# Méthodologie d’entrainement du modèle (fait)

## L’enrichissement de la donnée

La base de données de base est constituée de 9 fichiers csv pour un total de 225 colonnes. Il est possible de créer d’avantage d’informations sur les données avec un travail de feature engineering. La première étape est donc de créer de nouvelles variables à partir de celles disponibles.

Nous nous retrouvons avec 389 variables. Les nouvelles variables crées n’ont pas toutes un nombre suffisant de valeurs pour être utilisées. Nous supprimons les colonnes remplies à moins de 80%.

Entrainer un modèle sur autant de variable nécessiterai énormément de ressources. Aussi allons-nous établir l’importance des variables dans l’entrainement des modèles pour ne garder que les plus importante. On parle de « feature selection ».

## Feature selection

La méthodologie est d’estimer l’importance des variables à disposition dans le résultat à l’aide de plusieurs modèles. Nous compterons ensuite pour chaque variables le nombre de modèle qui l’ont estimé importante. Les variables estimées importantes par plusieurs modèles seront conservées.

### Corrélation de Pearson

Cette méthode regarde la corrélation de chaque variable avec la valeur de prédiction.

### CHI 2 avec SelectKBest

La méthode SelectKBest de scikitlearn va sélectionner le nombre demandé des meilleurs variables en se basant sur le score passé en argument.

### RFE avec la régression logistic (wrapper)

La bibliothèque RFE, Recursive FEatures selection, va entrainer un modèle sur un jeu de donnée comportant de moins en moins de variables à mesure que les variables les moins importantes sont éliminé à chaque nouvel entrainement. L’algorithme s’arrête une fois que le nombre de variable demandé est atteint.

### SelectFromModel (embedded): LogisticRegression, Random forest, LightGBM

La méthode SelectFromModel va entrainer un modèle sur notre jeu de données et donner un score d’importance à chacune des variables. On fixe alors une valeur seuil. Si la valeur absolu de la variable est supérieur à ce seuil, la variable est conservée.

### Dernière étape

La dernière étape est de créer un dataframe listant les variables de notre base et comptant la récurrence avec laquelle elles ont été estimées importantes de la part des différents algorithmes. On décide de ne garder que les variables avec une récurrence d’au moins 3. On peut alors filtrer la base en ne conservant que les variables jugées importantes et l’enregistrer en csv.

## Entrainement du modèle

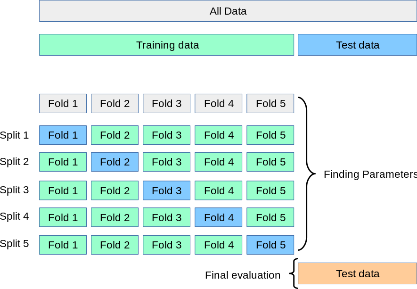
Pour choisir quel modèle utiliser, nous allons comparer différents algorithmes. Le résultat de chacun pouvant varier en fonction de leurs hyperparamètres, il faut dans un premier temps, pour chacun d’entre eux, déterminer leurs meilleurs hyperparamètres. La méthode la plus exhaustive est de procéder par grid search et cross validation.

Grid search :

Pour un algo donné on liste les hyperparamètres que nous voulons personnaliser. Pour chacun de ses hyperparamètres, nous spécifions également l’intervalles de valeurs qu’ils peuvent prendre. La méthode grid search va tester toutes les combinaisons possibles des différentes valeurs de chaque hyper-paramètre. Cette méthode est complète et aucun cas n’est laissé de côté

Cross validation :

La cross validation décrit une méthode d’entrainement d’algorithme dans laquelle le jeu de données d’entrainement est subdiviser en un nombre donné de sous base appelé des « folds ». LE but de cette méthode est de limiter l’overfitting.

