

Détection de fraude bancaire

CI 2 A – Projet STATISTIQUE

Raport final

28-avr.-21

*Equipe:*

BELAYD Hajar

BELMOUDDEN Manar

ECHCHAIR Zakaria

*Professeur :*

Pr. BENJELLOUN Saad

*Encadrante :*

Pr. CHAIRI Ikram

**Table des matières**

[1 Introduction 2](#_Toc70260727)

[2 Présentation de la base de données 3](#_Toc70260728)

[2.1 Description des variables : 3](#_Toc70260729)

[3 Traitement et analyse des données 4](#_Toc70260730)

[3.1 Description de la base données : 4](#_Toc70260731)

[3.2 Nombre de transactions frauduleuses et validées : 5](#_Toc70260732)

[3.3 Corrélation entre les deux variables « type » et « isFraud » : 6](#_Toc70260733)

[3.4 Pertinence de la variable « isFlaggedFraud » : 7](#_Toc70260734)

[3.5 La fréquence de la variable « step » : 7](#_Toc70260735)

[3.6 Analyse des montants transférés lors des transactions : 8](#_Toc70260736)

[3.7 Transactions à solde nul : 10](#_Toc70260737)

[3.8 Analyse du type de transaction : 10](#_Toc70260738)

[3.9 L’origine et la destination de la transaction : 10](#_Toc70260739)

[3.10 Analyse de l’erreur de la mise à jour du solde : 11](#_Toc70260740)

[4 Les tests de dépendance 12](#_Toc70260741)

[4.1 Test khi-deux : 12](#_Toc70260742)

[4.1.1 Application : 13](#_Toc70260743)

[4.1.1.1 Variable « type » 13](#_Toc70260744)

[4.1.1.2 Variable « amount\_cat » 14](#_Toc70260745)

[4.2 Test Anova : 15](#_Toc70260746)

[5 Application des modèles de détection des fraudes 17](#_Toc70260747)

[5.1 Préparation de la dataset. 17](#_Toc70260748)

[5.2 Traitement de la base des données 17](#_Toc70260749)

[5.3 Équilibrage de la base des données 18](#_Toc70260750)

[5.3.1 La technique de « undersampling » 18](#_Toc70260751)

[5.4 Modèle logistique 20](#_Toc70260752)

[5.5 Deuxième technique d’équilibrèrent de la data 24](#_Toc70260753)

[5.6 Arbres de décisions 25](#_Toc70260754)

[5.7 Gradient boosting method 29](#_Toc70260755)

[6 Conclusion : 32](#_Toc70260756)

[7 Annexe 33](#_Toc70260757)

# Introduction

Ces dernières années ont vu une augmentation rapide des tentatives de fraude bancaire dans les systèmes financiers africains[[1]](#footnote-1), ce qui rend la détection de la fraude non seulement importante mais aussi difficile. Malgré d'innombrables efforts, des sommes énormes sont perdues à cause de la fraude.

La fraude peut se produire en utilisant diverses méthodes telles que le vol des cartes de crédit, la comptabilité trompeuse, les courriels de phishing, etc. En raison de la faible incidence de la fraude sur une large population, sa détection est à la fois importante et difficile.

L’obtention d’accès à des ensembles de données de transactions mobiles pour la recherche paraît une tâche très difficile en raison de la nature intrinsèquement confidentielle de ces transactions. Ce processus prend du temps et empêche les chercheurs à se concentrer sur le problème principal, qui est d'effectuer des expériences sur les données et de trouver des nouvelles méthodes de résoudre des problèmes tels que le problème qui a été à l'origine de notre projet, à savoir la détection de la fraude sur les comptes financiers.

Les données synthétiques du projet sont générées à l’aide d’un simulateur PaySim. Ce simulateur génère des ensembles de données synthétiques similaires à des ensembles de données réels provenant des transactions d'argent mobiles et injecte des comportements malveillants pour évaluer ultérieurement les performances des méthodes de détection des fraudes.

Cette opération est réalisée par le biais de la simulation informatique, en particulier la simulation basée sur les agents. Certes, cette dernière a une grande valeur dans le contexte de détection de la fraude bancaire, mais les modèles créés dans la machine espionnent le comportement humain lors des transactions et détectent avec précision les tentatives de vol.

Dans cet article, PaySim simule des transactions d’argent mobiles sur la base d’un échantillon de transactions réelles extraites d’un mois de registres financiers d’un service d’argent mobile implémenté dans un pays africain. Grâce à l'analyse statistique et à l'analyse des réseaux sociaux, PaySim est en mesure de générer des résultats conformes à l'ensemble des données originales.

La portée de cet article couvre principalement l’exploration de données, l’étude des tests statistiques. Ceci, peut aider à anticiper et à identifier rapidement les fraudes et à y répondre afin de limiter les coûts. Grâce aux outils d'exploration de données, il est possible d'examiner un nombre considérable de transactions afin d'identifier les modèles de détection et de distinguer les transactions frauduleuses. Vers la fin, les résultats de la modélisation sont exposés et analysés d’une manière critique.

# Présentation de la base de données

La base de données « Fraud.csv » est un échantillon d’un mois de simulation générée à l’aide du simulateur PaySim, et qui a été enregistrée 6 362 620 transactions.

L’exploration de données, l’étude statistique et la réalisation des modèles déposent sur une base de données comprenant 11 variables significatives.

# Description des variables :

Les variables de cette base de données ainsi que leurs descriptions sont les suivantes :

**Variable « step » *-integer-* :**

Correspond à une unité de temps dans le monde réel. Dans ce cas, « 1 step » correspond à 1 heure de temps. Total des « steps » dans la base de données est 744 (simulation de 30 jours).

**Variable « type » *-string-* :**

Regroupe les cinq types de transaction : CASH-IN, CASH-OUT, DEBIT, PAYMENT et TRANSFER[[2]](#footnote-2).

