**NOTE METHODOLOGIQUE :**

Auteur : Xavier Fournat ([x.fournat@free.fr](mailto:x.fournat@free.fr))

Date de dernière mise à jour : 29.11.2021

Cette note méthodologique a pour but de présenter les différentes étapes de création de ce projet de création d’un modèle de scoring des demandes de crédits clients.

Plan de la note :

[1. Présentation du jeu de données 2](#_Toc89445831)

[2. Jointer les données 2](#_Toc89445832)

[3. Exploration Data Analysis du jeu de données final 4](#_Toc89445833)

[4. Features selection 8](#_Toc89445834)

[ Les features catégorielles : 8](#_Toc89445835)

[ Les features binaires : 10](#_Toc89445836)

[ Les features continues : 10](#_Toc89445837)

[5. Création du modèle final 11](#_Toc89445838)

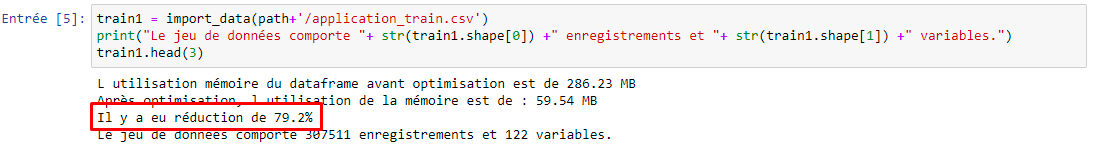
[6. Mieux comprendre les prédictions 16](#_Toc89445839)

# Présentation du jeu de données

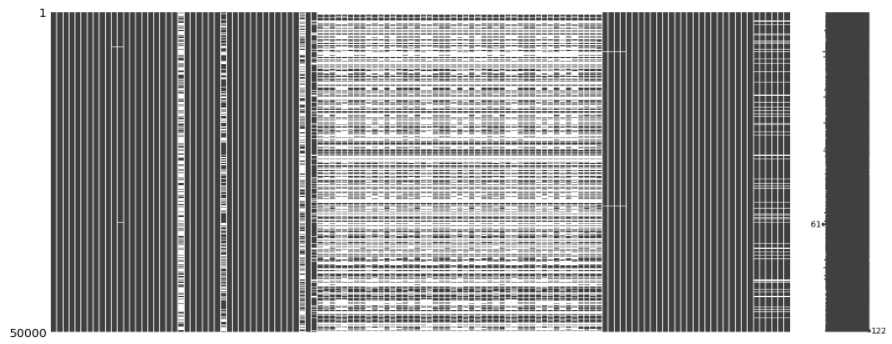
Le jeu de données est téléchargeable au lien suivant : <https://www.kaggle.com/c/home-credit-default-risk/data>

Lors du chargement des données, une fonction personnalisée « *reduce\_mem\_usage()* » d’optimisation de l’espace mémoire est utilisée. Elle va :

* Convertir en type de données catégorielles les variables qui peuvent l’être
* Pour les variables numériques, on va la convertir en entier si c’est possible. Puis, on va définir la valeur min et max, et appliquer le plus petit type de données qui irait avec cette plage.



Le dataset comportant également beaucoup de variables avec des taux de remplissages très faibles, on supprime toutes les données ayant un taux de remplissage inférieur à 75%.

