# NOTE DE SYNTHÈSE PROJET FINAL

## I) Démarche de travail :

- **Objectif** : Trouver la meilleure méthode pour transformer et agréger les données relatives au panier client des partenaires pour détecter les cas de fraude
- **Métrique** : l'aire sous la courbe Précision-Rappel, appelé également PR-AUC
- Le meilleur modèle correspondra à celui avec la valeur de PR-AUC la plus élevée.

## II) Présentation du jeu de données et de la problématique associée :

Problématique: Comprendre le mieux possible les données, ce qu'elles représentent, pour pouvoir développer par la suite une modélisation.

Analyse de forme :

On commence par identifier notre **target variable** qui est la variable que nous essayons de prédire. Dans notre cas, c'est fraud_flag, **variable de sortie** indiquant si une transaction est frauduleuse ou non ( =1 pour une transaction frauduleuse, =0 pour une transaction non frauduleuse).

Puis on identifie le nombre de **lignes et de colonnes** (92 790 lignes, 146 colonnes) pour obtenir une idée de la **taille** et de la structure des données. C'est notamment utile pour la suite du prétraitement, analyse, et nettoyage des données.

On détermine les **variables en entrée** : ID, item, prix, fabricant, model, code de l'item, nombre de produits, nombre d'item. Cela est crucial pour distinguer les features de chaque observation, pour extraire celles qui sont pertinentes et de qualité pour notre modèle.

En présence d'un grand dataset, il est important de connaître les **types de variables**: on a 94 variables qualitatives, et 52 quantitatives. En effet, nous serons amenés à traiter différemment les variables de types différents, comme la normalisation, l'encodage…

Enfin, on analyse les **valeurs manquantes**, en les visualisant. On affiche notre dataset dans une image avec la fonction heatmap.

## III) Prétraitement des données :

### a) Suppression des colonnes inutiles

Nous avons commencé par supprimer les colonnes goods_code car nous avons jugé que la probabilité que le client soit un fraudeur n'était pas corrélé avec le code de l'item choisi par le client. Nous avons donc créé une fonction à cet effet.

### b) Encodage

La plupart des algorithmes de machine learning ne sont pas capables de traiter les variables catégorielles telles quelles. Ils nécessitent que les données soient encodées sous forme numérique.

Pour encoder correctement les variables catégorielles, il faut choisir la bonne méthode. Les variables catégorielles que nous avions étaient les variables item, make et model. Nous avons commencé par encoder avec la méthode du Label Encoding. Ce dernier consiste à convertir des valeurs catégorielles en valeurs numériques en attribuant un entier unique à chaque catégorie distincte. Cependant, cette méthode est efficace lorsque les catégories ont une relation ordinale ou lorsqu'il n'y a pas trop de valeurs distinctes, car elle peut introduire des biais si les entiers attribués suggèrent une relation numérique inexistante entre les catégories. La performance des modèles testés plus tard confirmera bien l'inadéquation de choisir cette méthode d'encodage.

Nous nous sommes alors penché sur une autre méthode plus appropriée : le One Hot Encoding. Cette méthode consiste à créer des colonnes nouvelles pour chaque valeur de nos variables catégorielles et à attribuer des 1 et des 0 dans ces colonnes pour indiquer la présence ou l'absence de chaque catégorie dans les observations originales. Pour mettre en place cette méthode, nous avons d'abord appliqué la fonction get_dummies() de pandas. Cette fonction permet d'appliquer directement le OneHotEncoding à notre dataframe. La seule différence est que la valeur attribuée n'était pas 0 ou 1 mais False et True, ce qui ne change pas grand chose pour les modèles. Cependant, étant donné que le client avait le choix de 24 items (par conséquent 24 make et 24 model), nous nous sommes retrouvés avec par exemple deux colonnes portant sur les mêmes produits mais seul le numéro de l'item changeait : item1_COMPUTER et item2_COMPUTER. L'existence de ces deux colonnes est inutile, donc il a fallu les fusionner ensemble. Après quelques manipulations de listes pour stocker les valeurs uniques des colonnes item, make et model, nous avons finalement fusionné toutes les colonnes portant sur le même produit, grâce à la fonction .any(). Une fois le problème résolu, il n'y avait plus de doublons, ni pour item, ni pour make et ni pour model. Le dataframe était finalement bien encodé.

### c) Imputation des NaN

A la fin de l'encodage il nous restait plus qu'à imputer toutes les valeurs manquantes restantes dans le dataframe (dans les colonnes cash_price et Nbr_of_prod_purchas) grâce à la fonction .fillna(False), ce qui remplace tous les NaN par des False.