L’algorithme va s’entrainer sur l’ensemble des folds sauf un qui servira à calculer le score. Une fois le score calculé, l’algorithme va à nouveau s’entrainer mais cette fois, le folds ayant servi à l’évaluer va faire partie de la base d’entrainement et l’un des précédent folds d’entrainement servira à évaluer la modèle ajusté. L’opération se répète jusqu’à ce que chaque folds ait servi une fois à évaluer l’entrainement du modèle.

Cette méthode est très gourmande en ressource aussi d’autre approches existent.

Méthode bayesian :

Les méthodes bayesians évaluent également différentes valeurs possibles des hyperparamètres mais ne vont pas tester l’ensemble des combinaisons possibles. Elles vont en effet essayer un échantillon de ces combinaisons. Cela a pour effet de diminuer drastiquement le nombre de calcul pour un résultat très correcte.

Optuna

La librairie utilisée s’appelle optuna et applique cette approche bayesian. Son utilisation nécessite de définir une fonction « tune » indiquant le score à utiliser, le modèle à entrainer, le critère d’optimisation et le nombre de cycle d’entrainement. Une deuxième fonction « objective » contient les hyperparamètres avec leur fourchette de valeurs, l’instance du modèle à entrainer et le calcule du score.

MLflow

Pour suivre l’évolution de l’entrainement et des valeurs des hyperparamètres, la librairie MLflow a été utilisée. Elle permet de créer des « études » séparée dans lesquelles on peut sauvegarder le modèle entrainé avec les valeurs de ses hyperparamètres et son score.

Une fois tous les modèles entrainés avec la méthode décrite précédemment, nous avons alors un registre des modèles optimisé pour notre problématique. Il est alors facile de les comparer entre eux pour choisir le plus adapté.

Métric utilisée

La métric utilisé dans ce problème a été le roc\_auc qui était plus adapté que « l’accuracy ».

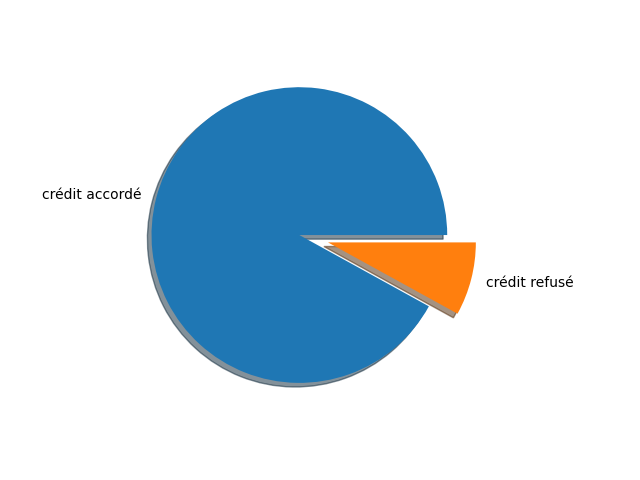
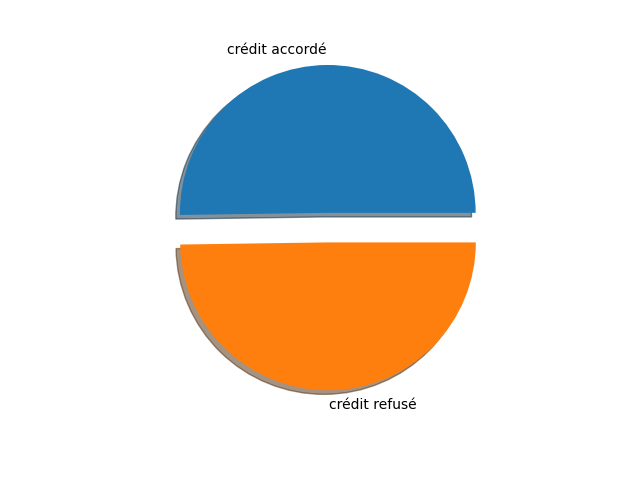
Résultat

Après comparaison, il s’avère que l’algorithme de régression logistique est le plus adapté à notre situation. Ce modèle sera entrainé avec un cout métier, détaillé plus loin, pour comparaison.