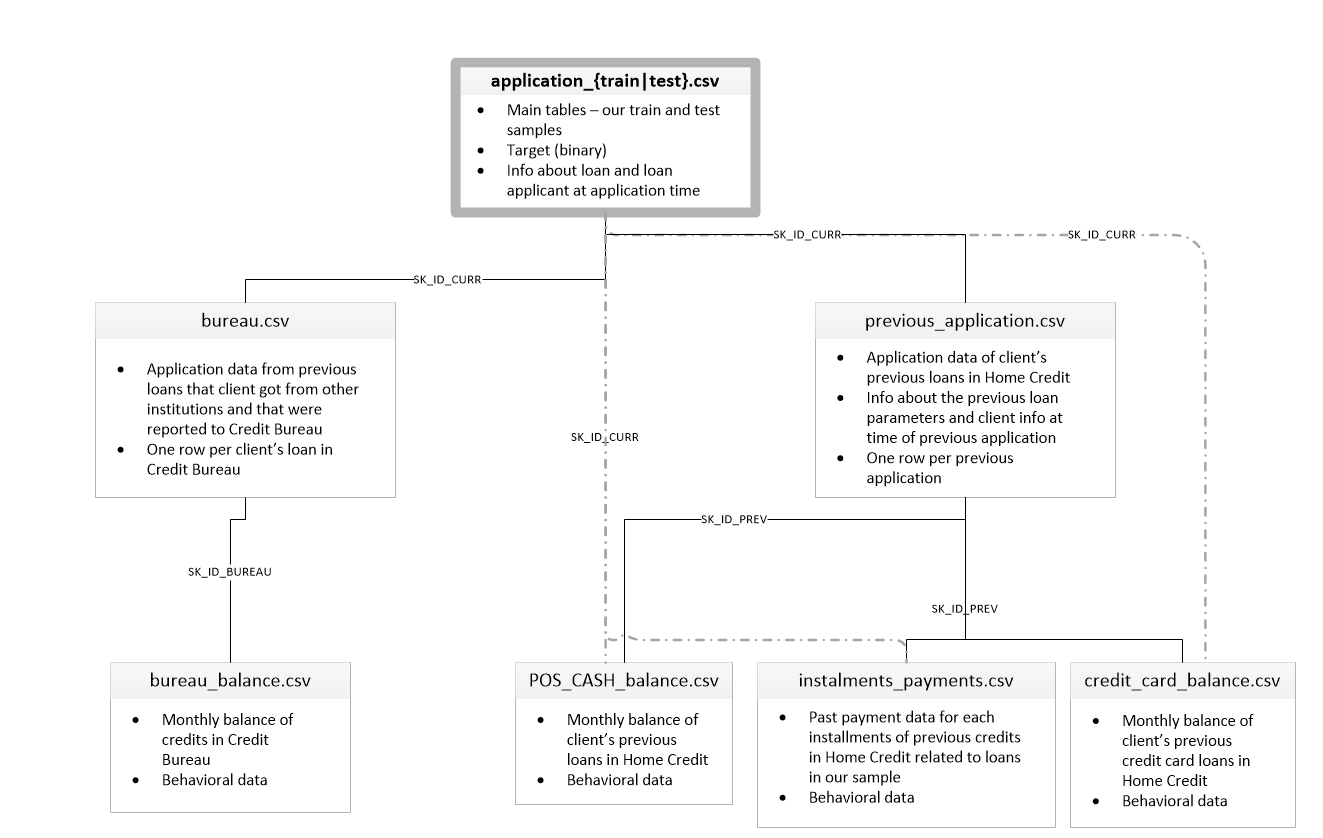


On va également préfixer tous les champs de chaque table, afin de conserver une trace de leur origine au moment de la jointure.

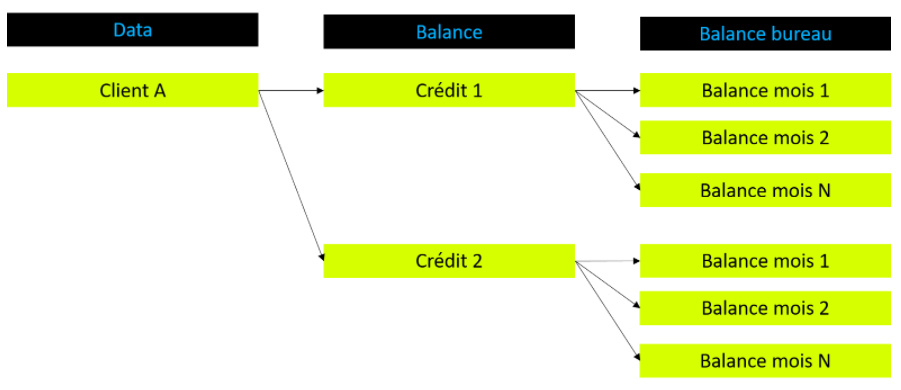


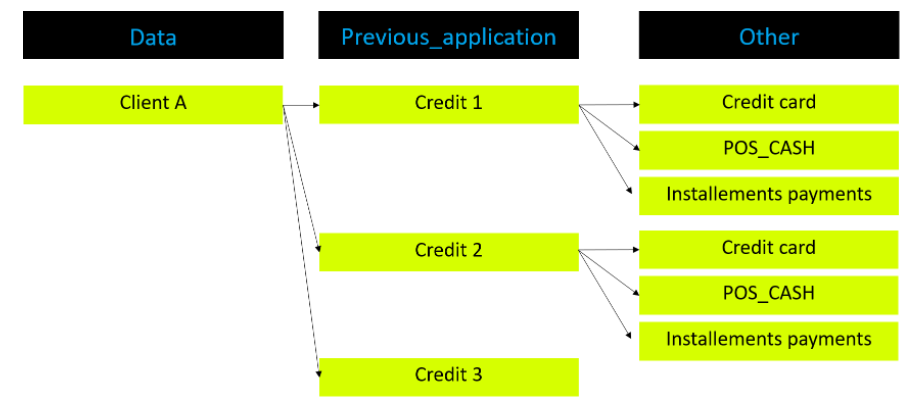
# Jointer les données

Le schéma de branchement des tables est le suivant :



Attention, le niveau de granularité des différentes tables n’est pas le même. Il va donc falloir réduire certains datasets avant de pouvoir les jointer. Par exemple, voici 2 diagrammes expliquant comment les données sont reliées entre elles.





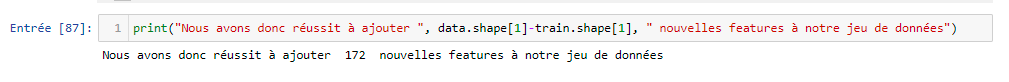
La méthode pour réduire les données est la suivante :

* Si un champ est numérique, on va agréger les différents enregistrements en créant des champs count, min, max, et mean.



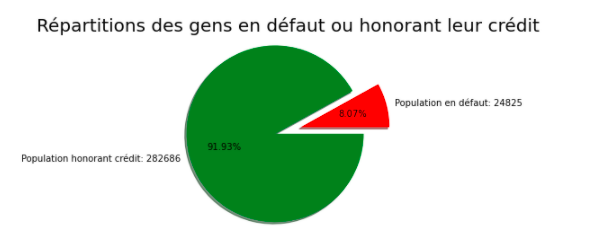
* Si un champ est catégoriel, on va agréger les différents enregistrements en conservant la classe modale la plus importante, et le nombre de catégories différentes pour un même ID.