* **CASH-IN :** le paiement en espèces à un commerçant pour augmenter le solde d'un compte.
* **CASH-OUT :** est l’opposé du CASH-IN. Il s'agit de retirer de l'argent du compte commerçant, ce qui diminue le solde du compte.
* **DEBIT :** est un processus similaire au CASH-OUT et consiste à envoyer de l'argent du service d'argent mobile vers un compte bancaire.
* **PAYMENT :** le processus de paiement de biens ou de services à des commerçants qui diminue le solde du compte et augmente le solde du destinataire.
* **TRANSFER :** le processus d'envoi d'argent à un autre compte via la plateforme d'argent mobile.

**Variable « amount »*-float-* :**

Représente le montant de la transaction en monnaie locale.

**Variable « nameOrig » *-string-* :**

Détermine le client qui envoie le montant de la transaction.

**Variable « nameDest » *-string-* :**

Détermine le client qui reçoit le montant de la transaction.

**Variable « oldbalanceOrg »*-float-* :**

Représente le solde initial avant la transaction.

**Variable « newbalanceOrig » *-float-* :**

Représente le nouveau solde après la transaction.

**Variable « oldbalanceDest » *-float-* :**

Indique le destinataire du solde initial avant la transaction.

**Variable « newbalanceDest » *-float-* :**

Indique le destinataire du nouveau solde après la transaction.

**Variable « isFraud » *-boolean-* :**

Décrit si la transaction est frauduleuse (encodée à 1) ou valide (encodée à 0)

Il s’agit des transactions effectuées par les agents frauduleux au sein de la simulation. Dans ce cas, le comportement frauduleux des agents vise à faire des profits en prenant le contrôle des comptes des clients et en essayant de vider les fonds en les transférant vers un autre compte, puis en sortant du système.

**Variable « isFlaggedFraud » *-boolean-* :**

Indique si la transaction est signalée comme frauduleuse (encodée à 1) ou pas du tout (encodée à 0).

Une observation est signalée si la transaction est frauduleuse et si elle implique un transfert de plus de 200 000 en monnaie locale.

# Traitement et analyse des données

Après avoir défini nos variables, l’analyse exploratoire des données nous permet de vérifier certaines hypothèses sur les transactions frauduleuses et d'obtenir des interprétations visuelles des données. En effet, cette analyse consiste à empêcher les transactions suspectes, à détecter des données aberrantes, et réaliser des études statistiques descriptives.

# Description de la base données :

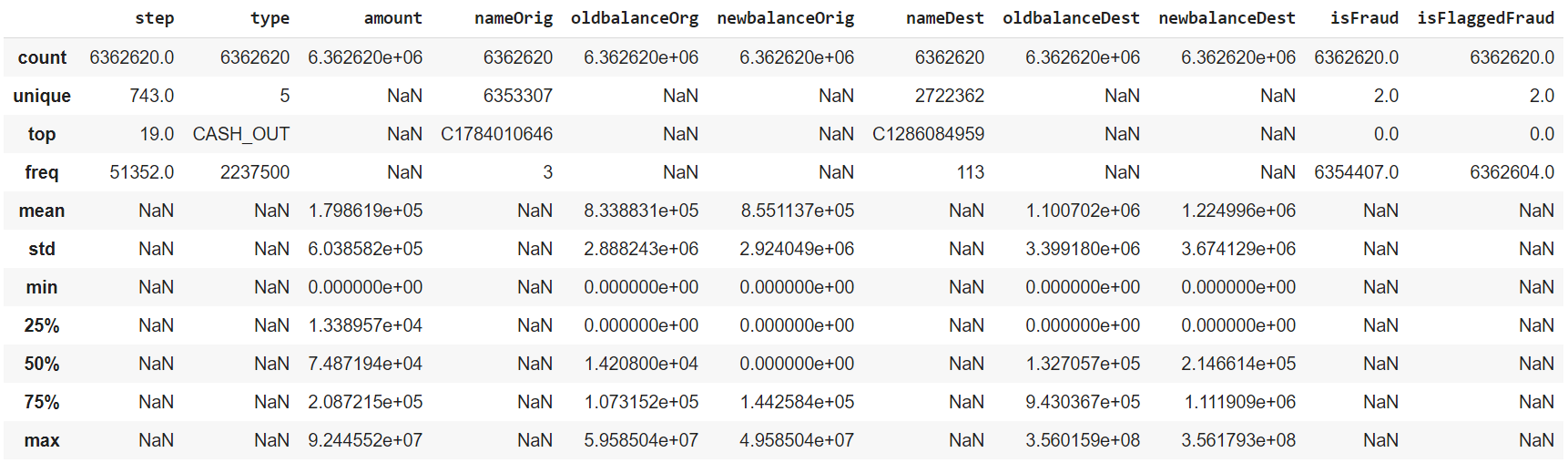


Figure 3.1‑1 : description de la base de données

Principaux acquis :

* Il n’y a pas de valeurs manquantes
* Il y a un peu plus de 6 millions d'observations et 11 variables
* La plupart des transactions portent sur des montants inférieurs à 1 million de devise
* La plupart des observations dans la base de données concernent des transactions valides, de sorte que tout modèle lié à l'identification des transactions frauduleuses peut être difficile à les percevoir
* Les données sont également déséquilibrées.
* Dans un échantillon d'observations, il y a de nombreux cas où ce qui apparaît sur le compte du destinataire (oldbalanceDest, newbalanceDest) n'a pas de sens (par exemple, la toute première observation concerne un paiement de 9839.64 et pourtant, le solde avant et après la transaction est égal à 0).