# Traitement du déséquilibre des classes (fait)

Avant l’entrainement des modèles, une rapide analyse de la variable cible montre une sur-représentation d’un résultat sur l’autre. On parle de déséquilibre des classes. Plusieurs librairies permettent de traiter ce problème.

### SMOTE

SMOTE est une méthode de la librairie ImbalancedLearn qui applique la technique de l’oversampling. Cela signifie que la méthode va apprendre le comportement des lignes de la classe minoritaire et générer de nouvelles lignes aléatoires respectant ce comportement.

Avant (17281/197973) Après (197973/197973)

# Métrique d’évaluation, coût métier et algorithme d’optimisation (fait)

## Métrique d’évaluation

Pendant toute la première partie d’entrainement du modèle nous avons utilisé une métrique d’évaluation statistique. Par défaut les modèles utilisent l’accuracy. Cependant cette métrique n’est pas adaptée dans notre cas. En effet l’accuracy va évaluer le nombre de bonne prédiction du modèle sur toutes les prédictions. Or dans nos données tests le déséquilibre des classes est toujours présent. Ainsi nous allons avoir une bonne accuracy avec la classe majoritaire mais le score sera aveugle quand à la qualité de prédiction de la classe minoritaire.

Nous avons donc utilisé la métrique roc\_auc plus adapté à notre sujet.

## Fonction coût métier

Les modèles ont donc dans un premier temps été entrainés avec un score statistique. Cependant dans certains cas, il peut être intéressant d’avoir un autre critère d’un point de vue métier. Ici, le modèle est sensé détecter les clients non solvables. En intelligence artificielle chaque modèle génère des vrais/faux positif/négatif. Il est possible de visualiser ces quatre cas simultanément dans une matrice de confusion. Dans notre cas un faux positif décrit un client solvable jugé non solvable et un faux négatif un client à risque non détecté. Chaque fausse estimation coûte de l’argent à la société. Cependant un faux positif génère la perte de l’intérêt sur le prêt alors qu’un faux négatif peut engendrer la perte de tout le prêt en plus des intérêts. Ainsi d’un point de vue métier il est intéressant de donner plus d’importance à ce dernier cas.

Une approche simple est de créer notre propre fonction de coût, qui sera utilisé par notre modèle, dans lequel les faux négatifs ont un ordre de grandeur d’importance supérieur et d’entrainer nos algorithmes pour minimiser ce coût.

Le code de notre nouvelle fonction de coût est le suivant :

Une image contenant texte

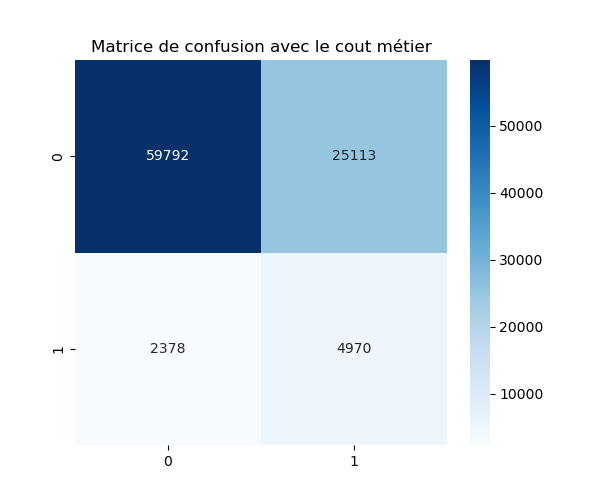
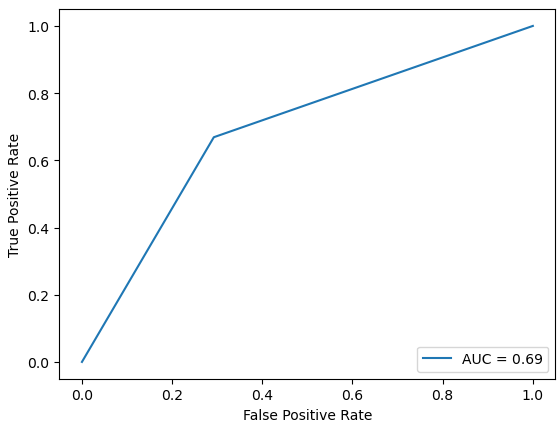
Description générée automatiquement On peut voir que le coût consiste à faire la somme des faux positifs et des faux négatifs multipliés par 10.

