ENSAE Paris

Projet de Statistiques Appliquées

Implémentation de méthodes de Machine Learning non supervisées pour la détection d'anomalies sur des datasets de credit

BELKHATIR Adam
DUONG Simon
JDEY Ons
MORAINE Louis

Encadrant : Alexis VIGNARD - Société Générale

May 20, 2025

Contents

1 Introduction

1.1 Contexte et motivation

Dans le cadre de l'analyse statistique des données, la détection d'anomalies vise à identifier des observations qui s'écartent significativement du comportement général d'un ensemble. Lorsqu'on ne dispose d'aucune information a priori sur ce qui constitue une observation "normale" ou "anormale", les méthodes non supervisées deviennent essentielles. Elles permettent d'explorer la structure interne des données sans recours à des labels, en se basant sur des critères tels que la densité locale ou la distance aux voisins proches par exemple. Ces approches sont particulièrement précieuses dans des contextes où les anomalies sont rares, mal définies, ou variables dans le temps.

Cependant, l'absence de supervision rend l'évaluation des performances de ces méthodes particulièrement complexe. En effet, sans vérité terrain, il devient difficile de dire si les points identifiés comme atypiques sont réellement pertinents. Contrairement à un cas supervisé où l'on peut calculer des métriques standard à partir de labels connus[1], les approches non supervisées doivent souvent s'appuyer sur des indicateurs indirects (stabilité des scores, cohérence entre les méthodes, etc.). Cette difficulté méthodologique constitue un défi central pour la fiabilité et l'adoption des techniques non supervisées dans les applications pratiques.

Dans le secteur financier, la détection d'anomalies joue un rôle crucial pour l'évaluation du risque de crédit. Elle intervient dans des domaines critiques tels que la détection de fraudes et la prévention des défauts de paiement. Ces événements, bien que rares, peuvent entraîner des conséquences financières lourdes pour les institutions bancaires, qu'il s'agisse de pertes directes, de sanctions réglementaires, ou d'atteintes à la réputation.

Cette mission est particulièrement difficile car, bien que des cas d'anomalies sont parfois identifiés a posteriori (comme des fraudes avérées ou des défauts de paiement enregistrés), ces vérités terrain sont extrêmement rares et souvent peu représentatives. Les institutions doivent ainsi s'appuyer sur des approches capables d'identifier automatiquement des comportements atypiques, sans supervision explicite.

Les méthodes statistiques classiques[2] montrent rapidement leurs limites lorsqu'il s'agit d'analyser des données multivariées complexes. Elles peinent notamment à capturer les interactions non-linéaires entre variables, ou à s'adapter à des distributions évolutives. Face à ces défis, le Machine Learning s'est imposé comme une alternative robuste et flexible, capable d'identifier automatiquement des comportements atypiques sans hypothèses fortes sur la structure des données. Il offre des performances supérieures dans la détection d'anomalies en situation réelle.

1.2 Problématiques

L'objectif de ce projet est alors d'explorer et de développer une réflexion autour des problématiques suivantes :

- Comment détecter efficacement des outliers dans des données multivariées issues du secteur bancaire, et identifier les modèles les plus performants selon le type de données ?
- Comment évaluer ces modèles dans un contexte non-supervisé, où il n'existe pas de vérité terrain ?

Nous allons tout d'abord définir tous les concepts théoriques sur lesquels se fonde notre étude. Puis nous présenterons les trois datasets de risque de crédit qui nous serviront de support. Enfin, nous

[1] Voir 2.5.1
[2] Voir 2.2

détaillerons notre pipeline de travail, avant de présenter nos résultats et d'aboutir, *in fine*, à un benchmark des différentes méthodes non supervisées de détection d'anomalie.

2 Cadre théorique

2.1 Définition des anomalies

Une **anomalie** (ou **outlier**) est une observation qui s'écarte significativement du comportement général du jeu de données. D'un point de vue statistique, elle viole les tendances de la majorité de l'échantillon, que ce soit en termes de distribution, de distance, de densité ou de structure. Sa détection dépend fortement du contexte d'analyse.

2.2 Méthodes statistiques classiques

Les méthodes statistiques classiques constituent une première approche intuitive et simple pour la détection d'anomalies. Bien qu'historiquement conçues pour l'analyse univariée, certaines d'entre elles peuvent être généralisées à des contextes multivariés.

a. Z-score

Pour des données univariées, le Z-score mesure l'écart d'une observation à la moyenne d'une variable, en nombre d'écarts-types :

$$Z_i = \frac{x_i - \mu}{\sigma}$$

Une observation est considérée comme atypique si son Z-score dépasse un seuil (généralement, on utilise $|Z| > 3$).

Limites : Hypothèse de normalité des données et ne tient pas compte des corrélations entre variables.

b. IQR et boîte à moustaches

L'écart interquartile (IQR) est une méthode non paramétrique utilisée pour détecter les valeurs extrêmes sans hypothèse sur la distribution. Il est défini comme :

$$IQR = Q_3 - Q_1$$

où Q_1 et Q_3 sont respectivement les premier et troisième quartiles. Les observations situées en dehors de l'intervalle $[Q_1 - 1.5 \times IQR, Q_3 + 1.5 \times IQR]$ sont considérées comme potentiellement aberrantes.

La boîte à moustaches (boxplot) est la représentation graphique associée à cette méthode. Elle visualise les quartiles et identifie les points extrêmes.

Avantages :Méthode robuste aux distributions non normales et facile à visualiser à l'aide d'une boîte à moustaches (boxplot).

Limites : Uniquement univariée.

c. Population Stability Index (PSI)

Le *Population Stability Index* (PSI) est une métrique permettant de quantifier le changement de distribution d'une variable entre deux populations : généralement une population de référence (ex. données historiques ou normales) et une population actuelle (ex. données récentes ou anomalies). Le PSI est défini par la formule suivante :

$$PSI = \sum_{i=1}^{n} (p_i - q_i) \cdot \ln\left(\frac{p_i}{q_i}\right)$$

où p_i et q_i représentent respectivement la proportion d'observations dans la classe i pour les distributions de référence et actuelle.

Le PSI est largement utilisé en gestion du risque pour surveiller les dérives de variables dans le temps, notamment dans le suivi de la stabilité des modèles de scoring.

Utilisation en multivarié : Dans notre approche, nous avons calculé le PSI pour chaque variable indépendamment, puis pris la moyenne des valeurs obtenues .

Utilité dans ce projet : On a utilise le PSI pour comparer les distributions entre l'ensemble des données et les observations identifiées comme anomalies. Cela permet d'évaluer dans quelle mesure ces dernières s'écartent du comportement "normal". Ensuite, on l'a utilise comme critère de sélection dans notre générateur d'anomalies synthétiques (voir 4.2.2)

2.3 Méthodes Machine Learning

2.3.1 K-Nearest Neighbors (Sridhar Ramaswamy and Shim [2000])

Cet article a été rédigé par Sridhar Ramaswamy, Rajeev Rastogi et Kyuseok Shim et publié le 16 mai 2000. Les auteurs y présentent trois algorithmes : le Nested-Loop Algorithm, l'Index-Based Algorithm et le Partition-Based Algorithm. Ces algorithmes reposent sur la notion de plus proche voisin et permettent d'identifier des valeurs aberrantes. Pour cela, ils classent chaque point en fonction de sa distance par rapport à son voisin le plus proche, notée D, et déclarent que les points les plus élevés de ce classement sont des valeurs aberrantes. Les points avec des valeurs de D plus élevées ont des voisinages plus clairsemés et sont donc généralement des valeurs aberrantes plus fortes que les points appartenant à des clusters denses qui auront tendance à avoir des valeurs de D plus faibles. Ces alogirthmes prennent en entrée deux entiers n (le nombre de valeurs aberrantes recherchées) et k (le nombre de voisins considérés).

Définition. *(Valeur abberantes) Pour un entier k et un point p, notons $D^k(p)$ la distance au k-ième voisin le plus proche de p. Les n premiers points avec les valeurs D^k maximales sont considérés comme des valeurs aberrantes.*

Fonctionnement des algorithmes

Nested-Loop Algorithm Le Nested-Loop Algorithm effectue le calcul de $D^k(p)$ pour chaque point p de la facon suivante : on initialise une liste des k points les plus proches de p, puis pour chaque point q de la base de données qui est considéré, une vérification est effectuée pour voir si $dist(p, q)$ est plus petite que la distance du k-ème voisin le plus proche trouvé jusqu'à présent. Si le résultat est positif, q est inclus dans la liste des k voisins les plus proches de p (si la liste contient plus de k voisins, alors le point le plus éloigné de p est supprimé de la liste).

Index-Based Algorithm L'Index-Based Algorithm optimise les calculs en stockant les points dans un arbre. Pour chaque point p, lors du calcul de $D^k(p)$, il élimine les nœuds trop loin de p pour posséder un plus proche voisin de p. Supposons que nous ayons calculé $D^k(p)$ pour un point p en examinant un sous-ensemble des points d'entrée. La valeur que nous avons est clairement une limite supérieure pour le $D^k(p)$ réel. Si la distance minimale entre p et le MBR (minimum bounding rectangle) d'un nœud dans l'arbre dépasse la valeur $D^k(p)$ que nous avons actuellement, aucun des points du sous-arbre enraciné sous le nœud ne sera parmi les k voisins les plus proches de p. Cette optimisation nous permet d'élaguer des sous-arbres entiers contenant des points non pertinents pour la recherche des k voisins les plus proches de p.

Partition-Based Algorithm Cet algorithme partitionne l'espace des données à l'aide d'un algorithme de clustering, puis élague les partitions. Les partitions trop denses ne peuvent pas contenir de valeur aberrantes et sont donc éliminées. Ensuite, le calcul de $D^k(p)$ se fait uniquement pour les points p appartenant aux partitions candidates.

Avantages et limites

- L'inconvénient du Nested-Loop Algorithm est que cette approche nécessite un $O(N^2)$ calculs de distance. Le nombre de calculs de distance peut être considérablement réduit en utilisant un index spatial comme un arbre.
- C'est l'avantage de l'Index-Based Algorithm qui exploite les structures d'indexation et améliore l'efficacité en élaguant des sous-arbres non pertinents.
- Enfin, le Partition-Based Algorithm se distingue par sa capacité à éviter des calculs inutiles en divisant l'espace des données. Cette stratégie d'élagage permet de réduire considérablement le nombre de calculs nécessaires et peut accélérer la détection des valeurs aberrantes. Les performances de cet algorithme ne se dégradent pas à mesure que k augmente. En outre, il s'adapte mieux que les autres aux dimensions supérieures.

2.3.2 Local Outlier Factor (Markus M. Breunig and Sander [2000])

Le LOF (Local Outlier Factor) est un algorithme conçu pour détecter des valeurs aberrantes locales dans des ensembles de données multidimensionnels. Il mesure le degré d'isolement d'un point par rapport à son voisinage immédiat, ce qui le rend particulièrement utile pour les données tabulaires ou multidimensionnelles où les densités locales varient.

Fonctionnement de l'algorithme

LOF repose sur plusieurs notions clés. Tout d'abord, la **k-distance** d'un point correspond à la distance entre p et un objet o. À partir de cette distance, on définit le **k-distance neighborhood**, qui regroupe tous les voisins situés à une distance inférieure ou égale à la k-distance :

$$N_k\text{-distance}(p) = \{q \in D \setminus \{p\} \mid d(p,q) \leq k\text{-distance}(p)\}$$

Ensuite, la **reachability distance** est une mesure ajustée de proximité entre deux points, prenant en compte la k-distance de l'objet voisin o :

$$\text{reach-dist}_k(p,o) = \max\{k\text{-distance}(o), d(p,o)\}.$$

La **Local Reachability Density (LRD)** d'un point p est une mesure de la densité locale d'un point basée sur les distance de portée des éléments de $N(p)$, l'ensemble des $MinPts$-voisins de p :

$$LRD(p) = \left(\sum_{o \in N(p)} \text{reach-dist}(p,o) \right)^{-1}$$

Enfin, le **Local Outlier Factor (LOF)** est le ratio entre la densité locale d'un point et celle de ses voisins :

$$LOF(p) = \frac{1}{|N(p)|} \sum_{o \in N(p)} \frac{LRD(o)}{LRD(p)}$$

L'algorithme suit les étapes suivantes :

1. Initialisation des paramètres ($MinPts$ - le nombre minimal de voisins pris en compte pour estimer la densité locale d'un point, ensemble de données).
2. Calcul de la k-distance et des distances de portée pour chaque point.
3. Calcul de la LRD pour chaque point.
4. Calcul du LOF en comparant la LRD d'un point à celle de ses voisins.
5. Identification des anomalies : un point est considéré comme aberrant si son LOF > 1.

Résultats

Sur les données synthétiques, LOF identifie avec précision les anomalies dans des clusters de densités variées. Les points en périphérie ont des LOF proches de 1, tandis que les outliers ont des LOF $> 1,5$. LOF est efficace pour les données de faible à moyenne dimension (2D à 5D), mais sa performance se dégrade en haute dimension ($>$ 10D) en raison de l'augmentation de la complexité calculatoire. Le choix de $MinPts$ influence fortement les résultats. Une faible valeur entraîne une grande variabilité des LOF, tandis qu'une valeur élevée stabilise les résultats.