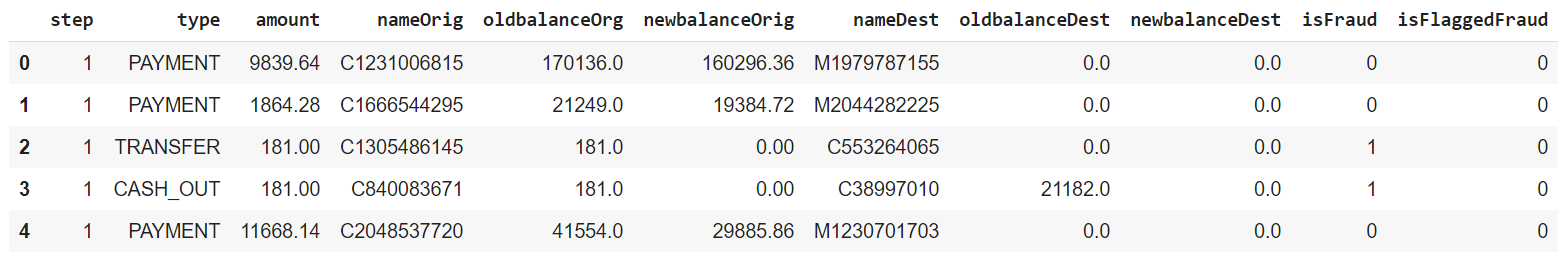


Figure 3.1‑2 : Exemple d’un solde nul avant et après la transaction

Comme les données ne sont pas équilibrées, nous allons comparer visuellement les transactions frauduleuses aux transactions valides et voir s'il existe des modèles importants qui pourraient être utiles.

# Nombre de transactions frauduleuses et validées :

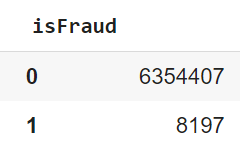


Figure 3.2‑1 : Nombre des fraudes et non fraudes

Le nombre de point de données est de 6 362 604, dont 8 197 sont des fraudes et 6 354 407 ne le sont pas. Cela montre que les données sont fortement déséquilibrées.

La distribution des variables cibles montre que nous avons à faire face à un problème fortement déséquilibré car il y a beaucoup plus de transactions authentiques que de transactions frauduleuses.

Par conséquent, nous devons être prudents dans la présentation de nos résultats et la précision n'est peut-être pas un bon paramètre à présenter pour ce problème de classification. Nous nous concentrerons sur la précision.

# Corrélation entre les deux variables « type » et « isFraud » :

Les transactions bancaires enregistrées dans la base de données constituent de cinq types différents : DEBIT, CASH-IN, CASH-OUT, PAYMENT et TRANSFER.

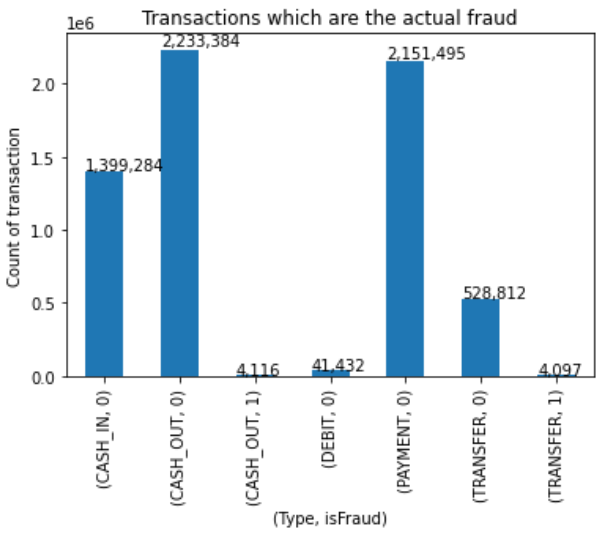


Figure 3.3‑1 : Transactions frauduleuses et non frauduleuses

D’après l’analyse exploratoire de la base de données, nous observons que les transactions frauduleuses ne se produisent que lorsque le type de la transaction est CASH-OUT ou TRANSFER.

Le nombre de TRANSFER frauduleux est 4097.

Le nombre de CASH-OUT frauduleux est 4116.

Comme la fraude ne se produit que si le type de transaction est CASH-OUT ou TRANSFER, nous sélectionnons que ces cas pour augmenter l’efficacité du modèle.

Dans la suite, nous travaillons sur la base de données qui contient que deux types de transactions : CASH-OUT et TRANSFER.

# Pertinence de la variable « isFlaggedFraud » :

La variable « isFlaggedFraud » est décrite comme les transactions qui ont été marquées comme frauduleuses.

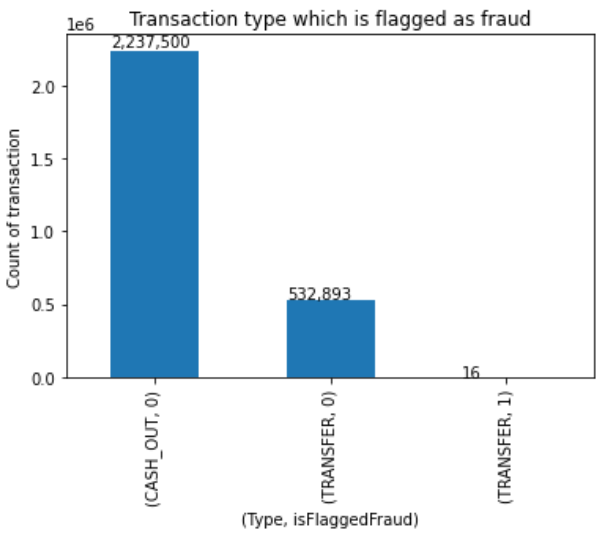


Figure 3.4‑1 : Transactions signalées comme frauduleuses par la machine

Pour être signalée comme frauduleuse, la transaction doit être frauduleuse et impliquer un transfert de plus de 200 000 de devise.

Dans un ensemble de données modifié comportant plus de 2 millions d'observations, une variable qui n'attire l'attention que sur 16 observations est insignifiante et qui appartiennent toutes au type : TRANSFER.