Ce travail de jointure permet d’ajouter 172 nouvelles features à notre jeu de données.



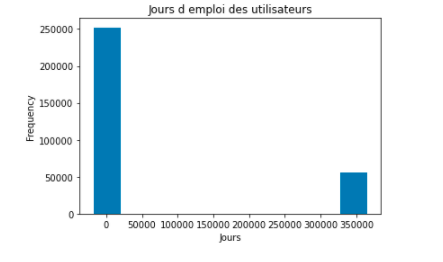
# Exploration Data Analysis du jeu de données final

L’observation la plus importante est que la variable target est fortement déséquilibrée, car le pourcentage des gens qui n’honorent pas leur crédit est seulement de 8.07%. Cela va avoir un impact très important dans notre approche de machine learning et la sélection des bons algorithmes et fonctions de couts.

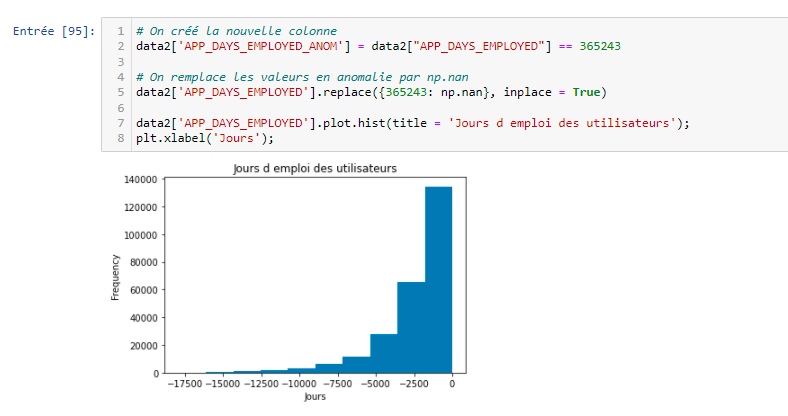


L’exploration de certaines variables permet de se rendre compte soit de données incohérentes, soit du besoin de feature engineering pour mieux les exploiter.

Par exemple, la variable DAYS\_EMPLOYED, qui donne l’ancienneté de la personne dans son travail. On voit des données incohérentes, car certains utilisateurs auraient plus de 1000 ans d’ancienneté.

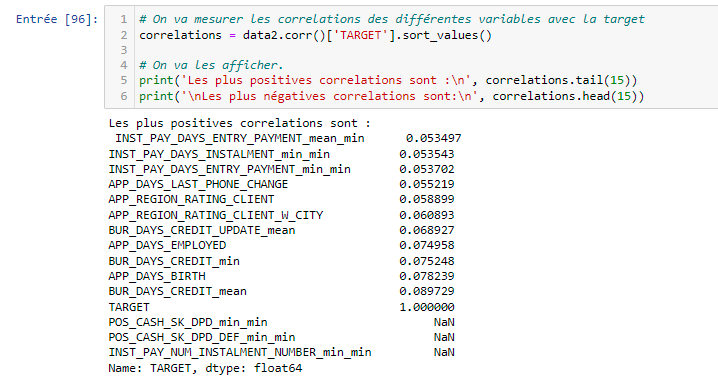


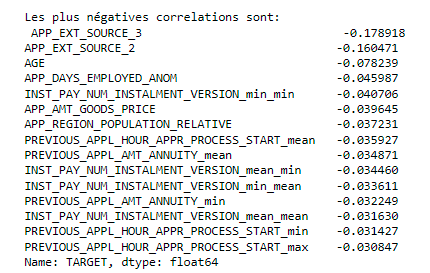
On va donc nettoyer ces données pour avoir quelque chose de plus cohérant :



On va ensuite analyser les corrélations des différentes variables avec la target pour voir celles qui ont le plus d’impact sur le modèle, avec l’hypotèse suivante :

* de 0.00 à 0.19 “très faible”
* de 0.20 à 0.39 “faible”
* de 0.40 à 0.59 “modéré”
* de 0.60 à 0.79 “fort”
* de 0.80 à 1.0 “très fort”





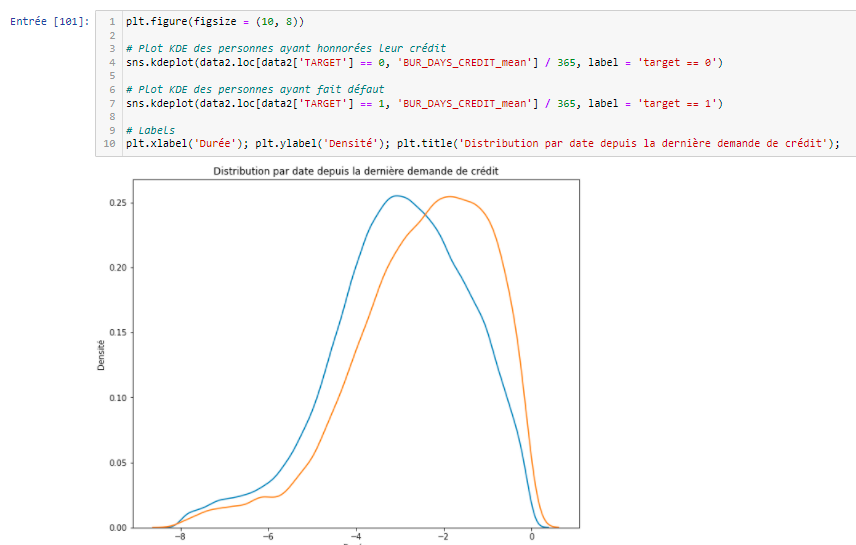
Les variables qui ont le plus d'influence sur la target sont donc :