Avantages et limites

Au final, les avantages de cet algorithme sont :

- Approche graduelle et non binaire: Contrairement aux méthodes classiques qui classent un point uniquement comme normal ou anormal (binaire), LOF attribue un score d'anomalie.
- Perspective locale adaptée aux variations de densité:Il détecte des anomalies même si les données sont regroupées en clusters de différentes densités, ce qui est difficile pour d'autres méthodes globales.

Et ses limites sont :

- Sensibilité au choix de MinPts.
- Complexité calculatoire élevée pour les données volumineuses ou de haute dimension.($> 10D$).

2.3.3 Isolation Forest (Fei Tony Liu and Zhou [2008])

L'**Isolation Forest** est une méthode efficace pour détecter les anomalies en isolant les points inhabituels plus rapidement que les points normaux. Contrairement aux approches basées sur la distance ou la densité, elle repose sur la construction d'arbres binaires récursifs appelés *Isolation Trees*. Chaque arbre est construit en divisant les données aléatoirement sur des attributs et des seuils. La propriété clé de cette méthode est que les anomalies, qui sont rares et éloignées, nécessitent moins de divisions pour être idsolées que les points normaux, en raison de la densité élevée de ces derniers.
L'Isolation Forest utilise des sous-échantillons pour atténuer les effets de *swamping* et *masking* dans la détection des anomalies. Le *swamping* se produit lorsque des points normaux sont trop proches des anomalies, rendant leur séparation difficile. Le *masking* survient quand des anomalies denses rendent leur détection plus complexe. Grâce à des sous-échantillons, iForest crée des modèles partiels, permettant une meilleure isolation des anomalies et une performance accrue, notamment en réduisant l'interférence des points normaux.

Fonctionnement de l'algorithme

L'algorithme construit d'abord un ensemble d'arbres en divisant les données de manière aléatoire sur un attribut et un seuil jusqu'à ce qu'un point soit isolé ou qu'une profondeur maximale soit atteinte. Une caractéristique importante de cette approche est que les anomalies ont une profondeur moyenne plus faible que les points normaux, car elles sont isolées plus rapidement. La détection d'anomalies repose ensuite sur le calcul du score d'anomalie $s(x)$, défini par la formule suivante :

$$s(x) = 2^{-\frac{E(h(x))}{c(n)}}$$

où $E(h(x))$ représente la longueur moyenne du chemin d'un point donné dans l'ensemble des arbres, et $c(n)$ une constante normalisant la profondeur en fonction du nombre total de points n. Un score élevé indique un point isolé rapidement, donc une potentielle anomalie.

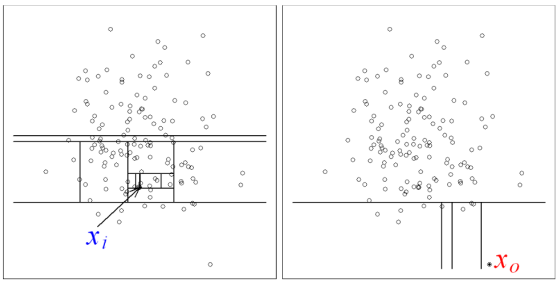

Figure 1: Les anomalies (droite) sont isolées plus rapidement que les points normaux (gauche).[3]

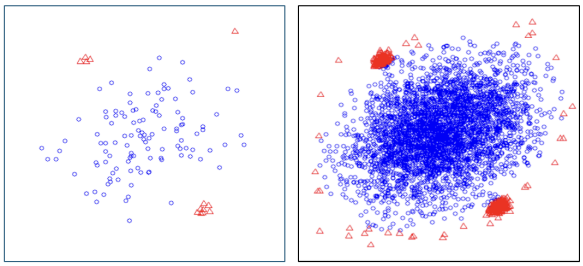

Figure 2: Illustration du *swamping* : à gauche, les anomalies (rouge) sont bien isolées des points normaux (bleu), tandis qu'à droite, elles sont noyées dans une forte densité de points normaux.[4]

Avantages et limites

Les avantages de l'Isolation Forest sont :

- L'algorithme est rapide et adapté aux grands ensembles de données grâce à sa faible complexité computationnelle.
- Il ne nécessite pas de calculs complexes de distance et peut identifier des anomalies même dans des données où la densité varie.
- Il est largement utilisé pour des applications diverses : surveillance des fraudes, détection d'intrusions, identification d'erreurs industrielles.

Et ses limites sont :

- Le nombre d'arbres et la profondeur maximale influencent fortement les performances et doivent être soigneusement ajustés.
- Lorsque la dimension des données est très élevée, l'efficacité de l'algorithme diminue car les divisions aléatoires perdent leur pertinence.

2.3.4 One-Class Support Vector Machine (Nguyen and Vien [2018])

Contrairement à un SVM classique, un one-class SVM (OC-SVM) ne cherche à identifier qu'une seule classe. Pour cela, il cherche un hyperplan qui sépare au mieux les points de l'origine, cette dernière étant traitée comme un élément représentatif d'une seconde classe. Les points au-delà de cet hyperplan sont considérés normaux, tandis que les autres sont classées comme des anomalies. Cependant, une séparation linéaire n'est souvent pas optimale. Pour pallier ce problème, il est possible d'appliquer une transformation non linéaire (noyau) sur les données, de l'espace de départ vers un espace de plus grande dimension dans lequel elles sont linéairement séparables. Dans l'espace de départ, cela apparaîtra comme une séparation non linéaire. L'OC-SVM est une méthode très performante pour la détection d'anomalie.

Fonctionnement de l'algorithme

L'OC-SVM a pour objectif de résoudre le problème d'optimisation suivant :

$$\min_{\mathbf{w},\rho} \frac{1}{2}\|\mathbf{w}\|^2 - \rho + \frac{1}{\nu n}\sum_{i=1}^{n} \max(0, \rho - \mathbf{w}^T z(x_i)). \tag{1}$$

où x_i désigne un vecteur de caractéristiques, $\mathbf{w}$ le vecteur des poids associés aux dimensions, ρ la distance minimale entre l'hyperplan et l'origine, ν le paramètre de régularisation qui définit une borne supérieure sur la fraction maximale d'anomalie détectée (i.e. la fraction d'anomalies que l'on s'autorise), et z la fonction projettant les inputs depuis l'espace d'origine vers un espace de plus grande dimension où ils peuvent être

[4]Fei Tony Liu and Zhou [2008]
[4]Fei Tony Liu and Zhou [2008]

séparés linéairement. Cette dernière fonction correspond à l'approximation d'un noyau RBF (Radial Basis Function), qui est le noyau le plus utilisé dans les OC-SVM. Les rôles des termes de la fonction objective sont les suivants :

- $\frac{1}{2}\|\mathbf{w}\|^2$: pénalise la complexité de l'hyperplan;
- $-\rho$: maximise la distance entre l'hyperplan et l'origine de l'espace projeté;
- $\frac{1}{\nu n}\sum_{i=1}^{n}\max(0, \rho - \mathbf{w}^T\phi(x_i))$: pénalise les inputs trop proches de l'origine, c'est à dire ceux considérés comme des anomalies.

En notant $g(x) = \mathbf{w} \cdot z(x_i) - \rho$, la fonction de décision est $f(x) = \text{sgn}(g(x))$, qui vaut 1 lorsque l'input est normal et -1 lorsque c'est une anomalie.

Résultats

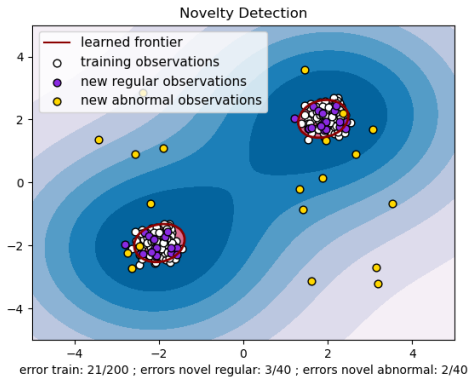

Figure 3: Détection d'anomalies avec un OC-SVM.[5]

On observe sur cette image qu'aucune séparation linéaire n'est possible pour détecter des anomalies, du moins elle aurait été inutile. Grâce au noyau RBF, il est possible d'établir une frontière non linéaire, qui apparaît ici sous la forme de deux cercles rouges. Tout les points à l'intérieur de ces cercles sont alors considérés comme normaux, et les autres sont classés comme des anomalies. Il est possible de comprendre visuellement le rôle du paramètre ν de la manière suivante : plus sa valeur est élevée, plus les diamètres des cercles seront petits, ce qui classera plus de points comme des anomalies, et inversement.

Avantages et limites

Les avantages de cet algorithme sont :

- Grâce au noyau, il peut traiter des relations non linéaires, et peut s'adapter à des données dans des espaces de dimension très élevée.
- Efficacité sur des données avec des densités variées, car il ne suppose pas que les anomalies sont isolées.

Et ses limites sont :

- Complexité computationnelle élevée en raison de l'optimisation quadratique.
- Sensibilité aux choix des hyperparamètres, en particulier ν et le type de noyau.

2.4 Méthode Deep Learning

2.4.1 Autoencodeur (Mayu Sakurada [2014])

L'autoencodeur est un réseau de neurones non supervisé conçu pour réduire la dimension d'un jeu de données tout en préservant au mieux ses caractéristiques essentielles. Sous l'hypothèse que les variables sont corrélées, l'autoencodeur peut être appliqué directement à la détection d'anomalies. En effet, il apprend à projeter les données dans un espace de plus petite dimension où les échantillons normaux et les anomalies se distinguent clairement.

[5]Source : scikit-learn, One-class SVM with non-linear kernel (RBF)

Fonctionnement de l'algorithme

Notons $\{\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(m)\}$ le vecteur des entrées, où $\mathbf{x}(i) \in \mathbb{R}^D$, m est la taille de l'échantillon et D sa dimension. L'objectif de l'autoencodeur est de réduire la dimension de l'échantillon en passant successivement les entrées dans des couches de neurones (encodeur), puis de reconstruire ces entrées sous la forme du vecteur $\{\hat{\mathbf{x}}(1), \hat{\mathbf{x}}(2), \ldots, \hat{\mathbf{x}}(m)\}$ de sorte que les échantillons obtenus soient aussi fidèles que possible aux entrées (décodeur). L'entraînement de l'autoencodeur consiste à minimiser la fonction objective suivante :

$$J(\mathbf{W}, \mathbf{b}) = \underbrace{\frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \left\| \mathbf{x}(i) - \hat{\mathbf{x}}(i) \right\|^2 \right)}_{\text{erreur de reconstruction}} + \underbrace{\frac{\lambda}{2} \sum_{l=1}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2}_{\text{terme de régularisation}}$$

où $\mathbf{W}$ et $\mathbf{b}$ sont les paramètres de poids et de biais respectivement, λ est le facteur de régularisation, n_l est le nombre de couches dans le réseau, et s_l est le nombre de neurones dans la couche L_l. Minimiser l'erreur de reconstruction sert à optimiser la capacité de l'autoencodeur à apprendre les caractéristiques du dataset, et minimiser le terme de régularisation pénalise les poids $\mathbf{W}$ pour contrôler la complexité du modèle et éviter l'overfitting. Les poids et le biais interviennent à chaque couche lorsque les entrées sont transformées via la relation :

$$a_i^{(l)} = f \left(\sum_{j=1}^{n} W_{ij}^{(l-1)} a_j^{(l-1)} + b_i^{(l)} \right)$$

où $\boldsymbol{a}^{(l)}$ est la représentation de l'entrée dans la couche L_l. Par exemple, $\boldsymbol{a}^{(1)} = \mathbf{x}$ et $\boldsymbol{a}^{(n_l)} = \hat{\mathbf{x}}$. La fonction d'activation f est un hyperparamètre essentiel, car elle introduit de la non-linéarité et permet au modèle d'apprendre des relations complexes. Dans le cadre de la détection d'anomalie, nous séparons notre dataset en un test sample et un training sample. Dans l'idéal, ce dernier doit être composé uniquement de données normales. Nous obtenons $\mathbf{W}$ et $\mathbf{b}$ en entraînant l'autoencodeur sur le training sample, puis nous appliquons l'autoencodeur au test sample. Les anomalies ont une erreur de reconstruction plus élevé que les entrées normales du fait qu'elles n'ont pas la même structure. En ce sens, l'erreur de reconstruction est utilisée comme score d'anomalie :

$$Err(i) = \sqrt{\sum_{j=1}^{D} \left(\mathbf{x}_j(i) - \hat{\mathbf{x}}_j(i) \right)^2}$$

Une extension possible de l'autoencodeur est le denoising autoencodeur. Brièvement, les données d'entrées sont artificiellement bruitées de sorte à forcer l'autoencodeur à extraire les caractéristiques importantes pour reconstruire le dataset malgré la corruption, et la dimension de l'espace latent est augmentée afin de capturer plus de structures et de motifs sous-jacents. Cela le rend plus robuste qu'un autoencodeur classique.