Nous essayons par la suite de développer la détection de la fraude qui ne dépend pas d’un schéma de détection de la fraude préexistant.

Pour cette raison, nous ne prenons pas en compte la variable « isFlaggedFrausd » dans l’analyse.

# La fréquence de la variable « step » :

La variable « step » correspond à une unité de temps dans le monde réel.

Nous vérifions par la suite les attaques frauduleuses sur une période de temps. Le graphique ci-dessous montre les attaques frauduleuses chaque heure au cours des 744 heures.

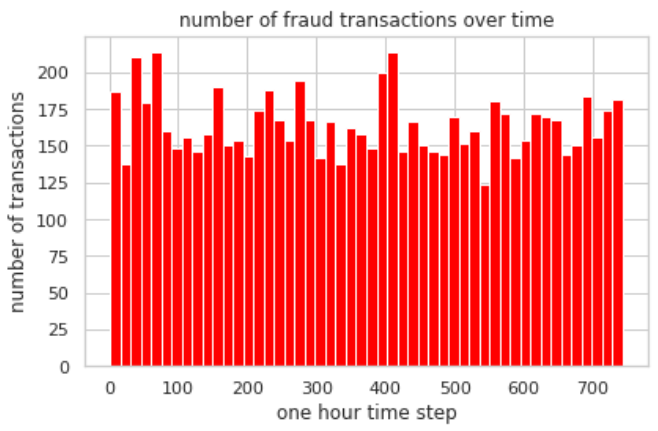


Figure 3.5‑1 : Nombre de transactions frauduleuses en 1 step

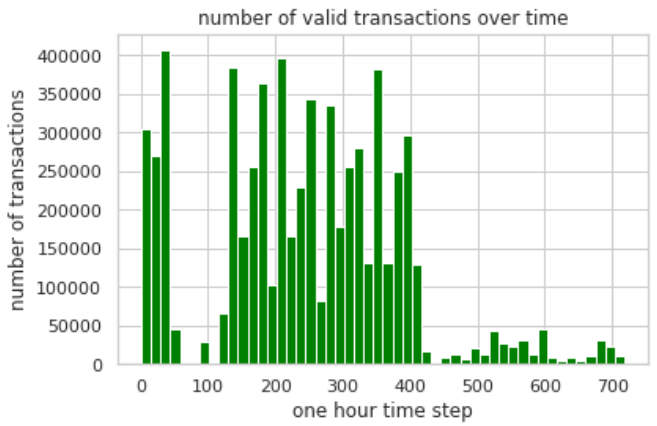


Figure 3.5‑2 : Nombre de transactions non frauduleuses en 1 step

Les visualisations montrent le nombre de transactions pour chaque pas de temps au cours d'un mois.

Dans chaque pas de temps, il existe certainement des transactions frauduleuses, et bien évidemment des transactions valides.

Une grande partie des transactions valides se produisent entre le 0e et le 60e pas de temps environ, ainsi qu'entre le 110e et le 410e pas de temps.

La fréquence à laquelle les transactions frauduleuses se produisent, est homogène en fonction du temps.

# Analyse des montants transférés lors des transactions :

Après avoir importé la base de données, nous affichons les statistiques descriptives des montants transférés lors des transactions frauduleuses et non frauduleuses.

Tableau 1 : Statistiques des montants transférés

|  |  |  |
| --- | --- | --- |
|  | Montants transférés lors des transactions frauduleuses | Montants transférés lors des transactions non frauduleuses |
| mean | 1.467967e+06 | 3.141155e+05 |
| std | 2.404253e+06 | 8.771441e+05 |
| min | 0.000000e+00 | 1.000000e-02 |
| max | 1.000000e+07 | 9.244552e+07 |

Nous remarquons que lors des transactions frauduleuses, le montant transféré est plafonné à 10 millions de devises.

Alors que pour les transactions valides, le montant transféré est plafonné à environ 92,4 millions de devises.

Nous observons que la moyenne des montants de transactions est très légèrement différente, donc on ne peut rien conclure. Par contre, le min nous indique que les transactions avec 0 solde comme montant ne peuvent être que frauduleuse. En ce qui concerne, le max des montants nous indique qu’il n’existe pas un seuil à partir duquel toutes les transactions seront frauduleuses.

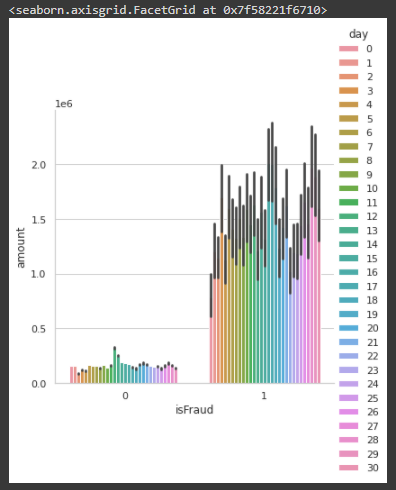


Figure 3.6‑1 : Histogramme de la moyenne des montants des transactions valides et frauduleuses dans chaque jour

Le graphe ci-dessus représente la distribution des montants lors des transactions frauduleuses et non frauduleuses pendant un mois de simulation.

Nous remarquons une distribution importante de grands montants lors des transactions frauduleuses, ainsi la majorité des transactions valides effectuées avec de faibles montants d’argent par jour.

Alors, nous pouvons interpréter cette observation comme suit :

Les personnes qui effectuent des transactions d'un montant plus élevé, elles sont plus susceptibles d'être victimes de fraudes et attirent donc les cyber-voleurs.

# Transactions à solde nul :

La base de données comporte plusieurs transactions ayant un solde nul sur le compte de destination, tant avant qu'après la transaction d'un montant non nul.

La fraction de ces transactions, où zéro indique probablement une valeur manquante, est beaucoup plus importante dans les transactions frauduleuses (**49,55 %**) que dans les transactions authentiques (**0,06 %**).

