Note méthodologique :

Objectif:

- Construire un modèle de scoring qui donnera une prédiction sur la probabilité de faillite d'un client de façon automatique.
- Construire un dashboard interactif à destination des gestionnaires de la relation client permettant d'interpréter les prédictions faites par le modèle, et d'améliorer la connaissance client des chargés de relation client

Données:

- Les données utilisées se trouve à l'adresse suivante: https://www.kaggle.com/c/home-credit-default-risk/data

Feature engineering:

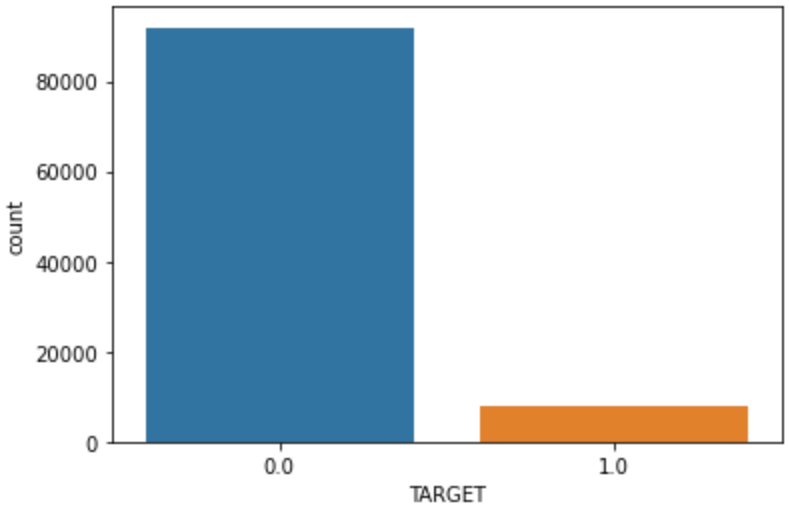
- Pour cette mission nous sommes incités à sélectionner un kernel Kaggle pour nous faciliter la préparation des données nécessaires à l'élaboration du modèle de scoring. ensuite analyser ce kernel et l'adapter pour nous assurer qu'il répond aux besoins de notre mission :

- 1/Télecharger les données
- 2/Concaténer les différents tableaux
- 3/Créer de nouvelles features

Le Kernel que j'ai utilisé et adapté ce trouve sur cette adresse: https://www.kaggle.com/jsaguiar/lightgbm-with-simple-features

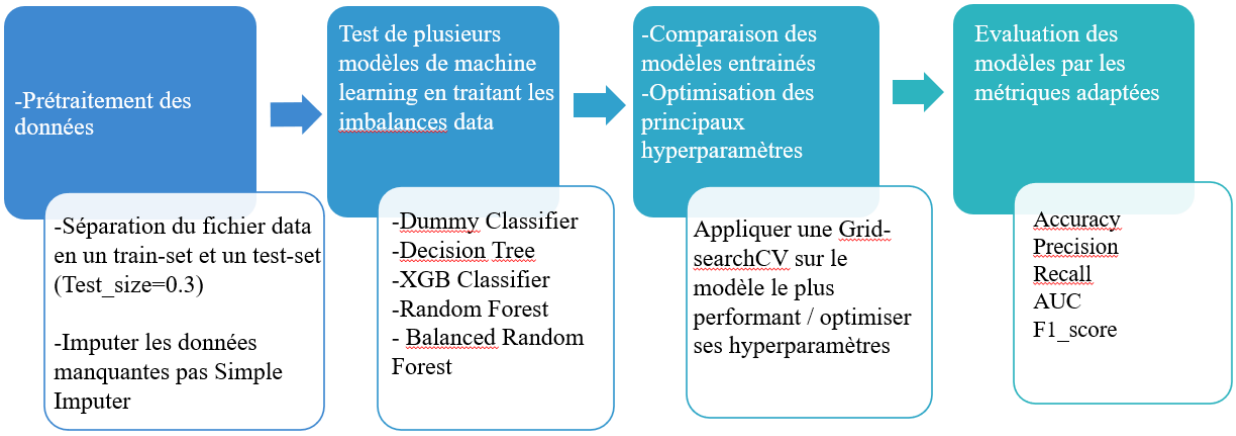
Imbalanced data:

- On remarque que la donnée à prédire "Target" est fortement désequilibrée (92% de bon clients"class:0")



simulation et comparaison des modèles:

- Comparaison de 4 modèles à un algorithme de base "DummyClassifier"; à ces modèles j'ai appliqué un "class_weight" pour traiter les classes non équilibrées/
- les modèles utilisés sont:
- XGBClassifier(scale_pos_weight=9)
- DecisionTreeClassifier(class_weight='balanced')
- RandomForestClassifier(class_weight='balanced')
- BalancedRandomForestClassifier()



Les meilleurs résultats sont obtenus par le modèle:

"Balanced Random Forest Classifier"

Fonction de coût métier – optimisation du modèle d'un point de vue métier

- Le modèle final doit pouvoir déterminer une classe 0 ou 1, ce qui implique de déterminer le seuil à partir duquel la proba calculée se transforme en classe 1 (pour un predict le seuil par défaut est 0.5).

- La problématique « métier » est de prendre en compte qu'un faux positifs (bon client considéré comme mauvais = crédit non accordé à tort, donc manque à gagner de la marge pour la banque) n'a pas le même coût qu'un faux négatif (mauvais client à qui on accorde un prêt, donc perte sur le capital non remboursé). Un faux négatif est environ 10 fois plus coûteux qu'un faux positif. Les mesures techniques ne le prennent pas en compte.

- Pour ma part j'ai utilisé le fbeta_score de sklearn, qui permet d'attribuer plus de poids à la minimisation des FN à travers la pondération du paramètre beta à qui j'ai attribué la valeur
'beta=3'