Résultats

Cet article compare l'autoencodeur (AE), le denoising autoencodeur (dAE), l'analyse en composantes principales linéaire (LPCA) et non linéaire (KPCA). Ces dernières sont des méthodes de machine learning de réduction de dimension basées sur des transformations linéaires ou non linéaires des données. Un dataset artificiel est créé à l'aide du système de Lorenz, contient 1000 observations et est de dimension 25. Deux datasets réels de télémétrie spatiale sont utilisés, respectivement de dimension 17 et 106. Les résultats sont les suivants :

- **Dataset artificiel** : AE, dAE et KPCA reconnaissent les anomalies introduites dans le dataset, tandis que LPCA n'identifie rien. De plus, dAE performe légèrement mieux que l'AE, confirmant sa pertinence.
- **Dataset réel 1** : AE et LPCA ont les mêmes résultats. dAE est meilleur que ces deux derniers.
- **Dataset réel 2** : dAE ne parvient pas à améliorer les résultats de l'AE. Comme pour le dataset précédent, AE et dAE performent au moins autant que KPCA, alors que KPCA nécessite de lourds calculs.

Avantages et limites

Les avantages de l'autoencodeur et du denoising autoencodeur sont :

- Permettent une réduction de dimension non linéaire, et capturent donc des structures complexes.
- Ont un coût de calcul inférieur à KPCA, pour des résultats au moins équivalents.
- dAE est plus robuste aux bruits et peut améliorer les performances de l'AE.

Et ses limites sont :

- Leur performance dépend fortement des hyperparamètres (nombre de couches, facteur de régularisation, etc.).
- Peuvent nécessiter un grand volume de données pour être bien entraînés (risque d'overfitting).

2.4.2 Création de notre propre autoencodeur

Nous avons jugé pertinent de programmer notre propre autoencodeur[6] (AE) avec la librairie `TensorFlow` afin de comparer ses performances avec celle de l'AE de `PyOD`. Par souci de simplicité, nous ne laissons pas la possibilité à l'utilisateur de choisir le nombres de couches latentes et le nombres de neurones par couche. Nous fixons l'architecture de l'AE de la manière suivante : l'espace latent est environ quatre fois plus petit que l'espace d'entrée et chacun des blocs (encodeur et décodeur) est composé de deux couches de neurones, dont la couche cachée comporte environ deux fois plus de neurones que l'espace latent. Les derniers éléments fixés sont les fonctions d'activation. Nous choisissons la fonction ReLU pour toutes les couches sauf celle de sortie pour laquelle nous utiliserons la fonction sigmoid (notée σ). Elles sont définies par :

$$\forall x \in \mathbb{R}, \quad \mathrm{ReLU}(x) = \max(0, x) \quad \text{et} \quad \sigma(x) = \frac{1}{1 + e^{-x}}$$

Nous prendrons soin de scaler nos données à l'aide du Min-Max Scaler pour qu'elles soient bien comprises dans l'intervalle $[0, 1]$ et pour que la fonction de sortie ait un sens. La fonction de perte utilisée sera la Mean Squared Error (MSE), qui est standard.
Notre modèle AE considère ensuite plusieurs paramètres en argument, que l'utilisateur peut modifier à sa guise mais qui ont tous des valeurs par défaut.

- `contamination` (float dans $[0, 1]$) : proportion d'outliers dans le dataset (par défaut = 0.1) ;
- `learning_rate` (float) : taille des pas de l'optimiseur (par défaut = 0.001) ;
- `epochs` (int): nombre de fois que le dataset est parcouru pendant l'entraînement (par défaut = 10) ;
- `batch_size` (int) : nombre de subdivisions du dataset pour l'entraînement (par défaut = 32) ;
- `early_stopping` (bool) : possibilité d'arrêter prématurément l'entraînement si l'erreur de reconstruction ne réduit plus suffisamment à chaque *batch* (par défaut = `True`) ;
- `patience` (int) : nombre de fois d'affilée où l'erreur de reconstruction peut ne pas réduire suffisamment avant l'*Early Stopping* (par défaut = 5) ;
- `min_delta` (float) : amélioration considérée comme suffisante pour l'erreur de reconstruction (par défaut = e^{-3}).

Les valeurs par défaut (sauf pour `early_stopping`, `patience` et `min_delta`) sont alignées avec celles de l'AE de `PyOD`. Enfin, notre AE possède les méthodes et attributs suivants :

- `epoch_losses_` : vecteur des erreurs de reconstruction par `epoch` lors de l'entraînement ;
- `(train_)reconstruction_` : vecteur de reconstruction $\hat{\mathbf{x}}$ (disponible pour le train ou test set) ;

[6] Voir Code 1

- `(train_)reconstruction_errors_` : erreur de reconstruction (disponible pour le train ou test set) ;
- `(train_)outliers_` : vecteur de booléens indiquant les outliers détectés par le modèle (disponible pour le train ou test set) ;
- `decision_function(dataset)` : calcule et renvoie l'erreur de reconstruction (score) sur `dataset` en utilisant le modèle entraîné ;
- `predict(dataset)` : calcule et renvoie un vecteur de booléens indiquant les outliers détectés sur `dataset` en utilisant le modèle entraîné ;
- `fit(dataset)` : entraîne l'AE sur `dataset` et initialise les attributs `epoch_losses_`, `train_reconstruction_`, `train_reconstruction_errors_` et `train_outliers_`.

Le préfixe `train_` indique que les attributs sont calculés sur le train set. Les attributs sans préfixe concernent les résultats obtenus sur le dernier test set auquel l'AE a été appliqué.

2.5 Évaluation en apprentissage supervisé, et non supervisé

2.5.1 Apprentissage supervisé

Matrice de confusion

La matrice de confusion est une table croisée qui résume les prédictions du modèle par rapport à la réalité terrain. Elle se décompose en quatre éléments :

- **Vrais Positifs (TP) :** Anomalies correctement détectées.
- **Faux Positifs (FP) :** Données normales classées à tort comme anomalies.
- **Faux Négatifs (FN) :** Anomalies non détectées.
- **Vrais Négatifs (TN) :** Données normales correctement identifiées.

Métriques dérivées

À partir de la matrice de confusion, on calcule plusieurs indicateurs clés :

- **Accuracy :**
$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$
- **Precision :**
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
- **Recall :**
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
- **F1-score :**
$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$
- **AUC-ROC (Area Under the ROC Curve)** :
La courbe ROC trace le taux de vrais positifs (Recall) en fonction du taux de faux positifs $\frac{\text{FP}}{\text{FP+TN}}$ à différents seuils. L'AUC mesure l'aire sous la courbe ROC.

2.5.2 Apprentissage non supervisé

L'évaluation des modèles dans un contexte non supervisé est une tâche complexe. Notre approche initiale consistait à exploiter le consensus entre modèles, en comptant le nombre de fois où différents algorithmes s'accordaient sur les points considérés comme atypiques. Bien que simple et intuitive, cette méthode s'est révélée biaisée. Cette limite nous a conduits à explorer la littérature afin d'identifier des critères plus robustes. C'est dans ce cadre que nous nous sommes tournés vers les travaux de Nicolas Goix.

Afin d'évaluer des modèles de détection d'anomalies dans un cadre non supervisé, il est nécessaire de recourir à des alternatives aux critères classiques fondés sur les courbes ROC et PR (précision-rappel). Nous considérons deux critères développés par Nicolas Goix, basés sur les courbes Excess-Mass (EM) et Mass-Volume (MV), qui peuvent être calculés à partir de données non étiquetées. Toutefois, leur estimation n'est pas directement applicable en grande dimension. Pour remédier à cela, nous introduirons une méthodologie également proposée par Nicolas Goix, reposant sur un sous-échantillonnage des variables, afin de permettre l'utilisation de ces critères dans des contextes de grande dimension.

Les modèles de détection d'anomalies ne se limitent pas à retourner un label binaire : ils calculent d'abord une fonction de score, qui est ensuite convertie en prédiction binaire. Il s'agit d'une fonction $s : \mathbb{R}^d \to \mathbb{R}_+$ telle que, plus $s(x)$ est faible, plus l'observation x est considérée comme anormale. Nous nous intéressons à l'évaluation de ces fonctions de score.

Nous considérons des données dans $\mathbb{R}^d$, supposées issues d'une distribution sous-jacente F admettant une densité $f : \mathbb{R}^d \to \mathbb{R}_+$, toutes deux inconnues.

Une fonction de score s est considérée comme pertinente si l'ordre qu'elle induit est proche de celui induit par la densité f, en supposant que les anomalies se situent dans la queue de la distribution des données. Étant donné que l'ordre induit par $T \circ s$ est identique à celui induit par s pour toute fonction strictement croissante T, nous cherchons un critère invariant par transformation strictement croissante. Par exemple, un critère de la forme suivante :

$$C^{\phi}(s) = \|\phi(s) - \phi(f)\| \;\; avec \;\; \phi \;\; telle \;\; que \;\; \phi(T \circ s) = \phi(s),$$

pour toute fonction strictement croissante T.

Nous définissons ci-dessous les courbes EM et MV associées à une fonction de score, ainsi que les critères d'évaluation correspondants.

Définition. *(Courbes MV et EM)*
Soit s une fonction de score, on définit pour tout $t > 0$ et $\alpha \in (0, 1)$,

$$\begin{cases} MV_s(\alpha) = \inf_{u \geq 0} Leb(s \geq u) \quad s.c. \quad \mathbb{P}(s(X) \geq u) \geq \alpha \\ EM_s(t) = \sup_{u \geq 0} \mathbb{P}(s(X) \geq u) - tLeb(s \geq u) \end{cases}$$

Définition. *(Critères d'évaluation)*

$$\begin{cases} C^{MV}(s) = \|MV_s\|_{L^1} \\ C^{EM}(s) = \|EM_s\|_{L^1} \end{cases}$$

Les courbes MV et EM vérifient la propriété d'invariance

$$MV^* := MV_f = MV_{T \circ f} \;\; et \;\; EM^* := EM_f = EM_{T \circ f}$$

et pour tout α, s:

$$MV^*(\alpha) \leq MV_s(\alpha) \;\; et \;\; EM^*(\alpha) \geq EM_s(\alpha).$$

Ce qui implique:

$$\operatorname*{argmin}_s \|MV_s - MV^*\|_{L^1} = \operatorname*{argmin}_s \|MV_s\|_{L^1} \;\; et \;\; \operatorname*{argmin}_s \|EM_s - EM^*\|_{L^1} = \operatorname*{argmax}_s \|EM_s\|_{L^1}.$$

Cette dernière remarque permet d'éliminer les termes EM^* et MV^*, ainsi que la dépendance explicite à la densité dans les critères, ce qui en facilite l'estimation.

2.6 Explicabilité des modèles: Valeurs de Shapley

Les valeurs de Shapley, issues de la théorie des jeux coopératifs, permettent d'attribuer de façon équitable la contribution de chaque variable explicative à la prédiction d'un modèle. Dans ce cadre, chaque variable est interprétée comme un joueur d'un jeu coopératif, dont le gain collectif correspond à la prédiction du modèle (par exemple, le score d'anomalie). Le problème consiste alors à répartir ce gain de manière juste entre les variables, en fonction de leur contribution réelle.

L'idée principale est de mesurer la contribution moyenne d'une variable donnée, en considérant toutes les combinaisons possibles avec les autres variables. Ainsi, on peut quantifier l'impact individuel de chaque variable sur la prédiction, indépendamment des interactions avec les autres.

Mathématiquement, soit f une fonction de prédiction (le modèle) et N l'ensemble des variables explicatives. La valeur de Shapley associée à une variable $i \in N$ est définie par :

$$\phi_i(f) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left[f(S \cup \{i\}) - f(S) \right]$$

où la somme porte sur tous les sous-ensembles S de N ne contenant pas i, $f(S)$ désignant la prédiction du modèle lorsqu'on ne considère que les variables dans S, et $f(S \cup \{i\})$ celle obtenue en ajoutant la variable i. Le coefficient associé à chaque différence pondère la contribution marginale de i en fonction du nombre de permutations possibles.

Cependant, le calcul exact des valeurs de Shapley présente une complexité exponentielle en le nombre de variables, car il nécessite d'évaluer 2^n combinaisons pour un modèle à n variables. En pratique, des méthodes d'approximation sont souvent employées pour rendre ce calcul tractable.

3 Données

3.1 Datasets utilisés

Trois jeux de données ont été exploités pour mener cette étude. Ils sont tous de nature *cross-sectionnelle* (sans suivi temporel) et comportent des variables *mixtes*, c'est-à-dire à la fois numériques (montant, revenu, dette, etc.) et catégorielles (type de logement, statut marital, objet du prêt, etc.). Voici les caractéristiques détaillées de chacun :

- **Credit Card Fraud Detection Dataset (2023)**
  Source : Kaggle
  Taille : Environ 550 000 observations, 30 variables
  Description : Ce jeu de données contient des transactions par carte de crédit réalisées sur deux jours. Les variables `V1` à `V28` sont issues d'une transformation par analyse en composantes principales (ACP), visant à protéger la confidentialité des données originales. La variable `Amount` représente le montant de la transaction, et `Class` indique si elle est frauduleuse (1) ou non (0).

- **Credit Risk Analysis**
  Source : Kaggle
  Taille : Environ 32 500 observations, 12 variables
  Description : Ce dataset regroupe des données personnelles et financières d'emprunteurs, notamment leur revenu, le montant du prêt, la durée du crédit, ainsi que des informations telles que le statut familial, le type de logement ou l'objectif du crédit.