Comme le solde nul du compte destinataire est un indicateur fort de fraude, nous ne reprochons pas le solde du compte (avant que la transaction ne soit effectuée) avec une statistique ou à partir d'une distribution avec un ajustement ultérieur pour le montant de la transaction.

Cela masquerait cet indicateur de fraude et ferait apparaître les transactions frauduleuses comme authentiques.

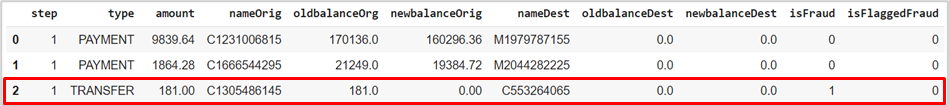


Figure 3.7‑1 : Exemple de transaction frauduleuse signalée valide

# Analyse du type de transaction :

Isoler les fraudes dans une dataset pour pouvoir détecter des particularités des fraudes nous a permis de remarquer que la succession d’une transaction fraude de type Transfer et une autre de type CASH\_OUT dans la même STEP et avec le même montant est un signe presque sûr de fraude.La totalité de la data des fraudes seulement est situées ainsi (succession de TRANSFER et CASH\_OUT avec le même montant dans la même step).

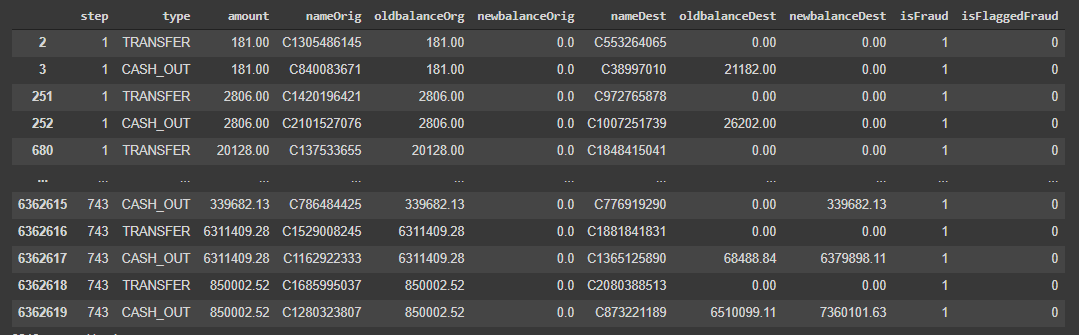


Figure 3.8‑1 : Nouvelle base de données des deux types de transactions Transfer et cash-out

# L’origine et la destination de la transaction :

Après analyse de la data, il s’est avéré que toutes les fraudes sont des transactions des clients vers des clients, Aucune transaction d’un client vers un marchand n’est connue comme fraude.



Figure 3.9‑1 : Histogramme des fraude de transactions C2C et C2M

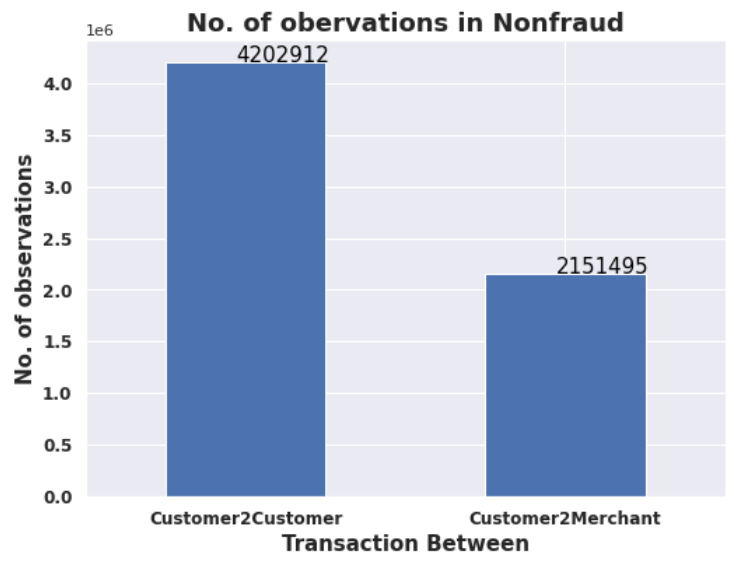


Figure 3.9‑2 : Histogramme des NON-FRAUDES de transactions C2C et C2M

# Analyse de l’erreur de la mise à jour du solde :

Dans notre data, il y’a quatre variables (newbalanceDest, olbalanceDest, newbalanceOrig, olbalanceOrig) dont on peut se servir pour analyser l’erreur de la mise à jour du solde du compte et cela en générant deux variables error\_dest et error\_orig (vue que nous avons remarqué qu’il y a un problème de mise à jour du solde) avec :

error\_dest = newbalanceDest - (amount + olbalanceDest)

error\_orig = newbalanceOrig - (olbalanceDest - amount)

En isolant seulement les transactions qui ont subi une mise à jour dans un jeu de donnée, on obtient la table de contingence suivante :

|  |  |  |
| --- | --- | --- |
| **isFraud** | **0** | **1** |
| **Error\_dest=0** | 1601336 | 2889 |
| **Pourcentage** | 39,68 % | 35% |

De même pour error\_orig :

|  |  |  |
| --- | --- | --- |
| **isFraud** | **0** | **1** |
| **Error\_orig=0** | 0 | 16 |
| **Pourcentage** | 0 % | 0,001% |

Le problème de la mise à jour du solde peut être aperçu comme suspect, alors que ce n’est pas le cas vu que cela existe dans des transactions valides aussi.

# Les tests de dépendance

Dans cette partie, nous allons effectuer un test de dépendance entre les variables indépendantes et variable « is\_Fraud ».

# Test khi-deux :

Le test Khi-deux est utilisé pour tester l’hypothèse nulle d’absence de relation entre deux variables catégorielles.