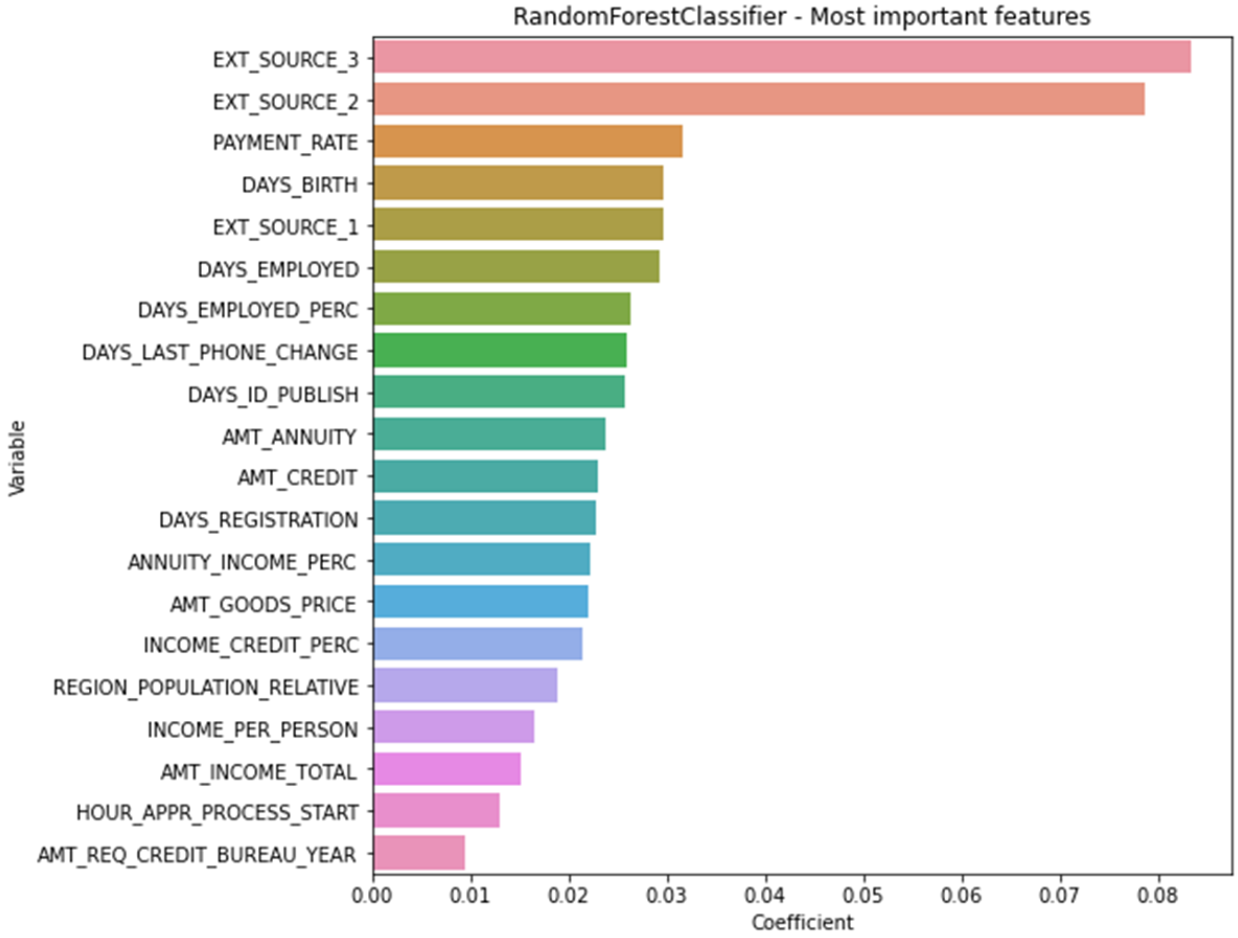
$$F_3 = \frac{TP}{TP + 0.1FP + 0.9FN}$$

Hyperparamètres à optimiser:

- n_estimators :nombre d'arbres de décision fixé à n=1000
- min_samples_split: nombres d'individus minimum pour que puisse avoir la séparation d'un noeud.
- min_samples_leaf: nombre de feuilles minimales dans un noeud
- max_features: nombre de features observées pour chaque arbre
- bootstrap: définit si chaque bootstrap se fait sur tout le jeu de donnée ou juste une partie

Interprétabilité du modèle:

-Feature Importance
globale:



Shap (Feature importance locale):

Le module SHAP peut être utilisé (il est indépendant du modèle) pour calculer les feature importance à partir du Random Forest. Il utilise les valeurs de Shapley Values de la théorie des jeux pour estimer la contribution de chaque feature à la prédiction. Il peut être facilement installé (pip install shap) et utilisé avec scikit-learn Random Forest Dans le dashboard les prédictions sont facilement interprétées par: -SHAP explanation force plot -SHAP explanation summary plot

Dashboard.

Implémentation.

Le Dashboard interactif a été implémenté via **Streamlit** et codé en Python. Les packages nécessaires pour exécuter cette application sont répertoriés dans le fichier *requirements.txt*. L'application principale *app.py* est livrée avec les éléments suivants :

- Les données se trouvent dans "train_Xy_sample et X_test_sample"qui sont des echantillons car les fichiers entiers sont très gros en terme de mémoire.
- Le fichier : *finalized_model.sav*, contient le modèle entraîné. Le package *pickle* a été utilisé pour exporter le modèle après validation croisée et apprentissage.

Déploiement:

J'ai déployé mon application via Heroku

- Lien du Dashboard: https://housnadashboardp7.herokuapp.com/?fbclid=IwAR28DSi57JUYtf5ChTW_8Bifc8aXWCSuiQ0ao7Ar6Owz9nr8Xc56IWRwrLw