- **Statlog (German Credit Data)**
  Source : UCI Machine Learning Repository
  Taille : 1 000 observations, 20 variables
  Description : Ce jeu contient des informations sur des individus ayant demandé un crédit en Allemagne. Chaque observation est étiquetée comme un "bon" ou "mauvais" client selon sa capacité à rembourser. Les variables incluent des informations sur l'emploi, l'épargne, le type de crédit demandé et les antécédents de paiement.

3.2 Statistiques descriptives

Nous allons observer et analyser en détail les statistiques descriptives du dataset `credit_risk.csv`[7], qui possède 32,581 observations d'emprunteurs et 11 variables. Il s'agit donc de notre dataset de taille intermédiaire.

Variable	Type	Description
`Id`	Identifiante	Identifiant unique de chaque emprunteur.
`Age`	Quantitative discrète	Âge de l'emprunteur (années).
`Income`	Quantitative discrète	Revenu annuel de l'emprunteur.
`Home`	Catégorielle nominale	Status de propriété.
`Emp_length`	Quantitative discrète	Ancienneté professionnelle (années).
`Intent`	Catégorielle nominale	Motif du prêt.
`Amount`	Quantitative discrètes	Montant du prêt demandé.
`Rate`	Quantitative continue	Taux d'intérêt appliqué sur le prêt (%).
`Status`	Binaire	Statut du prêt (remboursé, défaut, en cours).
`Percent_income`	Quantitative continue	Montant du prêt en pourcentage du revenu.
`Default`	Catégorielle nominale	Indique si l'emprunteur a déjà fait défaut lors d'un précédent prêt.
`Cred_length`	Quantitative discrète	Durée d'historique de crédit (années).

Table 1: Description des variables de `credit_risk.csv`

	Age	Income	Emp_length	Amount	Rate	Percent_income	Cred_length
count	32581.00	32581.00	31686.00	32581.00	29465.00	32581.00	32581.00
mean	27.73	66074.85	4.79	9589.37	11.01	0.17	5.80
median	26.00	55000.00	4.00	8000.00	10.99	0.15	4.00
std	6.35	61983.12	4.14	6322.09	3.24	0.11	4.06
min	20.00	4000.00	0.00	500.00	5.42	0.00	2.00
25%	23.00	38500.00	2.00	5000.00	7.90	0.09	3.00
50%	26.00	55000.00	4.00	8000.00	10.99	0.15	4.00
75%	30.00	79200.00	7.00	12200.00	13.47	0.23	8.00
max	144.00	6000000.00	123.00	35000.00	23.22	0.83	30.00
Missing Values	0.00	0.00	895.00	0.00	3116.00	0.00	0.00

Table 2: Statistiques descriptives des variables quantitatives

Les valeurs maximales de l'âge et de l'ancienneté professionnelles sont aberrantes puisqu'elles sont respectivement de 144 et 123, donc au-delà de la durée de vie possible d'un humain. Aussi, le revenu maximal est fortement supérieur au troisième quartile, ce qui suggère l'existence d'un ou plusieurs outliers. Par ailleurs, la catégorie `OTHER` de la variable `HOME` représente seulement 0.33% des observations, créant un déséquilibre dans les données et causant potentiellement des problèmes d'identification de son effet par les modèles.

	Occurences	Fréquence (%)
RENT	16446	50.48
MORTGAGE	13444	41.26
OWN	2584	7.93
OTHER	107	0.33

(a) `Home`

	Occurences	Fréquence (%)
EDUCATION	6453	19.81
MEDICAL	6071	18.63
VENTURE	5719	17.55
PERSONAL	5521	16.95
DEBTCONSOLIDATION	5212	16.00
HOMEIMPROVEMENT	3605	11.06

(b) `Intent`

	Occurences	Fréquence (%)
N	26836	82.37
Y	5745	17.63

(c) `Default`

	Occurences	Fréquence (%)
0	25473	78.18
1	7108	21.82

(d) `Status`

Table 3: Statistiques descriptives des variables catégorielles

Le premier histogramme confirme notre doute sur l'existence d'outliers dans les revenus puisque la distribution de la variable `Income` est fortement étalée vers la droite : le coefficient d'asymétrie (skewness) atteint 32.9.

Concernant le taux d'emprunt, sa distribution montre deux pics vers 8% et 11%, ce qui peut provenir du motif du crédit ou indiquer l'existence de deux segments distincts d'emprunteurs.
Les histogrammes relatifs aux variables `Age` et `Cred_length` confirment la présence de valeurs aberrantes.
Les hauts revenus empruntent plus en valeur absolue mais moins en proportion de leur revenu (`Income` est positivement corrélé à `Amount` et négativement à `Percent_income`).

Nous observons des corrélations positives entre l'âge, l'ancienneté et le revenu. Cela est cohérent avec l'idée que, généralement, les individus plus âgés tendent à avoir plus d'expérience professionnelle et, par conséquent, des revenus plus élevés.

Les variables `Age` et `Cred_length` sont fortement corrélées (corrélation de 0.86) puisque l'âge et la durée de l'historique de crédit évoluent souvent ensemble dans le temps, créant une corrélation naturelle.

Les écarts par rapport aux relations attendues entre les variables peuvent révéler des anomalies. Par exemple, un individu dont le montant emprunté est anormalement élevé au regard de son niveau de revenu constitue un cas à surveiller.

[7]Pore

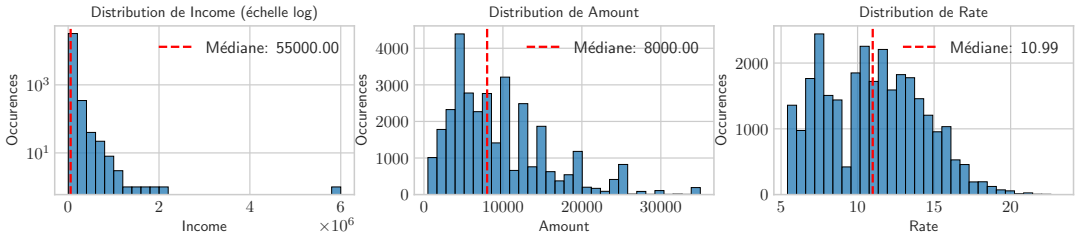

Figure 4: Distributions des variables `Income`, `Amount` et `Rate`

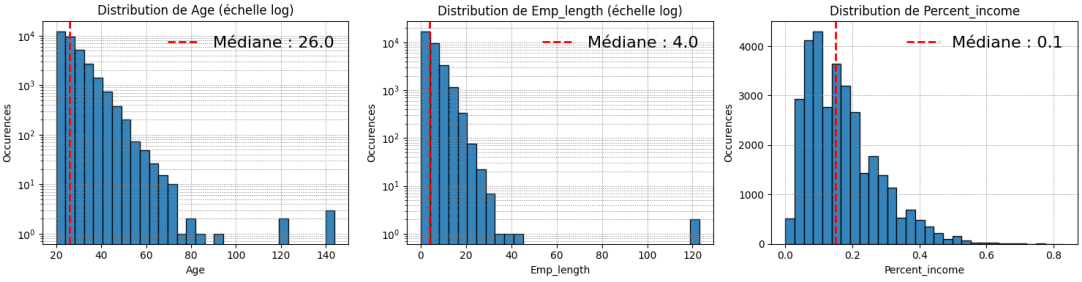

Figure 5: Distributions des variables `Age`, `Emp_length` et `Percent_income`

4 Approche Empirique

4.1 Pipeline de travail

Nous commençons par le prétraitement de nos datasets afin de les préparer aux différentes méthodes de détection d'ouliers. `credit_risk.csv` est le seul à présenter un nombre non négligeable de valeurs manquantes : environ 10% pour la variable `Rate` et 3% pour `Emp_length`. Nous les avons comblées par la médiane, qui est simple et plus robuste aux valeurs aberrantes que la moyenne. Il manque également la dernière observation de la variable `Amount` dans `credit_card.csv`. Nous choisissons de supprimer complètement cette dernière observation car, étant donné la taille du dataset (568,630 observations), cela n'aura pas de conséquences sur nos résultats.
Nous avons ensuite encodé les datasets `credit_risk.csv` et `credit_german.csv` (`credit_card.csv` ne contenant que des variables quantitatives) à l'aide du One-Hot encodeur. Ce choix est justifié par le fait que les variables ne comportent que peu de catégories (5 en moyenne), donc la dimension des datasets n'augmente pas trop, et qu'il n'y a pas d'ordres entre les différentes catégories.
Enfin, toutes les méthodes que nous allons comparer nécessitent que les variables soient mises à l'échelle, à l'exception d'Isolation Forest. Pour cela, nous choisissons le Min-Max Scaler.
Après cette première phase de prétraitement, nous allons entraîner les différentes méthodes (KNN, LOF, Isolation Forest, OC-SVM, Autoencodeur) à l'aide de la librairie `PyOD` et notre Autoencodeur développé manuellement sur nos trois datasets. Nous évaluerons leurs performances à l'aide de différentes méthodes non supervisées. Nous proposons ensuite une tentative de supervision du problème, afin d'utiliser des métriques d'évaluation plus classiques (AUC, F1 score, etc.). Enfin, nous explorons une méthode d'évaluation développée par Goix [2016]. Pour compléter notre étude, nous chercherons à comprendre, à l'aide des valeurs de Shapley et de la librairie `SHAP`, comment les modèles identifient les outliers. Tout cela nous permettra, *in fine*, de dresser un benchmark des différents modèles sur des datasets de tailles variées, en tenant compte de leur capacité à identifier des outliers ainsi que de leur temps d'exécution.

4.2 Évaluation empirique des modèles

Pour l'évaluation empirique des modèles de détection d'anomalies, nous avons retenu trois approches complémentaires. La première repose sur une comparaison visuelle des distributions de scores et des chevauchements entre modèles, afin d'analyser leur convergence. La deuxième consiste à superviser artificiellement le problème via l'injection de bruit pour générer des labels, permettant l'évaluation avec des métriques classiques ; la troisième repose sur des critères non supervisés (EM et MV) adaptés aux contextes sans étiquettes.

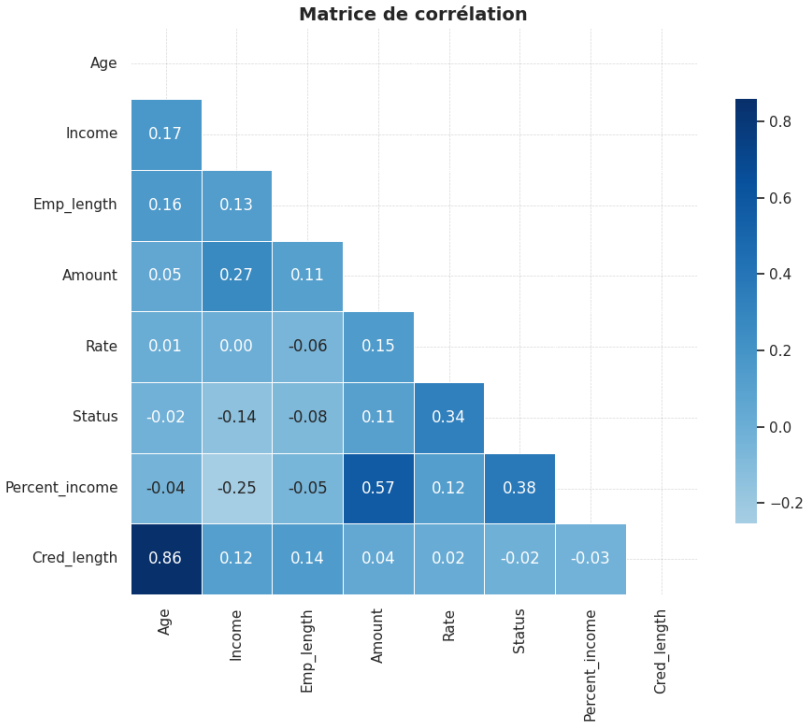

Figure 6: Corrélation des variables

4.2.1 Analyse comparative des modèles

L'objectif de cette section est de comparer différents modèles de détection d'anomalies à partir des **scores d'anomalie** qu'ils génèrent. Ces scores permettent d'estimer le *degré d'atypicité* de chaque observation, en la plaçant sur une **échelle continue**, plutôt que de la classifier de manière binaire (0 : normal, 1 : anomalie). La nature et l'échelle de ces scores dépendent de chaque algorithme.

Afin de permettre une comparaison équitable entre les modèles, les scores d'anomalie ont été **normalisés sur une échelle commune de 0 à 100**. Cela garantit que, malgré des logiques internes différentes, les modèles soient comparables en termes d'interprétation des scores.