Si deux variables sont dépendantes alors la variation de l’une influence celle de l’autre.

Le test de khi-deux est une statistique permettant de comparer les fréquences observées dans un échantillon avec des fréquences théorique qui découlent des hypothèses statistiques.

L’idée des tests du khi-deux est de comparer les valeurs observées et les valeurs moyennes qu’on observait si l’hypothèses nulle est vraie c'est-à-dire les valeurs théoriques. On cherche à établir si la différence entre les valeurs observées et les valeurs théoriques est importante ou simplement due à une variation aléatoire.

La statistique de khi-deux est données par :

Avec la valeur observée pour le nombre de fois où la modalité i et la valeur moyenne attendue.

Pour notre test d’indépendances on utilise les deux hypothèses ci-dessous :

S’il y a indépendance on devrait avoir

Posons la fréquence attendue pour les modalités i et j s’il y avait indépendance. La statistique pour le test du khi-deux est donnée par

Où k est le nombre de modalité de X et m le nombre de modalité de Y. Cette statistique est la mesure de dépendance entre les variable aléatoire X et Y.

*Condition d’application :*

Ce test approximatif est valide si

* Pour tout i et j.
* Il n’y a pas plus de 20 % des valeurs de plus petite que 5.

### **Application :**

### ***Variable « type »***

Les hypothèses :

: il y a une dépendance entre la variable qualitative type et la variable qualitative fraude.

: il y a une indépendance entre la variable qualitative type et la variable qualitative fraude.

Le tableau de contingence est le suivant

|  |  |  |
| --- | --- | --- |
|  | **CASH\_OUT** | **TRANSFER** |
| 0 | 2233384 | 528812 |
| 1 | 4116 | 4097 |

Condition d’application du test de khi deux est vérifiée

Tableau théorique :

|  |  |  |
| --- | --- | --- |
|  | **CASH\_OUT** | **TRANSFER** |
| 0 | 2230866.832 | 531329.168 |
| 1 | 6633.168 | 1579.832 |

Tableau des résiduels :

|  |  |  |
| --- | --- | --- |
|  | **CASH\_OUT** | **TRANSFER** |
| 0 | 1.685293 | -3.453268 |
| 1 | -30.906631 | 63.329587 |

Résultats :

|  |  |  |
| --- | --- | --- |
| **X-squared** | **Df** | **p-value** |
| 4978.6 | 1 | <2.2e-16 |

Donc on accepte l’hypothèse nulle ce qui approuve la relation d’indépendance entre la variable type et la variable isFraud.

### ***Variable « amount\_cat »***

Nous avons opté pour a discrétisation de amount.

Les hypothèses :

: il y a une dépendance entre la variable qualitative amount\_cat et la variable qualitative fraude.

: il y a indépendance entre la variable qualitative amount\_cat et la variable qualitative fraude.

Le tableau de contingence est le suivant :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **[0,10)** | **[10,100)** | **[100,1e+03)** | **[1e+03, 1e+04)** |
| 0 | 70 | 713 | 6960 | 70820 |
| 1 | 16 | 2 | 40 | 220 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **[1e+04, 1e+05)** | **[1e+05, 1e+06)** | **[1e+06, 1e+07)** | **[1e+07, 24e+07)** |
| 0 | 755842 | 1799990 | 122438 | 5363 |
| 1 | 1429 | 13800 | 2419 | 287 |

La condition de l’application de test khi deux n’est satisfaite donc vaut mieux unir le premier groupe [10,100) avec [100,1e+03).

Le tableau de contingence devient.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **[0,10)** | **[10, 1e+03)** | **[1e+03, 1e+04)** | **[1e+04, 1e+05)** |
| 0 | 70 | 7673 | 70820 | 755842 |
| 1 | 16 | 42 | 220 | 1429 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **[1e+05, 1e+06)** | **[1e+06, 1e+07)** | **[1e+07, 24e+07)** |
| 0 | 1799990 | 122438 | 5363 |
| 1 | 13800 | 2419 | 287 |

Tableau théorique :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **[0,10)** | **[10, 1e+03)** | **[1e+03, 1e+04)** | **[1e+04, 1e+05)** |
| 0 | 85.74 | 7692.12 | 70829.39 | 755026 |
| 1 | 0.25 | 22.28 | 210.60 | 2244 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **[1e+05, 1e+06)** | **[1e+06, 1e+07)** | **[1e+07, 24e+07)** |
| 0 | 1798442.58 | 124486.85 | 5633.25 |
| 1 | 5347 | 370.14 | 16.74 |

Tableau des résiduels :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **[0,10)** | **[10, 1e+03)** | **[1e+03, 1e+04)** | **[1e+04, 1e+05)** |
| 0 | -1.70 | -0.21 | -0.03 | 0.93 |
| 1 | 31.18 | 3.99 | 0.64 | -17.22 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **[1e+05, 1e+06)** | **[1e+06, 1e+07)** | **[1e+07, 24e+07)** |
| 0 | 1.15 | -5.80 | -3.60 |
| 1 | -21.16 | 106.49 | 66.03 |

Résultats du test :

|  |  |  |
| --- | --- | --- |
| **X-squared** | **Df** | **P-value** |
| 17488.267 | 7 | <2.22044604925031e-16 |

La p value est très petite on peut accepter l’hypothèse nulle. IsFraud est corrélée avec amount discrétisée (amount\_cat).

# Test Anova :

Test Anova ou l’analyse de la variance (ANOVA) compare la variation entre observations à l’intérieur de chaque groupe à la variation entre les groupes. Elle permet donc de tester globalement l’hypothèse nulle selon laquelle les observations de chaque groupe proviennent de populations avec la même moyenne.

L’Anova consiste à construire le test d’hypothèse suivant :

L’hypothèse nulle consiste à ce que La moyenne de la variable dépendante est la même quel que soit les groupes définis par le facteur, il est égal à la moyenne global (en filigrane, le facteur n ’a aucune influence sur la variable dépendante).