* Le temps depuis lequel le client n'a pas souscrit à un crédit
* L’âge
* La durée depuis laquelle son emploi actuel est en cours
* La région du client (classée par niveau de vie)

Les variables n'ont cependant que très peu de corrélation avec la target

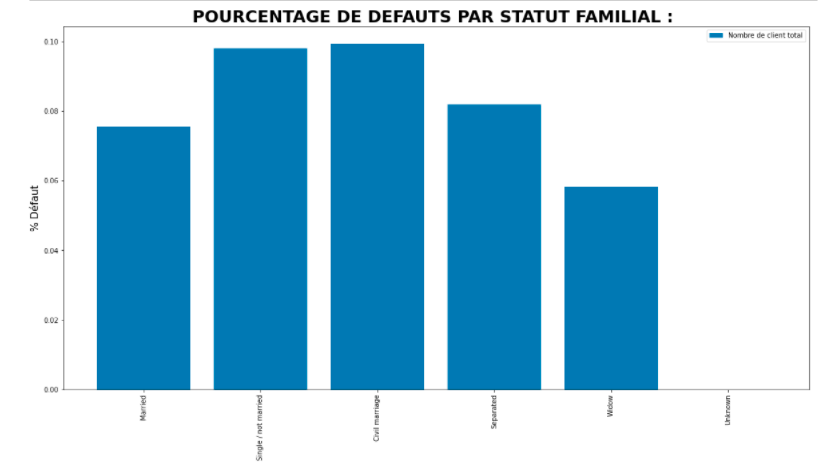
Pour les variables ayant la plus forte corrélation avec la target, on va utiliser une estimation par noyau (kernel density estimation plot (KDE)), en colorant la target. Le KDE permet de regarder la distribution d'une seule variable, et est en quelques sortes un histogramme lissé (on utilise un noyau Gaussien a chaque point des données, et on moyennise tous les noyaux pour développer la courbe).

Exemple pour la variable Temps depuis la dernière demande de crédit.

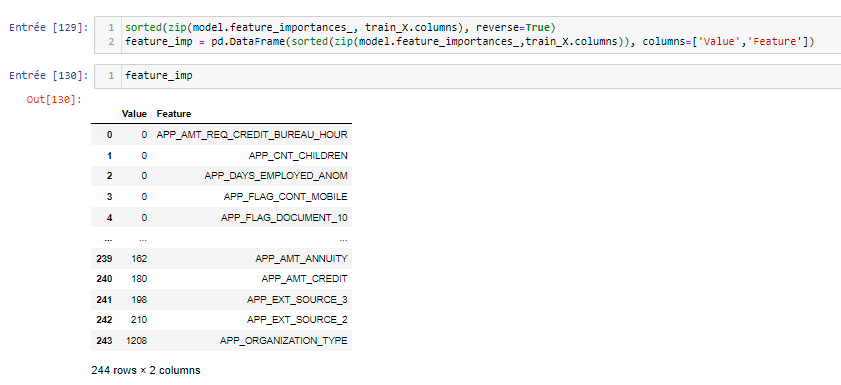


On voit que la courbe est skew vers les durées les plus récentes lorsque la target est 1. Même si le coefficient de corrélation est faible (0.089729) la variable est intéressante car elle affecte la target.

Pour les variables catégorielles les plus importantes, on va mesurer le pourcentage de défaut par catégorie, et regarder s’il y a des écarts significatifs. Par exemple pour le statut familial.

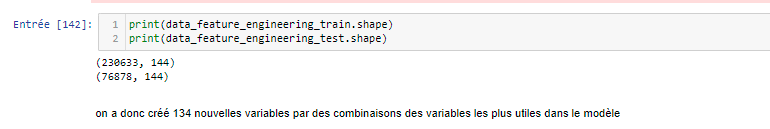


Pour aller plus loin dans le feature engineering, on va créer un premier modèle Light GBM très simple, pour comprendre quelles sont les features les plus importantes de notre jeu de données.



On va alors conserver les 10 plus importantes, et les combiner entre elles au automatisant avec la fonction personnalisée *feature\_engineering(Df)* la création de nouvelles features qui vont les sommer entre elles, mais aussi les soustraire, les multiplier et les diviser. La convention de nommage de ces nouveaux champs indique les deux champs dont ils sont issus et l’opération effectuée. Par exemple : INST\_PAY\_AMT\_PAYMENT\_min\_mean/APP\_EXT\_SOURCE\_3

Cela permet de créer un certain nombre de nouvelles variables.