Dans cette étude, nous utilisons la **base de données CreditRisk** et appliquons six modèles de détection d'anomalies : KNN, LOF (Local Outlier Factor), Isolation Forest, One-Class SVM, Autoencoder(Pyod) et Autoencoder(Challenger). Pour analyser les performances des modèles, nous allons procéder à plusieurs méthodes d'analyse de distribution des scores, incluant l'utilisation d'histogrammes, de boxplots et du Population Stability Index (PSI)et nous examinerons la corrélation entre les modèles.

a. Histogrammes des scores

Afin de comparer les performances des six modèles , nous avons analysé les distributions de leurs scores d'anomalie normalisés (échelle de 0 à 100) à l'aide des histogrammes des scores d'anomalie. Seules les observations ayant un score supérieur à 10 ont été représentées. L'objectif de cette visualisation est de juger la capacité de chaque modèle à discriminer les observations normales des observations atypiques. Trois critères visuels guident l'analyse :

- **Bimodalité :** suggère une nette séparation entre les deux groupes (normaux vs anomalies).
- **Queue lourde à droite (heavy tail) :** reflète une détection ciblée d'un petit nombre d'observations très atypiques.
- **Absence de scores moyens (compact/dispersé) :** plus la répartition est polarisée (faibles vs très élevés), plus le modèle est précis.

b. Boxplots des scores

Des **boxplots** ont été construits pour visualiser la dispersion des scores et identifier les *valeurs extrêmes*. Un modèle efficace devrait produire une majorité d'observations normales regroupées autour de la médiane avec un petit groupe d'anomalies représentées par des points éloignés, au-delà des moustaches du boxplot.

c. Corrélation des modèles

Pour analyser la corrélation entre les anomalies détectées par les différents modèles, nous avons utilisé plusieurs outils visuels. Un histogramme a d'abord été créé pour visualiser combien d'anomalies étaient détectées par un, deux, trois, ou même cinq modèles. Ensuite, une matrice de chevauchement a été construite pour évaluer dans quelle mesure les modèles partagent les mêmes anomalies, avec des zones de chevauchement plus importantes indiquant des modèles similaires dans leur approche. Enfin, un diagramme de Venn a permis d'illustrer de manière claire les intersections entre les anomalies détectées par les différents modèles.

4.2.2 Tentative de supervision du problème

La difficulté à évaluer la performance des méthodes non supervisées réside principalement dans le fait qu'il nous est impossible de distinguer ce qui est un vrai outlier de ce qui n'en est pas. Dès lors, nous avons tenté de superviser notre problème afin de pouvoir utiliser les métriques d'évaluation les plus classiques[8]. L'idée est de générer des anomalies dans nos datasets, afin d'avoir accès aux labels, puis d'appliquer les algorithmes en non supervisé. Enfin, nous comparons les résultats obtenus avec les résultats attendus (labels).
Malheureusement, la librairie `PyOD` ne propose pas de générateur d'outlier et nous n'avons pas trouvé d'alternative officielle et reconnue. Nous avons alors tenté notre propre approche pour générer des outliers. Bien que cette approche simplifie le problème, elle a de nombreuses limites que nous évoquerons par la suite.
Nous "générons" des outliers en bruitant certaines observations et en considérant que ces observations bruitées sont des outliers. Pour cela, nous choisissons la proportion du dataset (`frac`) et le nombre de colonnes (`cols_to_perturb`) que nous souhaitons bruiter, ainsi que la taille du bruit (`size_noise`). Notre fonction `create_noise`[9] sélectionne une fraction `frac` du dataset, puis itère dessus. À chaque itération, il sélectionne `cols_to_perturb` colonnes au hasard et bruite la valeur de ces colonnes. S'il s'agit d'une variable quantitative, il ajoute un bruit gaussien centré et de variance `size_noise`. S'il s'agit d'une variable catégorielle, il sélectionne au hasard une catégorie parmi celles que contient la variable. La fonction renvoie le dataset bruité, ainsi qu'un vecteur binaire indiquant les observations bruitées.
Il nous faut maintenant déterminer les hyperparamètres `size_noise` et `cols_to_perturb`. Pour cela, nous utilisons le Population Stability Index (PSI) et nous choisissons KNN comme modèle de référence. Pour chacun des trois datasets, nous appliquons le modèle en non supervisé, puis nous calculons le PSI moyen entre le dataset et les outliers. Nous obtenons 0.3 pour les trois datasets. Finalement, nous testons différentes combinaisons de `size_noise` et `cols_to_perturb`, en prenant soin d'utiliser les mêmes autres paramètres (contamination, nombre de voisins, etc.). Pour chaque combinaison, nous calculons le PSI moyen entre le dataset et les données bruitées (i.e. les outliers) et nous sélectionnons les hyperparamètres qui mènent à un PSI de 0.3 :

- `credit_risk.csv` : `size_noise` = 4.5, `cols_to_perturb` = 6;
- `credit_german.csv` : `size_noise` = 1, `cols_to_perturb` = 11.

Nous ne faisons pas cette étude sur le dataset `credit_card.csv` car il est trop couteux d'un point de vue computationnel, que ce soit pour le bruiter, calculer les PSI de ses variables, ou entraîner des modèles tels qu'OC-SVM et Autoencodeurs[10].
Les limites de cette tentative de supervision sont non négligeables. En effet, nous faisons l'hypothèse qu'une donnée bruitée est forcément un outlier et qu'une donnée non bruitée n'en est pas. Ainsi, même si un algorithme trouve toutes les données bruitées, nous ne sommes pas certains qu'il a trouvé tous les outliers et que toutes les observations qu'il classe comme outliers en sont réellement. En créant ce bruit synthétique, il

[8] Voir 2.5.1
[9] Voir Code 2
[10] Voir 5.4

y a un risque que les modèles apprennent à identifier ce bruit plutôt qu'à détecter de vraies anomalies. Cette approche est donc fortement biaisée. Enfin, le choix des hyperparamètres affecte grandement les résultats de l'évaluation. Toutefois, nous allons utiliser cette tentative de supervision pour évaluer et comparer les différents modèles par curiosité, mais nous n'en tirerons aucune conclusion définitive. Il sera cependant intéressant de comparer ses résultats à l'approche proposée par Nicolas Goix[11].

4.2.3 Évaluation des algorithmes de détection d'anomalies (Goix [2016])

Nous évaluons les différents modèles de détections d'anomalies en calculant les critères C^{EM} et C^{MV} basés sur les courbes EM et MV. Dans ce cadre, nous adoptons la convention selon laquelle plus le score est faible, plus l'observation est considérée comme anormale. Afin de renverser l'ordre des scores, nous appliquons la transformation $x \mapsto 100 - x$.

Pour calculer les critères, il est nécessaire d'estimer les courbes EM et MV, la distribution des données étant inconnue. Nous estimons donc $\mathbb{P}(s(X) \geq u)$ par son équivalent empirique $\frac{1}{n}\sum_{i=1}^{n} 1_{s(X_i)\geq u}$. Par ailleurs, nous approchons $Leb(s \geq u)$ à l'aide d'une méthode de Monte Carlo. Nous obtenons ainsi les courbes estimées $\widehat{EM}$ et $\widehat{MV}$, à partir desquelles nous estimons les critères d'évaluation :

$$\begin{cases} \widehat{C}^{MV}(s) = \|\widehat{MV}_s\|_{L^1} \\ \widehat{C}^{EM}(s) = \|\widehat{EM}_s\|_{L^1}. \end{cases}$$

L'approximation par Monte Carlo n'étant réalisable qu'en petite dimension, nous adoptons la méthodologie développée par Nicolas Goix afin de permettre le calcul des critères en grande dimension. Nous appliquons cette approche avec $d' = 5$ et $m = 10$ selon les étapes suivantes :

- Sous-échantillonner les données selon les variables, en sélectionnant aléatoirement d' variables pour le calcul du score, des courbes $\widehat{EM}$ et $\widehat{MV}$, ainsi que des critères de performance associés.
- Répéter ce sous-échantillonnage m fois, puis prendre la moyenne des critères de performances obtenus pour obtenir les critères de performance finaux.

5 Résultats

5.1 Analyse comparative des modèles

a. Histogrammes des scores

- **KNN** se démarque par une concentration marquée des scores dans les valeurs basses, avec très peu d'observations ayant un score compris entre 40 et 60. L'absence de scores moyens montre que les points sont soit clairement **normaux**, soit clairement **anormaux**.
- **LOF** produit une distribution plus irrégulière, avec un certain nombre de scores moyens, suggérant une discrimination plus diffuse. Le modèle semble hésiter à trancher entre **normalité** et **anomalie** pour certaines observations.
- **Isolation Forest** montre une distribution centrée et relativement uniforme, traduisant une faible sélectivité. Le modèle accorde des scores similaires à la majorité des points, ce qui limite sa capacité à hiérarchiser la **normalité**.
- **One-Class SVM** présente une **queue lourde à droite** et une répartition un peu plus étalée, traduisant une capacité correcte à détecter des **anomalies**, bien qu'un peu moins tranchée que **KNN**.
- **Autoencoder** concentre l'essentiel des observations dans des scores très faibles, avec une **queue extrêmement marquée à droite**. Ce modèle isole un petit nombre de points très atypiques avec des scores anormalement élevés.

[11] Goix [2016]

- **Autoencoder (Challenger)** présente un maximum d'observations avec un score égal à **30**, ce qui contraste avec les autres modèles dont les scores sont majoritairement concentrés autour de **10**. La distribution n'est pas continue, ce qui suggère une séparation moins progressive entre normales et anomalies.

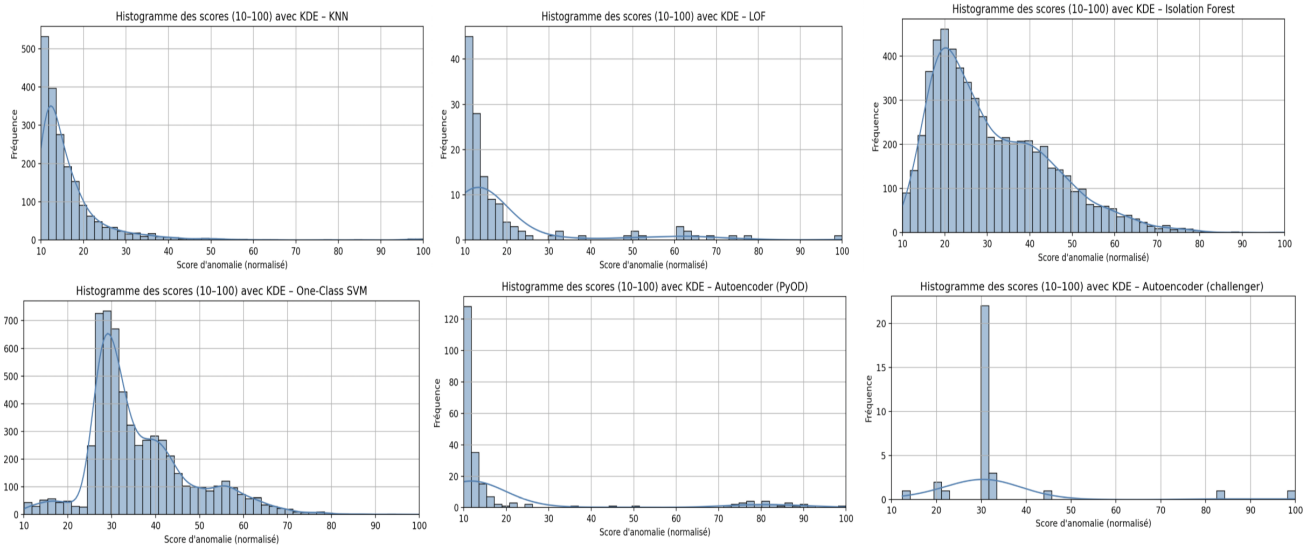

Figure 7: Histogrammes des scores d'anomalie pour chaque modèle

b. Boxplots

Voici les résultats obtenus à partir des boxplots des scores d'anomalie:

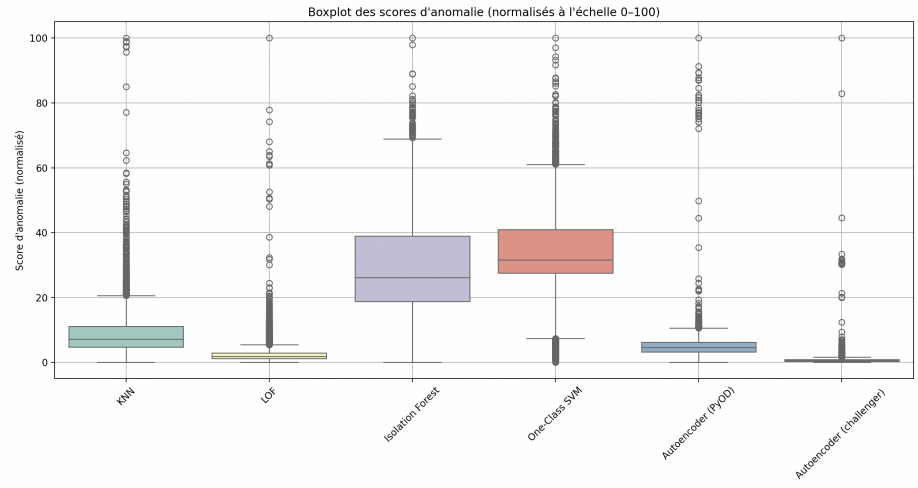

Figure 8: Boxplots des scores d'anomalie pour chaque algorithme

- **KNN** concentre les scores dans les faibles valeurs avec peu de scores intermédiaires, indiquant une séparation nette entre normales et anomalies. → *Très bonne capacité de détection, peu d'ambiguïté.*
- **LOF** présente une distribution resserrée avec une longue queue à droite, isolant efficacement les anomalies rares. → *Excellente capacité à détecter des comportements atypiques.*
- **Isolation Forest** montre une distribution étalée et centrée, traduisant une difficulté à hiérarchiser les observations. → *Capacité de détection limitée, peu sélective.*

- **One-Class SVM** affiche une répartition plus diffuse avec une queue droite marquée. → *Détection correcte mais manque de netteté dans la séparation.*
- **Autoencoder(Pyod)** regroupe les normales sur de très faibles scores avec quelques points très élevés. → *Bonne capacité à isoler des anomalies nettes mais peu nombreuses.*
- **Autoencoder (challenger)** concentre la quasi-totalité des scores sur des valeurs très faibles, avec une queue droite extrêmement courte. → *Très bonne capacité à identifier la majorité des données comme normales, mais potentiellement moins sensible aux anomalies subtiles.*

c. Corrélation entre modèles

L'analyse des résultats commence par l'examen de l'histogramme, qui représente la fréquence des anomalies détectées par 0, un, deux, jusqu'à six modèles.