## Algorithme d’optimisation

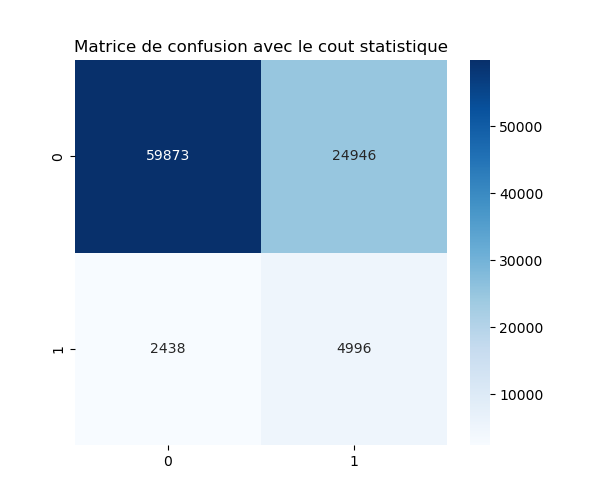
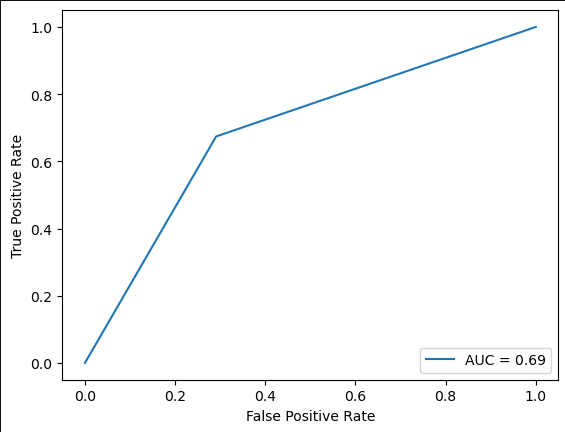
Nous entrainons ensuite l’algorithme exactement de la même manière que précédemment (optimisation des hyper paramètres avec optuna et stockage du modèle dans mlflow) mais en utilisant cette fonction de coût métier comme critère, en demandant à l’algorithme de trouver la plus petite valeur, option possible dans les paramétrages d’optuna.

# Synthèse des résultats (fait)

Nous venons donc d’entrainer notre modèle une nouvelle fois en lui demandant de minimiser le score métier que nous avons élaboré. La pertinence de cette fonction se décide en comparant les deux matrices de confusions :