# Features selection

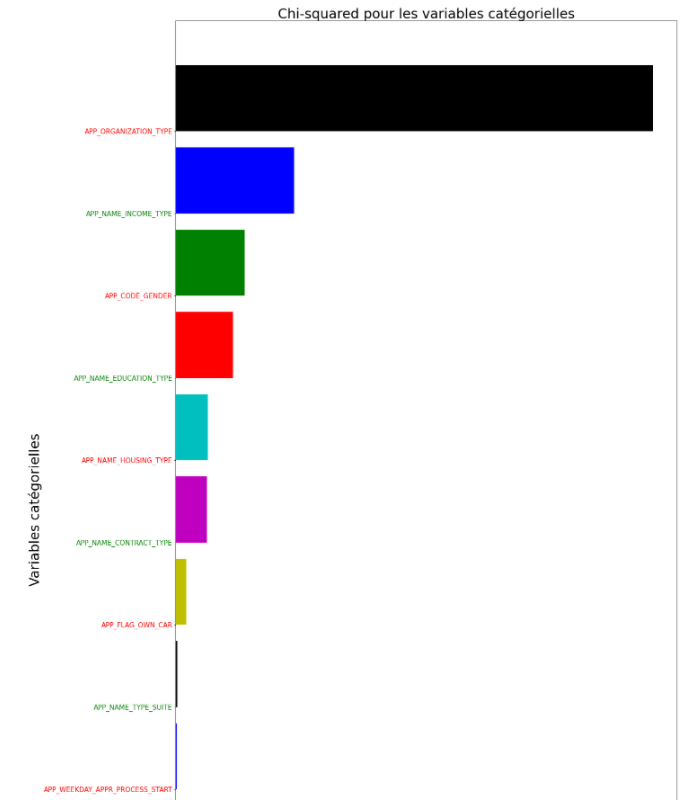
On divise l’approche en 3 parties :

## Les features catégorielles :

On a donc 11 features catégorielles. On va choisir pour la suite du modèle de ne pas les one hot encoder, car le modèle LGMB fonctionne parfaitement bien sans le faire à partir du moment où elles sont bien déclarées comme des variables catégorielles.

Pour sélectionner les meilleures features, on va utiliser le CHI2.

* Chi² examine le degré d'indépendance entre 2 vecteurs de variables catégorielles
* Chi² score est calculé pour les variables par rapport à la target
* On le calcule pour toutes les features et on le visualise.



Pour les sélectionner, on va boucler créer une boucle qui va :

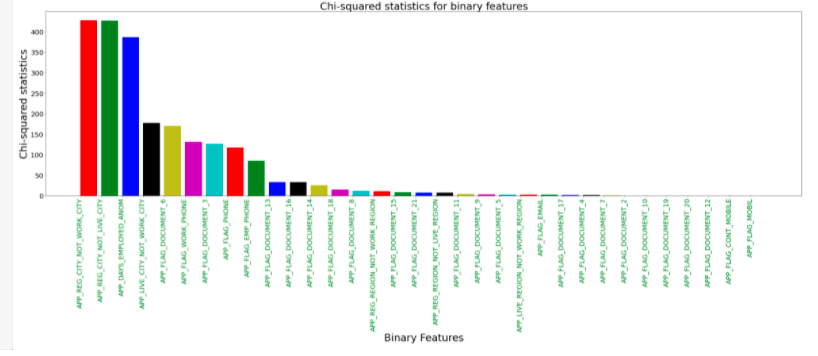
* Prendre les features 2 par 2, par ordre de score Chi2 décroissant
* Utiliser un modèle LGBM
* Calculer le ROC\_AUC socre pour chaque plage

On visualise l’évolution des performances du modèle pour voir si un éventuel effet plateau permettrait de réduire le nombre de variables du modèle.

## Les features binaires :

La même logique est utilisée pour les 32 variables binaires :

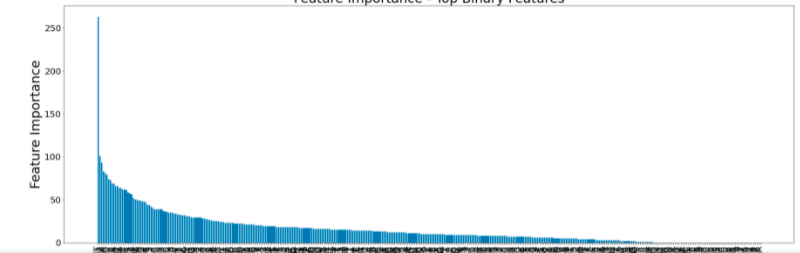
* Création d’un modèle baseline pour classer les features par ordre d’importance décroissant en utilisant le CHI2



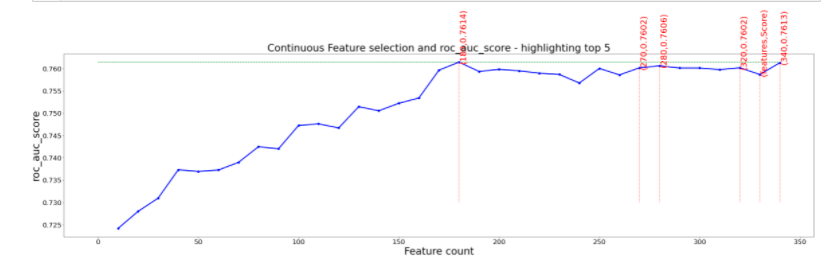
* Utilisation d’une boucle pour les ajouter 3 par trois au modèle et regarder si les performances de celui-ci trouvent un plateau qui permettrait d’en éliminer certaines. On a un plateau à 18 variables, ce qui permet d’en retirer 16 faiblement contributrices du modèle

## Les features continues :

La même logique est utilisée pour les 356 features continues. On va utiliser le ANOVA score pour scorer leur importance.