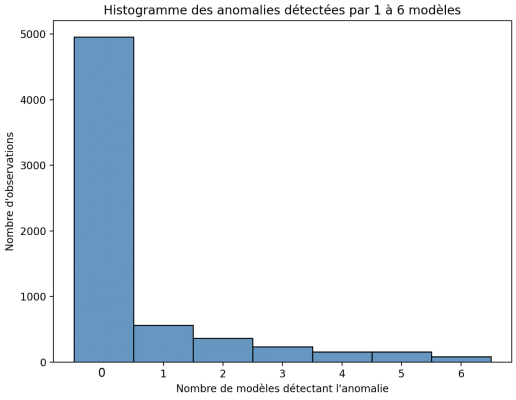

Figure 9: Histogramme des anomalies détectées par différents modèles

On remarque les points qui sont détectés par 0 modèles : potentiellement des points normaux. La majorité des anomalies sont détectées par un seul modèle, ce qui traduit une forte hétérogénéité dans les détections. Ce résultat suggère que les modèles ont des sensibilités différentes et identifient des types d'anomalies spécifiques que d'autres ne perçoivent pas. Les anomalies partagées par plusieurs modèles sont moins fréquentes, ce qui peut indiquer une complémentarité dans les approches ou une surdétection pour certains modèles.

Après avoir analysé les anomalies détectées individuellement par les modèles, nous avons cherché à mieux comprendre la cohérence et les divergences entre ces différentes méthodes. Pour cela, nous avons d'abord observé la **matrice de chevauchement**, qui comptabilise le nombre d'anomalies communes à chaque paire de modèles.

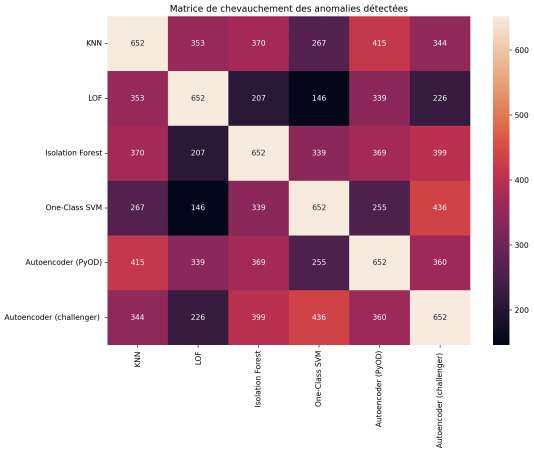

Figure 10: Matrice de chevauchement des anomalies détectées

Comme chaque algorithme a été contraint à détecter 652 anomalies, la diagonale de cette matrice est constante. En revanche, les valeurs hors diagonale révèlent des comportements très différents : on observe par exemple que **KNN** et **Autoencoder (PyOD)** ont **415 anomalies en commun**, traduisant une forte similarité dans leur détection. De même, l'**Autoencoder Challenger** et le **One-Class SVM** partagent **436 anomalies**. Enfin, **Isolation Forest** se distingue par un nombre d'anomalies communes élevé à la fois avec **Autoencoder (PyOD)**, **KNN**, **Autoencoder Challenger** et **SVM**.

Pour compléter cette analyse, nous avons représenté graphiquement les chevauchements entre certains modèles à l'aide de diagrammes de Venn. Le premier, basé sur **KNN**, **AE(PyOD)** et **Isolation Forest**, montre une intersection centrale de **268** anomalies détectées par les trois modèles, formant un socle d'accord fort entre ces approches. Toutefois, chacun de ces algorithmes conserve une part non négligeable d'anomalies qu'il détecte seul : l'**Autoencoder** en identifie **136**, **KNN 135** et **Isolation Forest 181**.

Un second diagramme de Venn, cette fois entre l'**AE Challenger**, le **One-Class SVM** et **Isolation Forest**, met en évidence des dynamiques similaires. On retrouve **269** anomalies détectées par les trois modèles, mais aussi des différences notables : l'**AE** détecte **86** anomalies de manière exclusive, le **One-Class SVM 146** et **Isolation Forest 183**. Ces chiffres confirment que si certains cas sont unanimement reconnus comme atypiques, chaque modèle a également sa propre vision de ce qui constitue une anomalie.

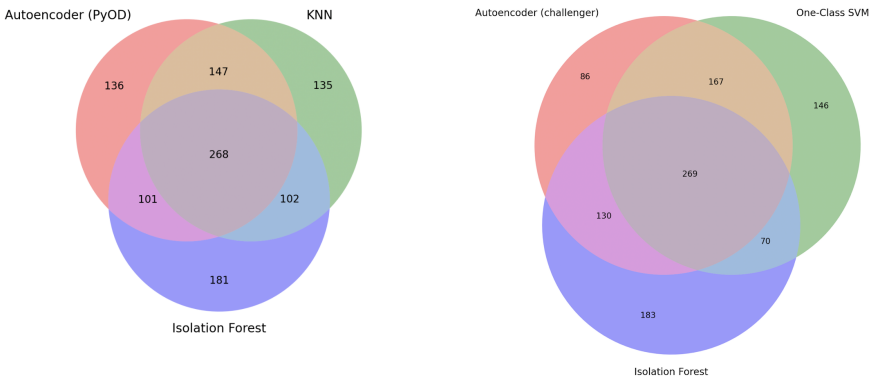

Figure 11: Diagramme de Venn des anomalies communes entre trois modèles

Cette approche d'analyse comparative des algorithmes a permis d'identifier des convergences, des divergences, ainsi que des comportements spécifiques propres à chaque modèle. Elle offre une évaluation visuelle à l'aide de graphiques mais elle n'est pas fondée sur des métriques scientifiques ou des étiquettes de référence, ce qui limite la robustesse de l'évaluation. Dons,il est pertinent de comparer ces résultats avec ceux obtenus à partir de notre générateur d'outliers.

5.2 Supervision du problème

Comme indiqué dans 4.2.2, nous n'avons testé cette tentative de supervision que sur les datasets `credit_risk.csv` et `credit_german.csv` (moyenne et petite taille) car le troisième est trop volumineux. Les résultats obtenus sont condensés dans le tableau suivant et nous permettent d'estimer les performances des différents modèles via l'AUC, la précision, le recall, le F1-score et l'accuracy[12]. Les meilleurs résultats pour chaque métrique sont affichés en gras pour plus de lisibilité.

Sur le dataset de taille moyenne, KNN domine les autres modèles pour toutes les métriques sauf l'accuracy, où LOF le dépasse légèrement. Avec un AUC de 0.960 et un F1-score de 0.852, KNN performe à la fois en détection globale et en classification effective. Au contraire, LOF atteint une accuracy de 0.948 et un AUC de 0.904 mais ses autres métriques restent autour de 0.7. Cela le rend globalement similaire à l'AE de `PyOD`. Notons que notre AE est loin de performer comme ce dernier et tous les autres modèles.

Sur le dataset de petite taille, les performances globales sont plus faibles, et KNN ne se démarque que sur la précision, indiquant seulement un taux de 51.5% de vrais positifs parmi les outliers détectés. Isolation Forest se démarque sur le recall, indiquant que 61.5% des vrais outliers ont été correctement identifiés comme tels. Cette fois, LOF obtient les meilleurs scores sur les trois dernières métriques. Cependant, les valeurs restent

[12]Voir 2.5.1

Dataset	Modèle	Métriques				
		AUC	Préc.	Recall	F1	Acc.
`credit_risk` 32,581 observations 10 variables	KNN	**0.960**	**0.819**	**0.888**	**0.852**	0.897
	IsolationForest	0.920	0.640	0.636	0.638	0.926
	LOF	0.904	0.747	0.725	0.736	**0.948**
	OC-SVM	0.758	0.503	0.524	0.513	0.899
	Autoencodeur	0.933	0.668	0.700	0.683	0.935
	AE Challenger	0.791	0.370	0.369	0.370	0.876
`credit_german` 1,000 observations 20 variables	KNN	0.852	**0.515**	0.500	0.507	0.890
	IsolationForest	0.862	0.381	**0.615**	0.471	0.880
	LOF	**0.871**	0.500	0.586	**0.540**	**0.903**
	OC-SVM	0.753	0.386	0.548	0.453	0.863
	Autoencodeur	0.856	0.382	0.520	0.441	0.890
	AE Challenger	0.672	0.250	0.219	0.233	0.847

Table 4: Comparaison des performances des modèles évaluées par supervision

basses et l'accuracy est trompeuse dans ce contexte déséquilibré. En effet, nous avons bruité 10% de nos observations donc un modèle classifiant toutes les données comme "normales" aura une accuracy de 0.9. Cela neutralise l'accuracy de 0.903 de LOF.

Dans l'ensemble, cette tentative de supervision nous permet de nous rattacher à des métriques plus simples et interprétables afin d'estimer les performances des différents modèles. Cependant, il faut garder à l'esprit les limites de cette approche[13] et considérer cette étude comme exploratoire plutôt qu'une évaluation rigoureuse. La section suivante s'appuiera sur l'approche de Goix [2016] pour dresser un réel benchmark et nous permettra d'observer si les résultats précédents sont cohérents.

5.3 Benchmark dans le cadre non supervisé à l'aide des métriques de Goix [2016]

Comme indiqué dans l'approche empirique, nous avons estimé les critères basés sur les courbes EM et MV en suivant la méthodologie développée par Nicolas Goix, reposant sur un sous-échantillonage. Nous présentons ci-dessous les résultats obtenus sur les datasets `credit_risk` et `credit_german`, le dataset `credit_card` étant trop volumineux. Nous indiquons également les temps d'estimation de ces critères. Les métriques associées aux courbes EM et MV concordent sur le choix du meilleur algorithme sur le dataset `credit_risk`, il s'agit du modèle KNN (puisqu'il maximise le critère $\widehat{C}^{EM}$ et minimise le critère $\widehat{C}^{MV}$), suivi des modèles OC-SVM et LOF. En revanche pour le dataset `credit_german`, les deux métriques divergent : KNN maximise le critère $\widehat{C}^{EM}$ tandis que OC-SVM minimise le critère $\widehat{C}^{MV}$.

[13] Voir 4.2.2

Dataset	Modèle	Métriques		Durée (s)
		$\hat{C}^{EM}$	$\hat{C}^{MV}$	
`credit_risk` 32,581 observations 10 variables	KNN	**0.078**	**19.76**	169
	IsolationForest	0.000086	199.33	64
	LOF	0.00034	195.24	**61**
	OC-SVM	0.00029	29.20	329
	Autoencodeur	0.000029	123.85	7301
	AE Challenger	0.000053	202.23	4377
`credit_german` 1,000 observations 20 variables	KNN	**0.00054**	33.83	119
	IsolationForest	0.000047	38.92	12
	LOF	0.000066	57.44	**11**
	OC-SVM	0.00025	**16.15**	12
	Autoencodeur	0.000065	18.42	591
	AE Challenger	0.000022	38.18	295

Table 5: Comparaison des performances des modèles et des temps d'estimation

Les deux autoencodeurs apparaissent globalement moins performants, en plus de présenter des temps d'estimation significativement plus élevés.

5.4 Benchmark temps et scalabilité

Nous avons comparé les temps d'exécution des différents algorithmes selon le dataset. Les résultats sont résumés dans le tableau suivant :

Dataset	Modèle	Durée (s)
`credit_risk` 32,581 observations 10 variables	KNN	0.432
	IsolationForest	0.617
	LOF	**0.413**
	OC-SVM	9.900
	Autoencodeur	163.128
	AE Challenger	183.753
`credit_card` 568,630 observations 29 variables	KNN	83.493
	IsolationForest	**4.506**
	LOF	80.226
	OC-SVM	6166.390
	Autoencodeur	3048.129
	AE Challenger	2198.708
`credit_german` 1,000 observations 20 variables	KNN	**0.015**
	IsolationForest	0.131
	LOF	0.016
	OC-SVM	0.020
	Autoencodeur	5.786
	AE Challenger	8.452

Table 6: Comparaison des temps d'exécution des modèles en fonction du dataset

On observe des différences radicales entre les modèles. L'OC-SVM se distingue par sa difficulté à s'adapter à des datasets volumineux. En effet, sur le plus petit (`credit_german`) il a une vitesse similaire à KNN et LOF et est même plus rapide qu'Isolation Forest, mais sur le dataset de taille moyenne (`credit_risk`) il est environ 20 fois plus lent que les modèles cités précédemment. Enfin, son temps d'exécution augmente drastiquement sur le dataset volumineux (`credit_card`), atteignant plus de 1h40 alors qu'Isolation Forest prend moins de 5 secondes. Ce dernier est de loin le plus rapide et scalable de tous. KNN et LOF, quant à eux, ont des durées d'exécution très similaires et correctes puisqu'ils s'entraînent en

un peu plus d'une minute sur le dataset volumineux. Comme attendu, les autoencodeurs prennent plus de temps à s'exécuter que les autres algorithmes, sauf OC-SVM sur le dataset volumineux. Notre AE a des résultats relativement similaires à l'AE de `PyOD` et est même 30% plus rapide que lui sur `credit_card`. Cela s'explique probablement par la fonctionnalité d'*Early Stopping* dont nous l'avons doté.