L’équation d ’analyse de variance est la suivante :

SCR : somme des carrés résiduels Exprime la variabilité résiduelle, à savoir la variation que le facteur n’arrive pas à expliquer

SCE : somme des carrés expliqués Exprime la variabilité expliquée, à savoir la variation que le facteur explique

SCT : somme des carrés totaux Exprime la variabilité totale des observations

Avec :

**:** Ecart à la moyenne globale.

**:** Ecart entre les groupes (définis par les facteurs).

Ecart à l ’intérieur des groupes.

**Calcul :**

**Application :**

**Variable amount :**

Les hypothèses :

: il y a une dépendance entre la variable quantitative amount et la variable qualitative fraude.

: il y a indépendance entre la variable quantitative amount et la variable qualitative fraude.

Avec la ligne de code si dessous sur R : fit <- aov(amount ~ isFraud, data = data)

On obtient les résultats suivants :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **DF** | **Sum Sq** | **Mean Sq** | **F value Pr>(F)** |
| **IsFraud**  SCE  SCR  CME  CMR  F | 1 | 1.090+16 | 1.090e+16 | 13902<2e-16 |
| **Résiduels** | 2770407 | 2.173e+18 | 7.842e+11 |  |

Et donc, on rejette l’hypothèses H0 il y a une corrélation entre la variable catégorielle isFraud et celle quantitative amount.

# Application des modèles de détection des fraudes

# Préparation de la dataset.

Après avoir importé les bibliothèques[[3]](#footnote-3), nous commençons par l’importation d’un sous ensemble de la base de données où il n’y a que les transferts et les cash-out*s* (afin de réduire le nombre total des observations). On souligne que dans le contexte de « data balancing », le sous-ensemble importé contient toutes les fraudes qui existent dans la base de données originale. Et il a une taille de 2.77 millions d’observations au lieu de 6.36 millions.

*dim(df)*

*## [1] 2770409 11*

Notre dataset importée contient 2770409 (2.77 millions) observations et 11 variables.

# Traitement de la base des données

Nous effectuons le test Chi-2 pour éliminer les variables qui n’ont aucune relation avec la variable dépendante « isFraud ».

*## [1] "Chi-2 p-value pour nameOrig est : 0.65 supérieur au seuil 0.05 donc on va admettre l’hypothèse H0 (il n'y a aucune relation entre nameOrig et isFraud) "*

*## [1] "Chi-2 p-value pour nameDest est : 0 inférieur au seuil 0.05 donc on va rejeter l’hypothse H0 (il n'y a aucune relation entre nameDest et isFraud) alors il y a une relation entre eux.*

*## [1] "chi-2 p-value pour type est : 0 inférieur au seuil 0.05 donc on va rejeter H0 (il n'y a aucune relation entre type et isFraud) alors il y a une relation entre eux.*

Certes p-value de « nameDest » est inférieur au seuil, cependant nous éliminons cette variable parce que le nom du destinataire est unique, autrement dit les gens qui effectuent les fraudes ne vont pas utiliser toujours les mêmes comptes bancaires, et en même temps il ne faut pas lier les fraudes à des comptes spécifiques. Alors, nous gardons la variable « type » parce qu’elle est significative. En revanche, nous supprimons les deux variables « nameDest » et « nameOrig ».

De plus, nous éliminons la variable « isFlaggedFraud » parce qu’elle signifie que la machine a détecté une fraude, cependant nous voulons détecter les fraudes par notre algorithme au lieu d’exploiter un autre déjà implémenté.

*df*=*subset(df, select=-c(isFlaggedFraud,nameOrig,nameDest))*

Nous définissons la variable « type » en tant qu’une variable catégorique.

*df$type*=*as.factor(df$type)*

# Équilibrage de la base des données

### **La technique de « undersampling »**

Nous proposons une discrétisation des variables importantes en plusieurs catégories (que nous choisissons soigneusement), pour effectuer une division de la base de données en gardant la même structure de la base initiale.

A cet égard, nous allons créer deux nouvelles variables catégoriques « amount\_cat » et « step\_cat ».

*df$amount\_cat*=*discretize(df$amount, method = "fixed", breaks = c(*0*,*1000*,*10000*,*100000*,*1000000*,*10000000*,max(df$amount)))  
df$step\_cat*=*discretize(df$step, method = "fixed", breaks = c(*1*,*48*,*96*,*144*,*192*,*240*,*288*,*336*,*384*,*432*,*480*,*528*,*576*,*624*,*672*,*720*))*

Puis, nous concaténons ces variables dans une nouvelle qui s’appelle « new » et qui regroupe les informations liées à « amount\_cat » et « step\_cat », et nous avons opté pour des catégories afin d’éviter les singletons de « new » pendant le split.

Nous proposons de diviser notre data à deux, une base de données des transactions frauduleuses et une autre des transactions non frauduleuses.

*df$new*=*paste(df$amount\_cat,df$step\_cat)  
df\_isFraud*=*df[ which(df$isFraud ==* 1*), ]  
df\_isNotFraud*=*df[ which(df$isFraud ==* 0 *), ]*

Nous prenons un sous-ensemble de la base des non-fraudes de 10000 observations qui représentent la majorité de la base de données originale.

**Ps.** La fonction “createDataPartition” prend une partition de la dataset en gardant les mêmes pourcentages des valeurs dans la variable “new”. Cette fonction n’admet qu’une seule variable “new” comme argument.

Alors, le raisonnement sur “ new” équivalent au raisonnement sur “amount\_cat” et “step\_cat” en même temps.