Puis on va les ajouter 20 par 20 dans notre modèle par ordre d’importance décroissante, et mesurer la performance à chaque itération.



On voit qu’à partir de 150 variables, le modèle connait un plateau de performance.

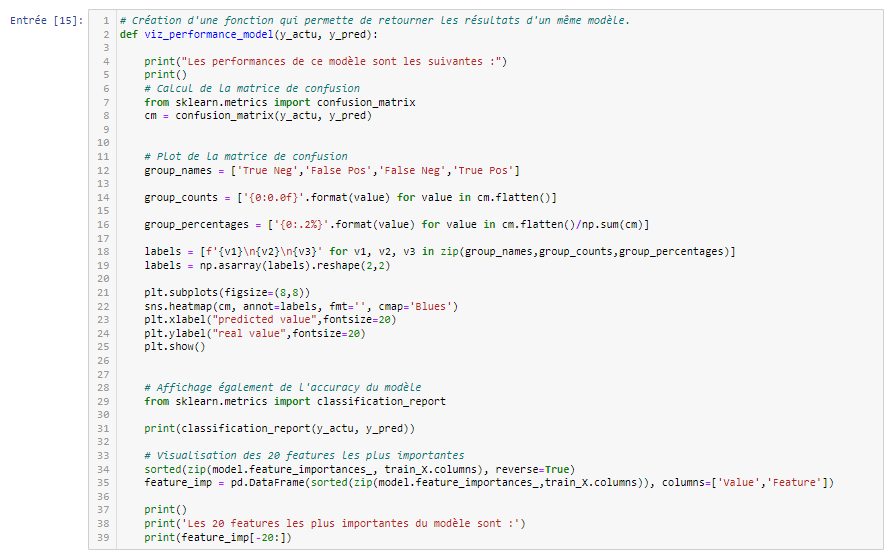
Ces 3 opérations successives permettent d’obtenir le dataset final qui comporte 178 variables.

# Création du modèle final

**Attention :** pour choisir le modèle final, on ne part pas uniquement sur des analyses des métriques de performances habituelles telles que le ROC\_AUC score.

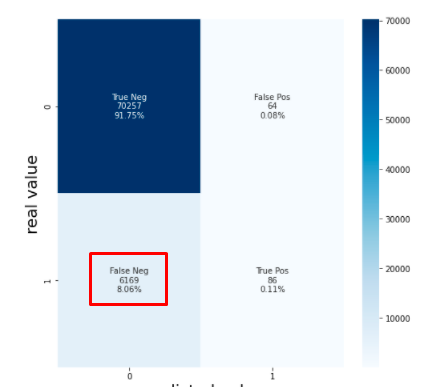
En effet, d’un point de vue métier, il est important que l’on ait un modèle qui prédise le moins possible de faux négatifs, c’est-à-dire de cas où le crédit est accepté alors qu’il ne le devrait pas. Ce sont ces cas de figure qui vont coûter le plus d’argent possible à la banque.

Une fonction est créée pour visualiser les résultats d’un modèle : *Viz\_perforamance \_model (model, test\_X, test\_y)* qui donne la matrice de confusion, le tableau des scores precision/recal/F1 score, et les 20 features les plus importantes du modèle.

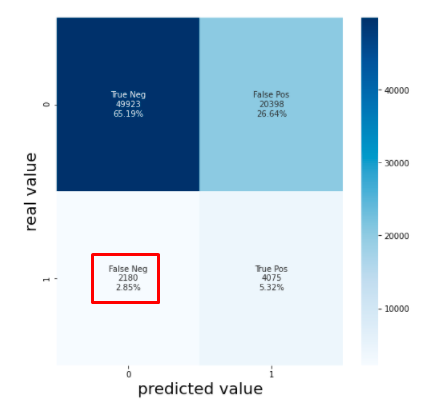


Les différents modèles testés sont les suivants :

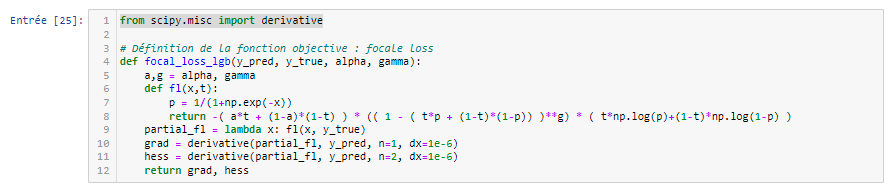
* LGBM, avec une fonction objective binary, comme il n’y a que 2 classes en sortie. Mais le modèle retourne beaucoup de faux négatifs

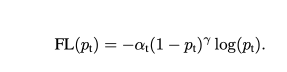


* LGBM, avec en utilisant le paramètre is\_unbalanced=True, sans aucun autre réglage, pour essayer de compenser l’inbalance des classes dans le jeu de données. Si le ROC\_AUC score est très proche du modèle précédent (0.744), le résultat est bien plus intéressant car on a un meilleur recall et bien moins de faux négatifs.