5.5 Explicabilité des modèles

- **Status** : Variable la plus influente. Un statut élevé est associé à une augmentation du score d'anomalie, ce qui suggère un comportement atypique ou à risque.
- **Emp_length** : L'ancienneté dans l'emploi influence fortement les prédictions. Des durées faibles (en bleu) augmentent les scores d'anomalie.
- **Home** : Le statut du logement (propriétaire, locataire, etc.) est aussi un facteur différenciant. Les profils `OWN` ou `RENT` ont des effets opposés.
- **Amount** : Des montants élevés de prêt contribuent fortement au caractère inhabituel du profil.
- **Intent** : Certains motifs de prêt (par exemple `VENTURE` ou `DEBTCONSOLIDATION`) semblent plus souvent associés à des anomalies.
- **Cred_length**, **Rate**, **Percent_income**, **Age**, **Income** : Ces variables ont un impact plus faible mais contribuent à affiner la prédiction.

L'analyse SHAP nous permet ainsi de mieux comprendre la logique du modèle en révélant les caractéristiques clés associées aux comportements atypiques. Cela est particulièrement utile pour les modèles non supervisés, où les décisions ne sont pas toujours intuitives.

6 Conclusion

6.1 Résumé des contributions

Ce mémoire a exploré la problématique complexe de la détection d'anomalies dans un cadre non supervisé, en mobilisant plusieurs approches complémentaires pour évaluer et comparer les performances de différents algorithmes. Trois axes d'analyse ont été déployés : une étude qualitative des distributions de scores, une évaluation semi-supervisée par injection de bruit artificiel, et enfin un cadre rigoureux d'évaluation non supervisée fondé sur les travaux de Goix (2016) via les courbes Excess-Mass (EM) et Mass-Volume (MV). Les résultats issus de ces approches convergent partiellement. KNN s'impose comme le modèle le plus robuste, performant à la fois sur les scores, les métriques supervisées et les critères non supervisés, notamment sur le jeu credit risk. OC-SVM et LOF offrent également de bons compromis. En revanche, les autoencodeurs, bien que théoriquement puissants, se montrent instables et coûteux en ressources, ce qui limite leur applicabilité. En complément, une analyse d'explicabilité par SHAP a permis de mieux comprendre les décisions des modèles. Elle révèle que des variables comme le Status (niveau élevé associé à un risque accru), Emp length (ancienneté faible liée à une anomalie probable), et Home (effets contrastés selon le type de logement) jouent un rôle déterminant dans les prédictions.

6.2 Limites

Ces résultats doivent toutefois être nuancés au regard de plusieurs limites méthodologiques et structurelles. D'une part, la nature artificielle des anomalies injectées peut biaiser l'évaluation supervisée : les modèles apprennent à détecter du bruit statistique plus que des ruptures structurelles. D'autre part, l'absence de labels empêche une validation fine des détections. À cela s'ajoutent des contraintes techniques, comme la sensibilité aux hyperparamètres et le manque de scalabilité de certaines méthodes, en particulier les autoencodeurs.

De plus, les données utilisées sont issues de jeux *open source*, dont certains, comme le jeu `credit_risk`, sont fortement anonymisés, limitant la richesse informative des variables. Cela peut nuire à la capacité des

modèles à détecter des anomalies réalistes. Enfin, l'évaluation du taux de faux positifs reste approximative dans un cadre non supervisé, et ne peut s'appuyer que sur des recoupements indirects ou des critères théoriques.

6.3 Perspectives

Ces constats ouvrent plusieurs pistes d'amélioration. L'approche par injection pourrait être affinée en générant des anomalies plus réalistes, issues de violations de règles métiers ou de comportements atypiques observés dans des contextes concrets. En parallèle, les modèles testés pourraient être appliqués à des cas d'usage réels comme la fraude bancaire ou les abus de marché, où une supervision partielle — par retour expert ou étiquetage a posteriori — permettrait de guider l'apprentissage tout en conservant une logique non supervisée. De plus, l'intégration systématique d'outils d'explicabilité, comme SHAP, pourrait enrichir les analyses en rendant les modèles plus transparents et interprétables, facilitant ainsi leur adoption opérationnelle dans des contextes réglementés. Un effort d'optimisation sur les modèles lourds, combiné à des techniques de sélection de variables non supervisées, permettrait enfin d'améliorer la robustesse et la lisibilité des résultats en grande dimension..

7 Annexes

Statistiques descriptives sur le dataset relatif à la qualité des créditeurs allemands

Le dataset `statlog_german_credit_data.csv`[14] possède 1,000 observations, 20 variables explicatives et une variable d'intérêt.

Variable	Type	Description
`Attribute1`	Catégorielle	Statut du compte courant existant.
`Attribute2`	Quantitative discrète	Durée du crédit (mois).
`Attribute3`	Catégorielle	Historique de crédit.
`Attribute4`	Catégorielle	Objet du crédit.
`Attribute5`	Quantitative discrète	Montant du crédit.
`Attribute6`	Catégorielle	Compte d'épargne / obligations.
`Attribute7`	Catégorielle	Ancienneté dans l'emploi actuel.
`Attribute8`	Quantitative discrète	Taux de mensualité en pourcentage du revenu disponible.
`Attribute9`	Catégorielle	Statut personnel et sexe.
`Attribute10`	Catégorielle	Autres débiteurs / garants.
`Attribute11`	Quantitative discrète	Durée de résidence actuelle.
`Attribute12`	Catégorielle	Type de propriété.
`Attribute13`	Quantitative discrète	Âge.
`Attribute14`	Catégorielle	Autres plans de financement.
`Attribute15`	Catégorielle	Type de logement.
`Attribute16`	Quantitative discrète	Nombre de crédits existants dans cette banque.
`Attribute17`	Catégorielle	Type d'emploi.
`Attribute18`	Quantitative discrète	Nombre de personnes à charge financièrement.
`Attribute19`	Binaire	Possession d'un téléphone.
`Attribute20`	Binaire	Travailleur étranger.
`class`	Binaire	Qualité du créditeur (1 = Bon, 2 = Mauvais) ; variable d'intérêt.

Table 7: Description des variables de `statlog_german_credit_data.csv`

	Attribute2	Attribute5	Attribute8	Attribute11	Attribute13	Attribute16	Attribute18
count	1000.00	1000.00	1000.00	1000.00	1000.00	1000.00	1000.00
mean	20.90	3271.26	2.97	2.85	35.55	1.41	1.16
median	18.00	2319.50	3.00	3.00	33.00	1.00	1.00
std	12.06	2822.74	1.12	1.10	11.38	0.58	0.36
min	4.00	250.00	1.00	1.00	19.00	1.00	1.00
25%	12.00	1365.50	2.00	2.00	27.00	1.00	1.00
50%	18.00	2319.50	3.00	3.00	33.00	1.00	1.00
75%	24.00	3972.25	4.00	4.00	42.00	2.00	1.00
max	72.00	18424.00	4.00	4.00	75.00	4.00	2.00
Missing Values	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 8: Statistiques descriptives des variables quantitatives

Statistiques descriptives sur le dataset relatif aux transactions par carte de crédit

Le dataset `creditcard_2023.csv`[15] possède 568,630 observations et 31 variables.

[14] Hofmann [1994]
[15] Elgiriyewithana [2023]

	Occurences	Fréquence (%)
A14	394	39.40
A11	274	27.40
A12	269	26.90
A13	63	6.30

(a) `Attibute1`

	Occurences	Fréquence (%)
A32	530	53.00
A34	293	29.30
A33	88	8.80
A31	49	4.90
A30	40	4.00

(b) `Attibute3`

	Occurences	Fréquence (%)
A43	280	28.00
A40	234	23.40
A42	181	18.10
A41	103	10.30
A49	97	9.70
A46	50	5.00
A45	22	2.20
A44	12	1.20
A410	12	1.20
A48	9	0.90

(c) `Attibute4`

	Occurences	Fréquence (%)
A61	603	60.30
A65	183	18.30
A62	103	10.30
A63	63	6.30
A64	48	4.80

(d) `Attibute6`

	Occurences	Fréquence (%)
A73	339	33.90
A75	253	25.30
A74	174	17.40
A72	172	17.20
A71	62	6.20

(e) `Attibute7`

	Occurences	Fréquence (%)
A93	548	54.80
A92	310	31.00
A94	92	9.20
A91	50	5.00

(f) `Attibute9`

	Occurences	Fréquence (%)
A101	907	90.70
A103	52	5.20
A102	41	4.10

(g) `Attibute10`

	Occurences	Fréquence (%)
A123	332	33.20
A121	282	28.20
A122	232	23.20
A124	154	15.40

(h) `Attibute12`

	Occurences	Fréquence (%)
A143	814	81.40
A141	139	13.90
A142	47	4.70

(i) `Attibute14`

	Occurences	Fréquence (%)
A152	713	71.30
A151	179	17.90
A153	108	10.80

(j) `Attibute15`

	Occurences	Fréquence (%)
A173	630	63.00
A172	200	20.00
A174	148	14.80
A171	22	2.20

(k) `Attibute17`

	Occurences	Fréquence (%)
A191	596	59.60
A192	404	40.40

(l) `Attibute19`

	Occurences	Fréquence (%)
A201	963	96.30
A202	37	3.70

(m) `Attibute20`

	Occurences	Fréquence (%)
1	700	70.00
2	300	30.00

(n) `class`

Table 9: Statistiques descriptives des variables catégorielles

Variable	**Type**	**Description**
`id`	Identifiante	Identifiant unique pour chaque transaction
`V1-V28`	Quantitatives continues	Caractéristiques anonymisées représentant divers attributs de transaction (par exemple, heure, lieu, etc.)
`Amount`	Quantitative continue	Le montant de la transaction
`Class`	Binaire	Etiquette indiquant si la transaction est frauduleuse (1) ou non (0)

Table 10: Description des variables de `creditcard_2023.csv`

Les variables `V1`,...,`V28` sont centrées et réduites, la variable `Amount` a une moyenne de 12041.96 et un écart-type de 6919.64. La variable binaire `Class` admet autant de 0 que de 1. Le dataset ne contient aucune valeur manquante.

Variable	**Q1 (25%)**	**Médiane**	**Q3 (75%)**	**Min**	**Max**
`Amount`	6054,89	12030,15	18036,33	50,01	24039,93
`V1`	-0.57	-0.09	0.83	-3.50	2.23
`V2`	-0.49	-0.14	0.34	-49.97	4.36
`V3`	-0.65	0.00	0.63	-3.18	14.13
`V4`	-0.66	-0.07	0.71	-4.95	3.20
`V5`	-0.29	0.08	0.44	-9.95	42.72
`V6`	-0.45	0.08	0.50	-21.11	26.17
`V7`	-0.28	0.23	0.53	-4.35	217.87
`V8`	-0.19	-0.11	0.05	-10.76	5.96
`V9`	-0.57	0.09	0.56	-3.75	20.27
`V10`	-0.59	0.26	0.59	-3.16	31.72
`V11`	-0.70	-0.04	0.75	-5.95	2.51
`V12`	-0.83	0.16	0.74	-2.02	17.91
`V13`	-0.70	0.02	0.69	-5.96	7.19
`V14`	-0.87	0.23	0.75	-2.11	19.17

Variable	Q1 (25%)	Médiane	Q3 (75%)	Min	Max
V15	-0.62	-0.04	0.67	-3.86	14.53
V16	-0.72	0.13	0.66	-2.21	46.65
V17	-0.62	0.27	0.52	-2.48	6.99
V18	-0.56	0.09	0.54	-2.42	6.78
V19	-0.57	-0.03	0.56	-7.80	3.83
V20	-0.35	-0.12	0.25	-78.15	29.87
V21	-0.17	-0.04	0.15	-19.38	8.09
V22	-0.49	-0.03	0.46	-7.73	12.63
V23	-0.24	-0.06	0.16	-30.30	31.71
V24	-0.65	0.02	0.70	-4.07	12.97
V25	-0.55	-0.01	0.55	-13.61	14.62
V26	-0.63	-0.01	0.67	-8.23	5.62
V27	-0.31	-0.17	0.33	-10.50	113.23
V28	-0.23	-0.01	0.41	-39.04	77.26