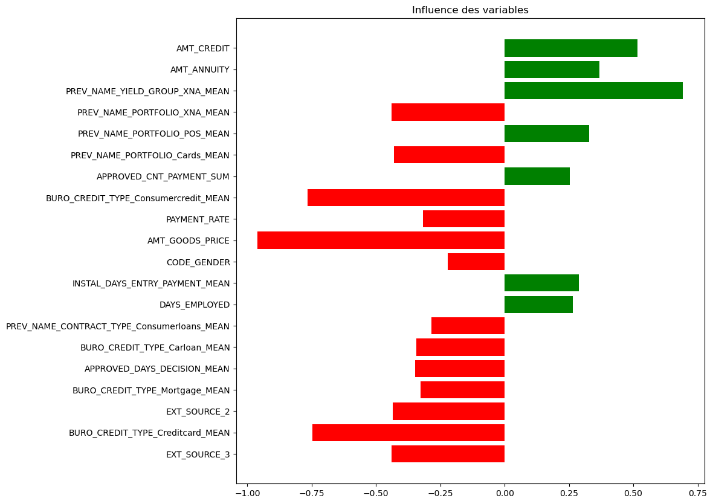


Nous pouvons voir que notre fonction de coût métier nous permet de diminuer les faux négatifs de 60 ce qui est appréciable tout en conservant la AUC\_ROC. Nous utiliserons donc le modèle entrainé avec la fonction de coût métier.



# Interprétabilité locale et globale (fait)

## Interprétabilité globale

L’interprétabilité global sert à expliciter le comportement du modèle. On va aller chercher l’importance de chacune des variables et comment elles influencent le résultat.

Cette information est directement disponible dans le modèle lui-même. En effet nombreuses sont les bibliothèques de machine learning qui, une fois le modèle entrainé, conserve l’importance des variables comme attribut de l’instance du modèle.

Ainsi dans notre cas de la régression logistique qui a été le modèle retenu et sauvegardé grâce à l’outil MLFlow, l’importance des variables est accessible via la ligne de code suivante : *[ i for i in zip(log\_reg\_metier.\_model\_impl.coef\_[0],log\_reg\_metier.\_model\_impl.feature\_names\_in\_) if abs(i[0])>=0.2].* Ici nous ne gardons que les variables ayant une valeur absolue du score supérieur à 0.2 et étant donc les plus importantes dans le comportement du modèle.

