vus accordés de prêts alors qu'ils étaient solvables, mais à l'inverse moins de clients ont été prédits solvables alors qu'ils ne l'étaient pas. Ce que l'on a essayé par la construction d'une métrique métier est donc vérifié. Le bilan est in fine supérieur pour la métrique métier, à condition d'avoir posé les bons poids au départ.

Ainsi, les pertes d'un crédit en raison d'une mauvaise classification dépendent des probabilités ***Faux Positifs*** et ***Faux Negatifs***. L'idée est d'éviter les clients avec un fort risque de défaut. Il est donc nécessaire de pénaliser les ***FP et FN*** cités précédemment. Pour réduire ce risque de perte financière, il faut maximiser deux critères ***Recall*** et ***Precision***.

Une image contenant texte

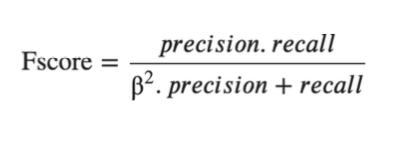
Description générée automatiquement

## Précision, Rappel et Roc\_auc



L'optimisation par métrique bancaire fait perdre un peu de précision au modèle. En revanche le gain en rappel est très important, ce qui était ou à fait prédictible puisque nous avons fortement pénalisé les 'False Negatives'. Le gain est in fine plus important pour l'optimisation via la métrique bancaire comme nous l'avons vu ci-dessus.

Fonction d'optimisation Recall et Precision avec une importance plus forte pour le critère Precision :



**L'application de cette métrique métier passe par la quantification de l'importance relative entre *Recall* et *Precision*, à savoir Beta (β).**

Cela revient à estimer le coût moyen d'un défaut, et le coût d'opportunité́ d'un client refusé par erreur. Cette connaissance métier n'est pas évoquée à ce stade du projet, nous allons donc l'estimer. Cette hypothèse pourra bien entendu être modifiée avec un interlocuteur métier.

● Défaut de paiement 30% du montant du crédit en pertes et autres recouvrements.

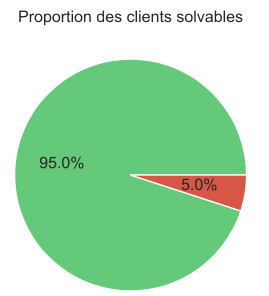
● 10% de chance d'obtenir un crédit pour un client lambda qui souhaite emprunter.

***Compréhension du score en pourcentage sur le risque de défaillance d’un client :***

**0%** *Risque faible de défaillance* --------------//----------------- **100%** *Risque fort de défaillance*

## Prédiction sur les données test non étiquetées

L'objectif étant de prédire la solvabilité des clients souhaitant emprunter, nous utilisons le modèle déterminé pour prédire leur solvabilité.



# 2.4 Les features les plus importants avec lime

**Interprétabilité du modèle**

La réponse au besoin d’Interprétabilité est prépondérante, le contexte de prédiction n’est pas uniquement appliqué à des experts de la data science mais au contraire à des experts du crédit. Un chargé de clientèle doit pouvoir utiliser le modèle via l’application mise à disposition, en face à face avec son client, dans le but de lui expliquer le plus simplement possible la décision envisagée dans l’étude de son dossier.

En d’autres termes, « l’interprétation » désigne l’évaluation globale du processus de prise de décision. Elle vise à représenter l’importance relative de chaque variable. L’idée est donc d’expliciter au mieux le score renvoyé́ par le modèle. La classe LGBMClassifier permet de mieux comprendre le choix et l’importance des features via un attribut *feature\_importances\_*

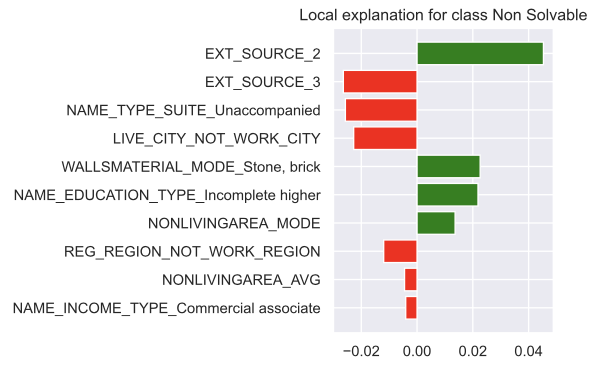
Une image contenant texte, capture d’écran, climatiseur

Description générée automatiquement

## Importance des features pour un client spécifique

Une image contenant table

Description générée automatiquement



La non-solvabilité du client '100763' est supérieur au seuil de solvabilité de 0.1, donc le client est non solvable. La variable qui joue le plus en sa défaveur est 'EXT\_SOURCE\_3' qui est un score normalisé provenant d'autres données. La variable qui joue le plus en sa faveur est 'LIVE\_REGION\_NOT\_WORK\_REGION', il ne vit pas dans la même région qu'il travaille.

**Perspectif d’amélioration**

L’espace de recherche HyperOpt permet de larges possibilités, le choix des hyperparamètres est évolutif, la question d’élargissement vers d’autres hyperparamètres peut également permettre d’augmenter les performances actuelles.