Table 11: Statistiques descriptives des variables quantitatives

Création de notre propre autoencodeur

Code 1: Autoencodeur codé avec TensorFlow

```
import tensorflow as tf
import numpy as np
import pandas as pd

class AE(tf.Module):
    def __init__(self, contamination=0.1, learning_rate=0.001, epochs=10,
       batch_size=32, early_stopping=True, patience=5, min_delta=1e-3,
       name=None):

        super().__init__(name=name)

        #On initialise les parametres necessaires pour l'autoencodeur
        self.contamination = contamination
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.batch_size = batch_size
        self.early_stopping = early_stopping
        self.patience = patience   #Nombre de pas de gradient qui n'
           ameliorent pas l'erreur de reconstruction autorises
        self.min_delta = min_delta   #Contribution minimum d'un pas de
           gradient pour ne pas etre considere par l'early stopping

        #On initialise les variables qui contiendront les resultats de l'
           autoencodeur
        #Pour le train set :
        self.epoch_losses_ = []   #Liste des pertes moyenne par epoch
        self.train_reconstruction_ = 0   #Reconstruction finale
        self.train_reconstruction_errors_ = []   #Erreur de reconstruction
            pour chaque observation
        self.train_outliers_ = []   #Liste de booleen indiquant les
           outliers detectes par le modele
        #Pour le dernier dataset utilise :
        self.reconstruction_ = 0
```

```
        self.reconstruction_errors_ = []
        self.outliers_ = []

        #On initialise les parametres necessaires a la detection d'
            anomalies
        self.threshold_ = 0   #Palier a partir duquel le modele considere
            l'erreur de reconstruction comme trop elevee

    def initialization(self, input_dim):
        hidden_dim = input_dim // 2
        latent_dim = hidden_dim // 2

        #Encodeur

        #Couche 1
        self.w1 = tf.Variable(tf.random.normal([input_dim, hidden_dim]),
            name="w1")
        self.b1 = tf.Variable(tf.zeros([hidden_dim]), name="b1")
        #Couche 2
        self.w2 = tf.Variable(tf.random.normal([hidden_dim, latent_dim]),
            name="w2")
        self.b2 = tf.Variable(tf.zeros([latent_dim]), name="b2")

        #Decodeur

        #Couche 1
        self.w3 = tf.Variable(tf.random.normal([latent_dim, hidden_dim]),
            name="w3")
        self.b3 = tf.Variable(tf.zeros([hidden_dim]), name="b3")
        #Couche 2
        self.w4 = tf.Variable(tf.random.normal([hidden_dim, input_dim]),
            name="w4")
        self.b4 = tf.Variable(tf.zeros([input_dim]), name="b4")

        self.optimizer=tf.keras.optimizers.Adam(learning_rate=self.
            learning_rate)   #Optimiseur

    def activation(self, x):
        return tf.nn.relu(x)

    def output_activation(self, x):
        return tf.nn.sigmoid(x)

    def forward(self, x):

        #Encodeur
        h1 = self.activation(tf.matmul(x, self.w1) + self.b1)
        z = self.activation(tf.matmul(h1, self.w2) + self.b2)

        #Decodeur
        h2 = self.activation(tf.matmul(z, self.w3) + self.b3)
        x_hat = self.output_activation(tf.matmul(h2, self.w4) + self.b4)

        return x_hat
```

```
    def decision_function(self, dataset):
        dataset_np = dataset.values.astype("float32")
        dataset_tensor = tf.convert_to_tensor(dataset_np)
        self.reconstruction_ = self.forward(dataset_tensor)
        self.reconstruction_errors_ = tf.reduce_mean(tf.square(
            dataset_tensor - self.reconstruction_), axis=1).numpy()

        return self.reconstruction_errors_

    def predict(self, dataset):
        dataset_np = dataset.values.astype("float32")
        dataset_tensor = tf.convert_to_tensor(dataset_np)
        self.reconstruction_ = self.forward(dataset_tensor)
        self.reconstruction_errors_ = tf.reduce_mean(tf.square(
            dataset_tensor - self.reconstruction_), axis=1).numpy()

        self.outliers_ = (self.reconstruction_errors_ > self.threshold_).
            astype(int)

        return self.outliers_

    def fit(self, dataset):
        #On initialise les poids et biais
        self.initialization(dataset.shape[1])

        #On batch le dataset
        dataset_np = dataset.values.astype("float32")
        dataset_sliced = tf.data.Dataset.from_tensor_slices(dataset_np)
        dataset_batched = dataset_sliced.batch(self.batch_size)

        #Initialisation pour l'early stopping
        best_loss = np.inf
        patience_counter = 0

        for epoch in range(self.epochs):
            epoch_loss = 0

            for batch in dataset_batched:

                with tf.GradientTape() as tape:
                    reconstruction = self.forward(batch)
                    loss = tf.reduce_mean(tf.square(batch - reconstruction
                        ))

                #On calcule les gradients pour chaque poids et biais
                gradients = tape.gradient(loss, self.trainable_variables)

                #On modifie les poids et biais via l'optimiseur grace aux
                    gradients calcules
                self.optimizer.apply_gradients(zip(gradients, self.
                    trainable_variables))
                #Cela revient a faire w <- w - learning_rate *
                    loss_gradient (avec des optimisations supplementaires)

                #On additionne la perte sur chacun des batchs de l'epoch
```

```
                epoch_loss += loss.numpy()

            avg_epoch_loss = epoch_loss / len(dataset_batched)
            self.epoch_losses_.append(float(avg_epoch_loss))

            #Verification pour l'early stopping
            if self.early_stopping:
                if avg_epoch_loss < best_loss - self.min_delta:
                    best_loss = avg_epoch_loss
                    patience_counter = 0
                else:
                    patience_counter += 1
                    if patience_counter >= self.patience:
                        print("Early Stopping")
                        break

            #On affiche la perte moyenne sur chacune des epochs
            print(f"Epoch {epoch+1}, Loss: {avg_epoch_loss:.4f}")

        dataset_tensor = tf.convert_to_tensor(dataset_np)
        self.train_reconstruction_ = self.forward(dataset_tensor)
        self.train_reconstruction_errors_ = tf.reduce_mean(tf.square(
           dataset_tensor - self.train_reconstruction_), axis=1).numpy()

        #On convertit la reconstruction pour qu'elle ait la meme forme que
            le dataset de depart
        self.train_reconstruction_ = pd.DataFrame(self.
           train_reconstruction_.numpy(), columns=dataset.columns, index=
           dataset.index)

        self.threshold_ = np.percentile(self.train_reconstruction_errors_,
            100 * (1 - self.contamination))
        self.train_outliers_ = (self.train_reconstruction_errors_ > self.
           threshold_).astype(int)
```

Tentative de supervision du problème

Code 2: Fonction permettant de bruiter nos données

```
def create_noise(df, quant_col, frac=0.01, cols_to_perturb=4, size_noise
   =5):
    df_out = df.copy()
    n = int(frac * len(df))
    indices = np.random.choice(df.index, size=n, replace=False)

    for i in indices:
        cols = np.random.choice(df.columns, size=cols_to_perturb, replace=
           False)
        for col in cols:
            if col in quant_col:
                df_out[col] = df_out[col].astype(float)
                df_out.loc[i, col] += np.random.normal(0, size_noise * df[
                   col].std())
            else:
                df_out.loc[i, col] = np.random.choice(df[col].unique())
```

```
    outliers = np.zeros(len(df))
    for i in indices:
        outliers[i] = 1

    return df_out, outliers
```

SHAP : Interprétation et compatibilité avec PyOD

Les différents *explainers*

SHAP propose plusieurs modules pour s'adapter au modèle utilisé :

- **TreeExplainer** : très rapide pour XGBoost, LightGBM, Random Forest.
- **KernelExplainer** : modèle générique (black-box), basé sur LIME.
- **LinearExplainer** : pour les modèles linéaires.
- **DeepExplainer** : pour les réseaux de neurones (Keras/Tensorflow).

Compatibilité avec PyOD

SHAP n'est pas directement compatible avec les modèles de détection d'anomalies de PyOD car :

- Certains modèles ne possèdent pas de méthode `predict_proba()`.
- Les sorties sont des scores d'anomalie, souvent non calibrés.

Solution : Utiliser un modèle de `scikit-learn` compatible (comme `IsolationForest`) ou construire une fonction prédictive customisée pour le `KernelExplainer`.

Interprétation du graphique SHAP

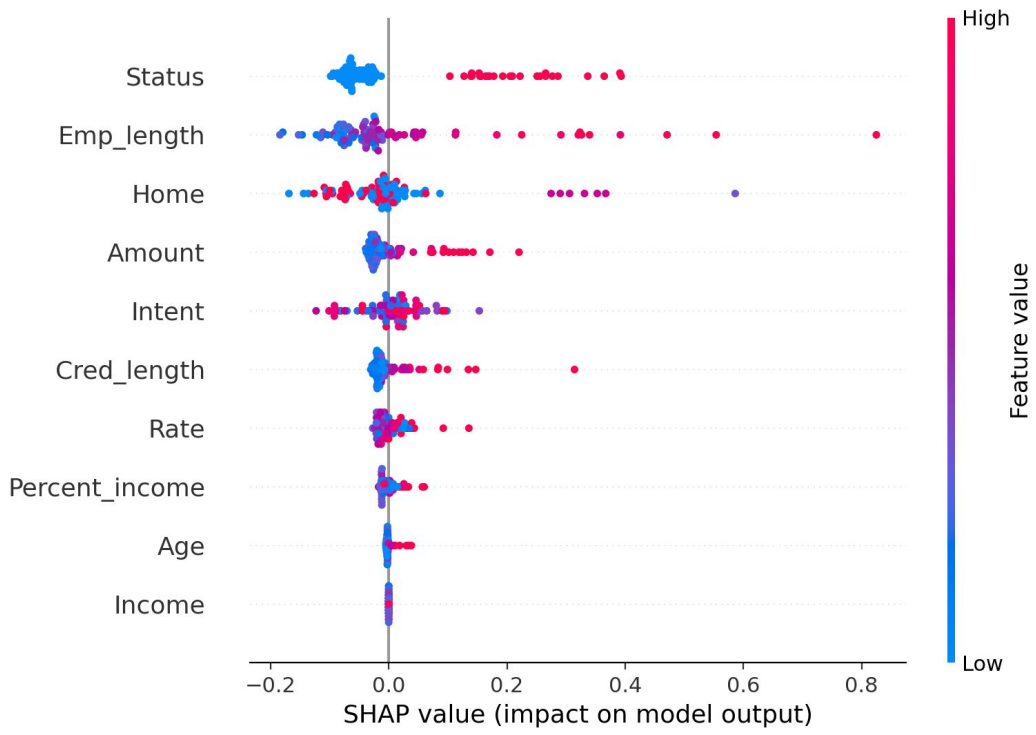

Figure 12: SHAP Summary Plot – Modèle KNN (score d'anomalie)

- Chaque **ligne horizontale** représente une feature du jeu de données.
- Chaque **point** est une observation (transaction) individuelle.
- La **couleur** des points indique la valeur de la feature :
  - Bleu : valeur faible
  - Rouge : valeur élevée
- L'**axe horizontal (valeurs SHAP)** mesure l'impact de la feature sur le score de sortie :
  - Valeurs SHAP positives : la feature **augmente** le score d'anomalie
  - Valeurs SHAP négatives : la feature **réduit** le score d'anomalie

Lecture locale : Chaque point donne l'explication d'une transaction. On peut observer comment chaque variable agit sur le score pour chaque observation.
Lecture globale : La distribution des points autour de zéro permet d'identifier l'effet global de chaque variable. Plus une ligne est étalée horizontalement, plus la variable est importante.

Au final, ce graphique SHAP permet de :

- Identifier les features les plus critiques pour le score d'anomalie (ici : `V11`, `V1`, `V4`...)
- Comprendre si ce sont des valeurs hautes ou basses qui déclenchent l'anomalie
- Fournir une interprétation à la fois locale (point par point) et globale (feature par feature)

References

Nidula Elgiriyewithana. Credit card fraud detection dataset, 2023. URL https://www.kaggle.com/dsv/6492730.

Kai Ming Ting Fei Tony Liu and Zhi-Hua Zhou. Isolation forest. *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, 2008. URL https://doi.org/10.1109/ICDM.2008.17.

Nicolas Goix. How to evaluate the quality of unsupervised anomaly detection algorithms? 2016. URL https://doi.org/10.48550/arXiv.1607.01152.

Hans Hofmann. Statlog (german credit data), 1994. URL https://doi.org/10.24432/C5NC77.

Raymond T. Ng Markus M. Breunig, Hans-Peter Kriegel and Jörg Sander. Lof: identifying density-based local outliers. *ACM SIGMOD Record*, 29, 2000. URL https://doi.org/10.1145/335191.335388.

Takehisa Yairi Mayu Sakurada. Anomaly detection using autoencoders with nonlinear dimensionality reduction. *MLSDA*, 2014. URL https://dl.acm.org/doi/abs/10.1145/2689746.2689747?

Minh-Nghia Nguyen and Ngo Anh Vien. Scalable and interpretable one-class svms with deep learning and random fourier features. *CoRR*, abs/1804.04888, 2018. URL http://arxiv.org/abs/1804.04888.

Nandita Pore. Credit-risk-analysis. URL https://www.kaggle.com/datasets/nanditapore/credit-risk-analysis.

Rajeev Rastogi Sridhar Ramaswamy and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. *ACM SIGMOD Record*, 29, 2000. URL https://dl.acm.org/doi/10.1145/335191.335437.