* LGBM en utilisant une nouvelle fonction objective, la focale loss, qui est spécialement utilisée dans les cas de fortes imbalances de classes.

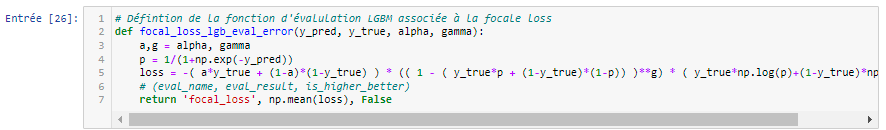




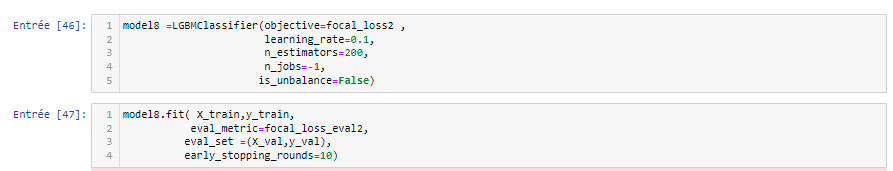
L’article suivant explique son fonctionnement mathématique :

<https://towardsdatascience.com/lightgbm-with-the-focal-loss-for-imbalanced-datasets-9836a9ae00ca>

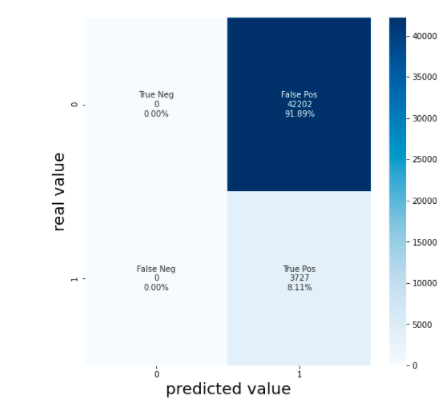
On va créer la fonction d’évaluation associée.



Sans modifier les paramètres, et après quelques essais pour correctement utiliser les fonctions (qui sont détaillés dans le notebook), on arrive à un premier modèle :

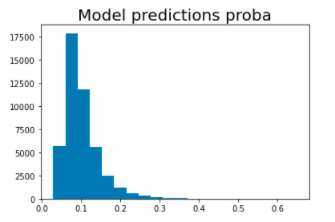


Il ne prédit par contre que des 1, et doit donc être correctement paramétré.

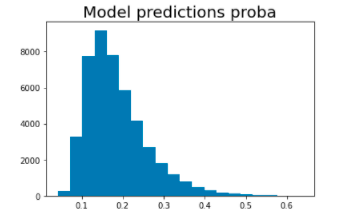


On va désormais essayer de faire varier les 2 hyperparamètres alpha et gamma de manière empirique pour comprendre leur influence sur le modèle. En sortie, on va à chaque fois regarder un histogramme de la distribution des proba (en faisant attention d’utiliser la fonction sigmoïde pour bien transformer la donnée de sortie du modèle en une proba réelle).

Pour a=0.2 et g =1



Puis en augmentant alpha à 0.5… jusqu’à 1, on se rend compte que alpha permet d’ajouter les prédictions entre 0 et 1 dans un axe horizontal.



Si on se rapproche de 1, seuls des 1 sont prédits. On définit donc que alpha doit être compris entre 0 et 1.

Si on procède de même pour Gamma, on se rend compte que cet hyperparamètre permet de plus ou moins resserrer la distribution autour de son pic. On définit de manière empirique que les valeurs sont entre 10 et 30.

On va ensuite réaliser une GridSearch, sur ces ranges d’hyperparamètres.

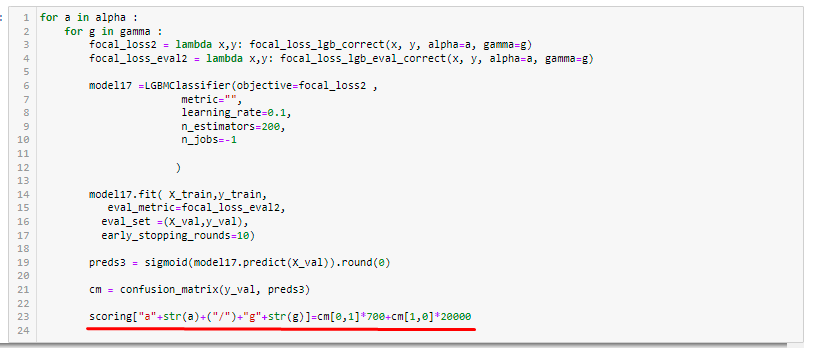


Pour scorer le modèle final, on va partir du métier. Comme il est important de ne pas prédire d'acceptation des crédits là où le client a fait effectivement défaut, c'est à dire de limiter au maximum le nombre de faux négatif. En ce qui concerne les faux positifs, ces dossiers restants devront être revus manuellement par un service dédié avant de notifier le client de son rejet. Il est important que cette quantité reste relativement faible pour pouvoir être gérable manuellement