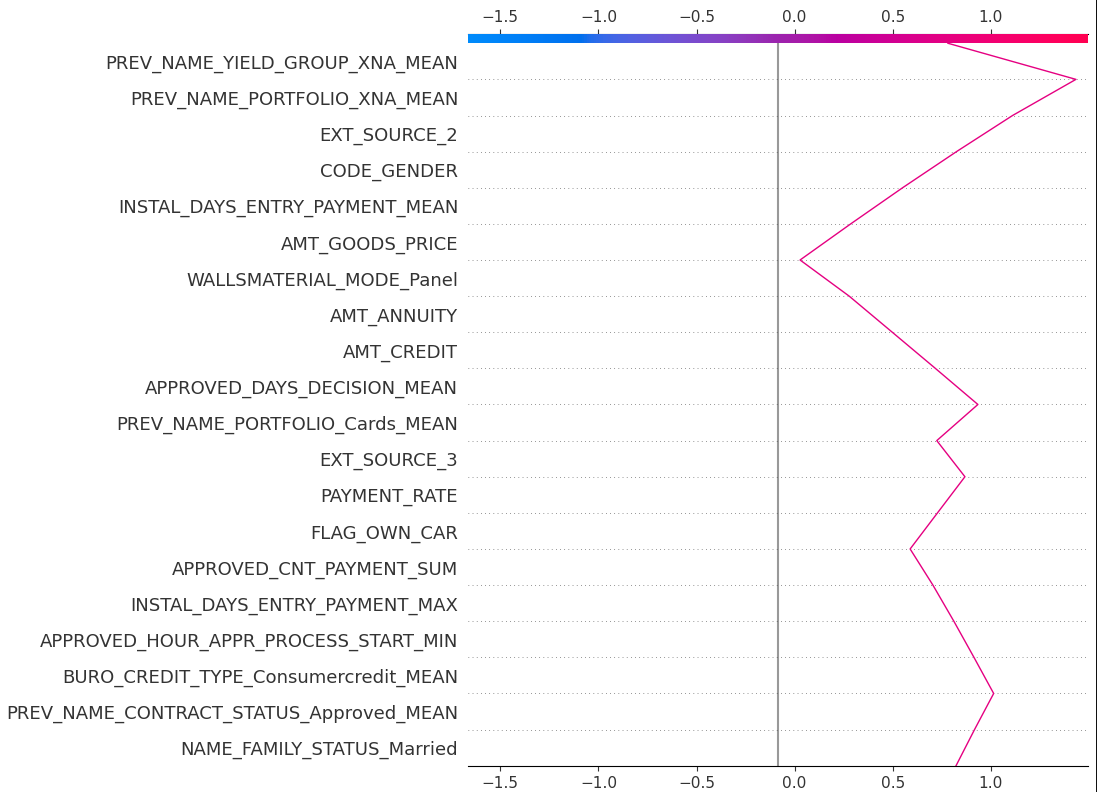
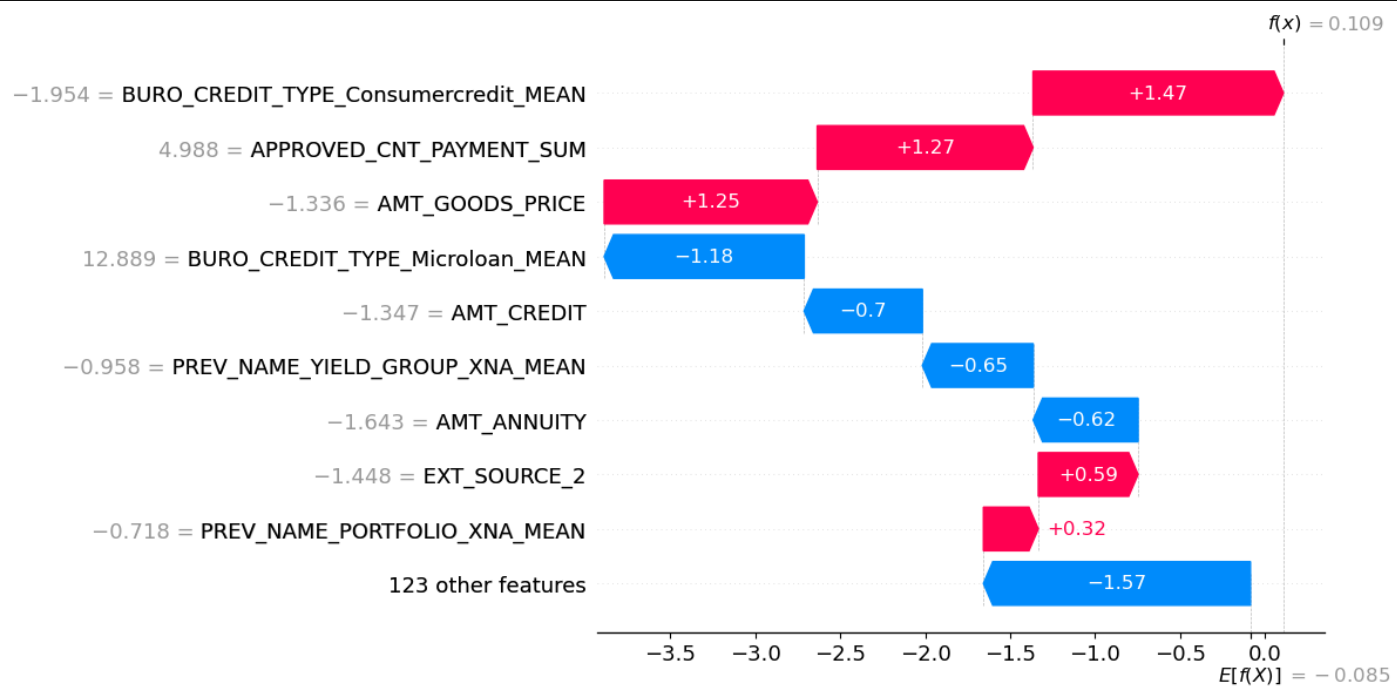
## Interprétabilité locale

L’interprétabilité locale va être utilisée pour expliquer le résultat d’une prédiction en particulier.