## IV) Choix des modèles :

### a) Méthode d'évaluation

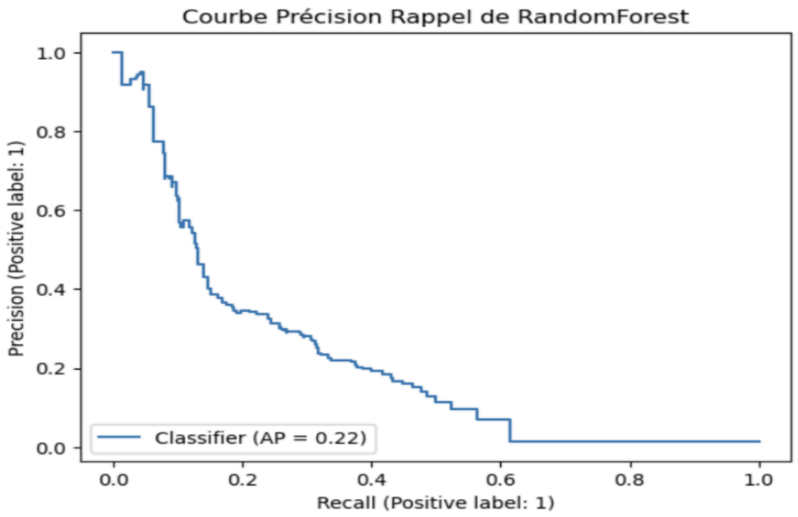
Maintenant que nous avons obtenu un jeu de données traitable par un modèle. Il faut évaluer différents modèles pour sélectionner le meilleur. Pour cela, on fait un train test split, c'est-à-dire qu'on va réserver un dataset pour entraîner le modèle dessus puis évaluer la performance sur un autre.

Pour choisir les modèles, on va définir une méthode d'évaluation. Généralement, le score de précision est la métrique utilisée à cet effet. Cependant, dans notre cas, le score de précision n'est pas adapté car la répartition des cibles n'est pas équilibrée, ce qui peut conduire à une évaluation trompeuse de la performance du modèle en favorisant la classe majoritaire au détriment des classes minoritaires. La métrique recommandée est l'aire sous la courbe Précision-Rappel. L'aire sous la courbe Précision-Rappel, PR-AUC, représente la capacité d'un modèle à distinguer entre les classes positives et négatives en tenant compte de l'équilibre entre la précision et le rappel, fournissant ainsi une mesure plus informative de la performance du modèle, particulièrement utile en présence de données déséquilibrées. Plus l'aire sous la courbe est élevée, plus le modèle est capable de maintenir un haut niveau de précision tout en atteignant un rappel élevé, ce qui indique une meilleure capacité à discriminer les classes positives des classes négatives, même dans des scénarios où les données sont déséquilibrées. Pour calculer cette aire, nous allons utiliser la fonction average_precision_score de sklearn.metrics.

### b) Test des modèles

Après avoir défini notre méthode d'évaluation, il nous reste plus qu'à tester notre dataset sur différents modèles en fonction du critère PR-AUC. Pour cela, nous avons utilisé les paramètres par défaut sur les modèles suivants : RandomForest, DecisionTree et LogisticRegression. Le résultat est unanime : 0.22 pour le RandomForest, 0.07 pour le

DecisionTree et 0.01 pour le LogisticRegression. Nous allons donc chercher à optimiser les hyperparamètres du RandomForest.




## V) Optimisation & présentation des résultats :

Notre premier résultat a été obtenu grâce à un RandomForest Classifier (hyperparamètres par défaut) et du label encoding : 0.036. Suite à ce résultat très peu encourageant, nous avons décidé d'abandonner le label encoding qui n'était pas la méthode d'encoding adéquate pour nos données. Grâce au One Hot Encoding, nous avons obtenu un résultat bien meilleur : 0.186 toujours sans avoir supprimé de colonnes par rapport aux valeurs manquantes. Suite à cela, nous avons décidé d'optimiser les hyperparamètres et de réduire la dimension de notre dataframe en supprimant les colonnes où énormément de valeurs étaient manquantes (le % variait, mais le plus optimal était 99%). Pour optimiser les hyperparamètres : nous avons réalisé un train-test-split sur notre x_train de façon à pouvoir évaluer notre score grâce à average-precision-score. Pour trouver les hyperparamètres optimaux, nous avons utilisé RandomSearchCV plutôt que GridSearchCV. La différence est que GridSearchCV va tester toutes les combinaisons possibles de toutes les valeurs des hyperparamètrès tandis que RandomSearchCV va en sélectionner aléatoirement (donc beaucoup moins long et coûteux). Je précise que dans les deux cas, on utilise la Cross-Validation : les données sont divisées en plusieurs sous-ensembles (folds), et le modèle est entraîné sur une partie des sous-ensembles et évalué sur les restants. On moyenne ensuite les résultats. Nous avons dans un premier temps obtenu un score de 0.26 en supprimant les colonnes où 99.9% des valeurs étaient manquantes. Enfin, notre résultat final, obtenu avec les meilleurs hyperparamètres et la suppression des colonnes avec cette fois-ci 99% des valeurs manquantes : 0.268.

NOTA BENE : A quelques jours de la fin du challenge, nous avons tenté de fit un autre modèle : GradientBoostingClassifier. Ce dernier est particulièrement intéressant en raison de sa capacité à modéliser des relations complexes et non linéaires, à gérer les ensembles de données déséquilibrés, et à offrir des performances élevées en termes de précision. Cet algorithme peut identifier des patterns subtils dans les transactions, cruciales pour la détection de fraudes, tout en minimisant les faux positifs et négatifs. En optimisant ses hyperparamètres et toujours en supprimant les colonnes où 99% des valeurs étaient manquantes, nous avons obtenu notre meilleure score : 0.2813.