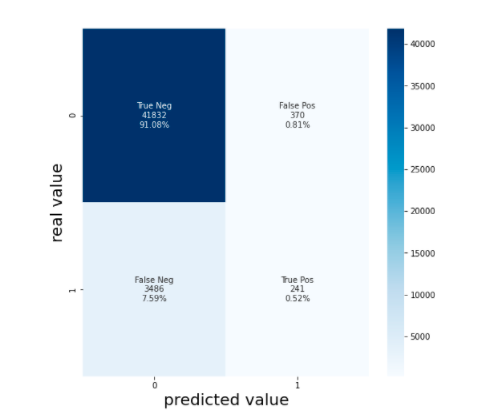
Pour valider correctement le bon modèle, on va définir la logique suivante :

* Chaque faux négatif va coûter 20 000€ à la banque
* Chaque faux positif va coûter 700€ de frais de retraitement manuel à la banque

On va enregistrer dans un dictionnaire le résultat de scoring de chaque itération, puis chercher le modèle qui permet de minimiser ces couts.



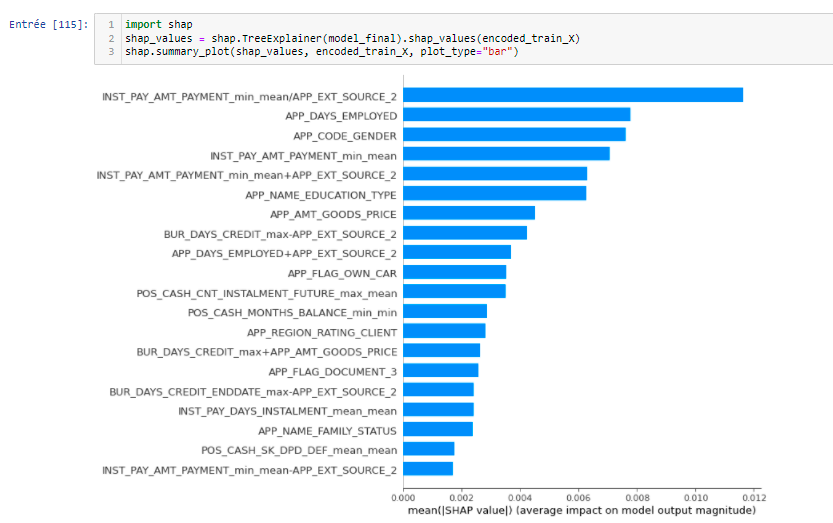
La meilleure combinaison d’hyperparamètres retenue est donc a=0.7 et g=20, qui donne ces performances :



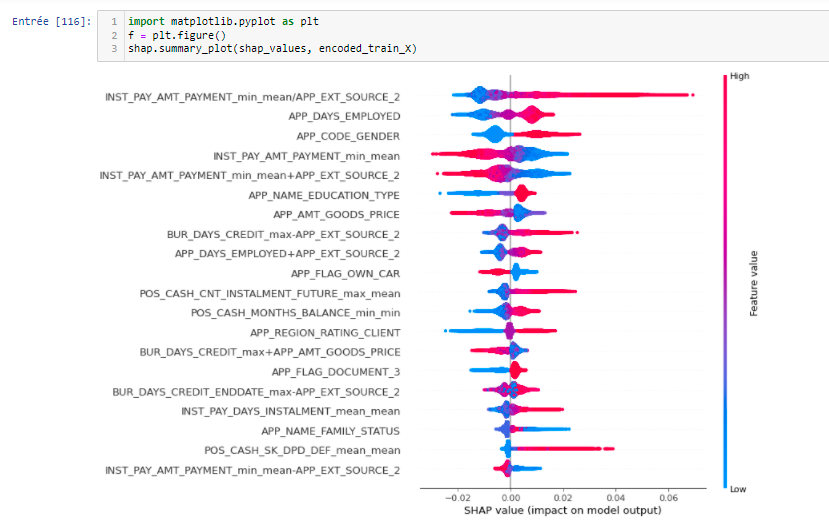
# Mieux comprendre les prédictions

Pour mieux comprendre les prédictions, on va utiliser le système des SHAP Values.

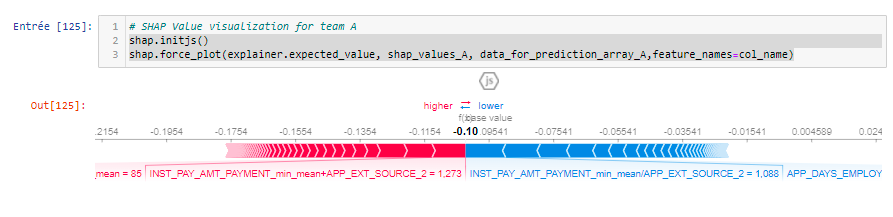
Tout d’abord pour visualiser les variables par ordre d’importance :



Puis pour visualiser l’impact positif et négatif des prédicteurs sur la target.



Enfin, pour mieux comprendre l’importance des prédicteurs sur une prédiction individuelle, on va utiliser le force\_plot.



Pour pouvoir utiliser le modèle dans d’autres programmes, on va l’exporter à l’aide de joblib, mais également à partir de la fonction dédiée du LGMB.