Dans le notebook nous avons étudié deux librairies : LIME et SHAP. La librairies shap propose des graphiques dont la lecture et la compréhension sont plus intuitives.

Shap va prendre le modèle en argument. À partir de ce modèle il va évaluer l’impact de chacune des variables sur le résultat du modèle et établir une « expected value ». Cette variable dont la valeur est unique pour tout le modèle va servir de seuil pour classer une prédiction dans une classe ou une autre.

Pour chacune des lignes, il va ainsi donner un score, positif ou négatif à chacune des variables. Quand on regarde le résultat d’une prédiction, le modèle va faire la somme des scores de chaque variable. Le résultat de cette somme, comparé à la valeur de « expected value » va expliciter le résultat de la prédiction. L’évolution de ce score est présentée de manières différentes sur les graphiques que la librairie propose. Ici à gauche nous avons un graphique de type « waterfallplot » et à droite un graphique de type « decision plot » où l’expected value est représentée par la ligne verticale noire.



# Limites et améliorations envisageables

Le champ des possibles pour l’amélioration des performances d’un modèle est extrêmement vaste. Dans le but d’identifier des pistes il faut tout d’abord connaitre les limites du modèle qui vient d’être entrainé

## Limites du modèle

* Le modèle n’a pas pu être entrainé sur la totalité des données par manque de ressources matérielles.
* Le coût métier est somme toute assez simpliste.
* La mise à jour de la base de données est manuelle et nécessite une mise en production manuelle à chaque fois.
* Le modèle ne propose pas de pistes d’améliorations pour le client qui verrait son prêt refusé.
* Les données sont désiquilibrées.

## Hypothèses envisageables

* Mixer l’oversampling et l’undersampling lors de l’équilibrage des classes pourrait permettre un plus grand réalisme des données en limitant la création d’entrées factice et comportant donc un biais potentiellement important.
* De plus lors de l’équilibrage des classes, l’objectif pourrait être de ne pas forcément arriver à un équilibre des classes parfait. Si les données réelles font que l’une des classes constitue le résultat majoritaire d’une étude de prêt, vouloir reproduire cette tendance peut améliorer le score du modèle.
* Nous pouvons supposer qu’avec plus de ressource et donc de données le modèle peut améliorer son apprentissage et donc son score.
* Affiner d’avantage les hyperparamètres
* Dans l’établissement du score métier, faire intervenir un professionnel du métier qui puisse apporter des métriques bancaires plus pertinentes.
* Définition des données anonymisées externes.
* Dashboard plus élaboré (utilisation des cache function, mettre plus de design)

# Analyse du datadrift (fait)

Lorsqu’un modèle de machine learning est entrainé, il utilise les données telles qu’elles sont à l’instant « t ». Au fil du temps, la distribution des données peut changer. Il est donc important de regarder comment évolue la pertinence du modèle avec l’évolution des données à disposition. La datadrift décrit l’évolution des données du monde réelles par rapports aux données utilisées pour entrainer notre modèle. Avant de rafraichir la base de données il faut s’assurer que ces nouvelles données ne provoquent pas un datadrift.

La librairie evidently permet de réaliser des contrôles de datadrift de manière simplifiée avec des rapports « out of the box ». Ici « DataDriftPreset » met à disposition un ensemble de métriques, évaluation d’un aspect spécifique de la donnée, utiles pour l’évaluation du datadrift et mis en forme sous forme d’un rapport visuel accessible au format HTML (attention le rapport html n’est pas affichable dans un notebook tournant sur windows).

## Méthodologie

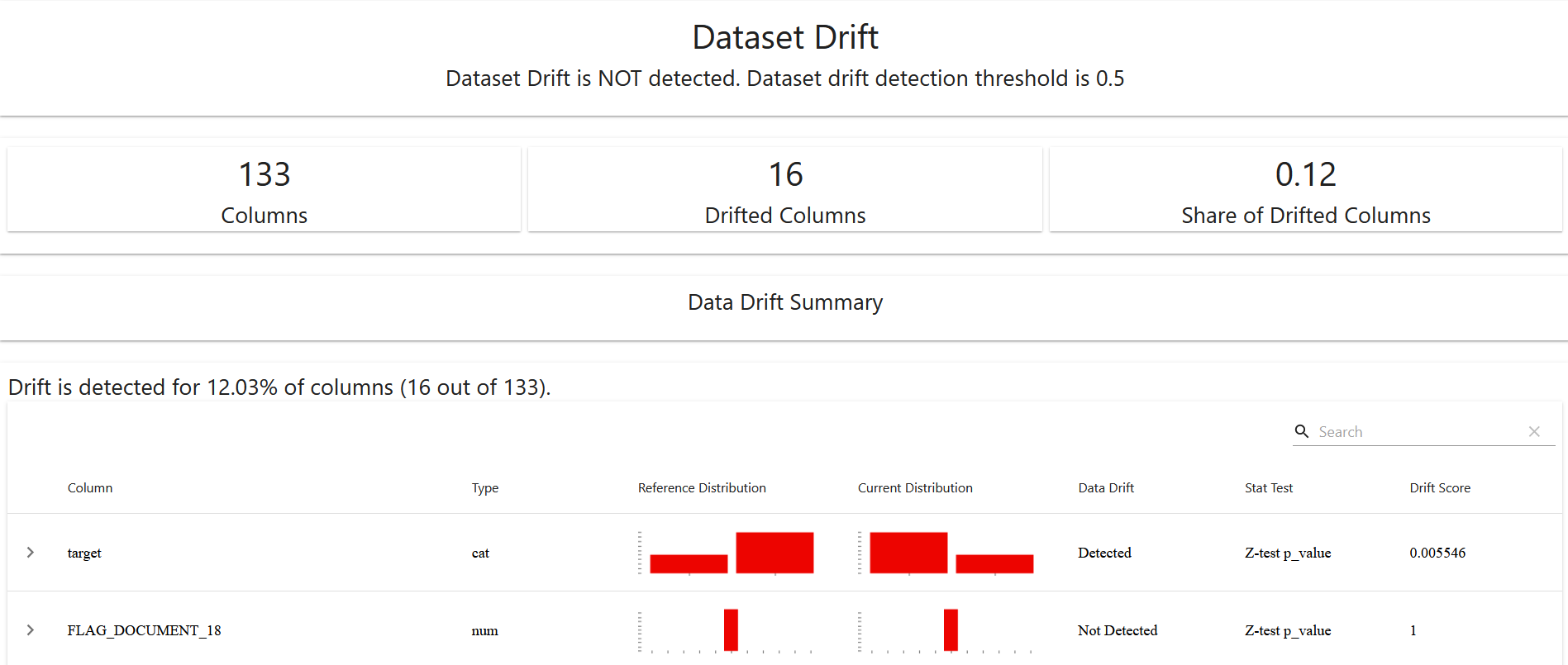
Après la phase d’équilibrage des classes, une partie des données d’entrainement sont sauvegardées sous format csv. On fait de même avec les données de tests en s’assurant d’avoir le même nombre de lignes et les mêmes variables.

On lance ensuite les lignes de code suivantes : Une image contenant texte

Description générée automatiquement

Nous obtenons ainsi un rapport html de l’analyse du datadrift.

## Résultats :



Nous pouvons voir que dans la globalité des données le datadrift n’est pas présent et pourra facilement être évalué lors de chaque mise à jour de la base de donnée.

Le tableau présenté sur la capture d’écran est tronqué, l’ensemble des 133 variables sont analysées.

Le drift détecté pour la TARGET n’en est pas vraiment un. En effet les données de test n’ont pas été sujet à un équilibrage des classes.