*my.ids* <- *createDataPartition(df\_isNotFraud$new, p =* 10000*/dim(df\_isNotFraud)[1])  
df\_small* <- *df\_isNotFraud[as.numeric(my.ids[[*1*]]), ]  
nrow(df\_small)*

*## [1] 10048*

Nous remarquons que la partition se constitue de 10048 au lieu de 10000, parce que cette fonction cherche toujours un compromis entre le pourcentage des valeurs de "new"  et le nombre des observations dans le sous-ensemble.

Les graphiques suivants représentent le rapport entre la proportion de chaque catégorie dans la base de données initiale (avant le split) et finale (après le split).

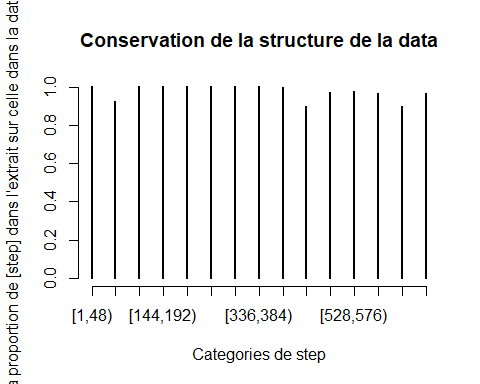


Figure 5.3‑1 : Histogramme du rapport de la proportion de « Step » dans la base de données avant et après split

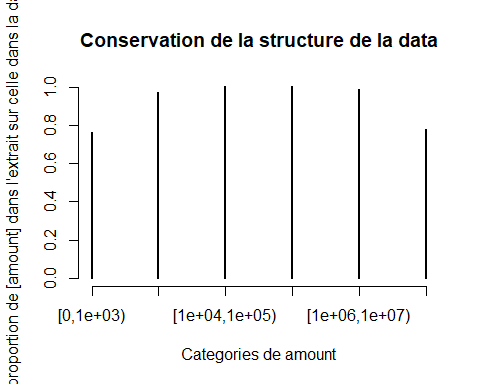


Figure 5.3‑2 : Histogramme du rapport de la proportion de « Amount » dans la base de données avant et après split

Nous remarquons que le rapport est presque 1 (100%) sauf pour quelques catégories parce que les observations de ses catégories sont très peu. Donc l’algorithme n’arrive pas à prendre un nombre entier des lignes en gardant le pourcentage exact. Et ce n’est pas évident d’extraire 10000 observations parmi 2.77 millions en gardant le même pourcentage des catégories.

*split*=*sample.split(df\_isFraud$new,SplitRatio =* 0.75*)   
df\_isFraud\_train* = *subset(df\_isFraud, split==*TRUE*)  
df\_isFraud\_test* = *subset(df\_isFraud, split==FALSE)  
  
split*=*sample.split(df\_isNotFraud$isFraud,SplitRatio =* 0.75*)   
df\_isNotFraud\_train* = *subset(df\_isNotFraud, split==*TRUE*)  
df\_isNotFraud\_test* = *subset(df\_isNotFraud, split==*FALSE*)  
  
split*=*sample.split(df\_small$isFraud,SplitRatio =* 0.75*)   
df\_small\_train* = *subset(df\_small, split==*TRUE*)  
df\_small\_test* = *subset(df\_small, split==*FALSE*)*

Ensuite, nous concaténons l’extrait 75% des 10000 observations des non-fraudes avec 75% des fraudes, en tant que **small training set**. Et les 25% qui restent des non-fraudes avec ceux qui restent des fraudes en tant que **small testing set**.

Et de même, nous concaténons 75% des non-fraudes globaux (2 millions observations) avec 75% des fraudes en tant que **big training set** et bien évidement le reste en tant que **big testing set**.

Train\_big=rbind(df\_isNotFraud\_train,df\_isFraud\_train)  
Train\_big=Train\_big[order(Train\_big$step),]  
  
Test\_big=rbind(df\_isNotFraud\_test,df\_isFraud\_test)  
Test\_big=Test\_big[order(Test\_big$step),]  
  
Train\_small=rbind(df\_small\_train,df\_isFraud\_train)  
Train\_small=Train\_small[order(Train\_small$step),]  
  
Test\_small=rbind(df\_small\_test,df\_isFraud\_test)  
Test\_small=Test\_small[order(Test\_small$step),]

**Ps.** Nous optons pour cette technique afin qu’nous puissons générer des testings sets où les observations “frauds” sont toutes nouvelles pour tous les trainings set en même temps. Nous obtenons traning and testing sets “Train\_small” et “Test\_small” , et de même pour la base de données globale nous obtenons “Train\_big” et “Test\_big”

Finalement, nous supprimons les variables catégoriques qui étaient très importantes pour les splits.

*Train\_small* = *subset(Train\_small ,select=-c(new,amount\_cat,step\_cat))  
df\_isFraud* = *subset(df\_isFraud ,select=-c(new,amount\_cat,step\_cat))  
df\_isNotFraud* = *subset(df\_isNotFraud ,select=-c(new,amount\_cat,step\_cat))  
Test\_small*=*subset(Test\_small ,select=-c(new,amount\_cat,step\_cat))  
Train\_big* = *subset(Train\_big ,select=-c(new,amount\_cat,step\_cat))  
Test\_big* = *subset(Test\_big ,select=-c(new,amount\_cat,step\_cat))  
df* = *subset(df ,select=-c(new,amount\_cat,step\_cat))  
df\_isFraud\_train* = *subset(df\_isFraud\_train ,select=-c(new,amount\_cat,step\_cat))  
df\_isFraud\_test* = *subset(df\_isFraud\_test ,select=-c(new,amount\_cat,step\_cat))  
df\_isNotFraud\_test* = *subset(df\_isNotFraud\_test ,select=-c(new,amount\_cat,step\_cat))  
df\_isNotFraud\_train* = *subset(df\_isNotFraud\_train ,select=-c(new,amount\_cat,step\_cat))  
df\_small\_train* = *subset(df\_small\_train ,select=-c(new,amount\_cat,step\_cat))  
df\_small\_test* = *subset(df\_small\_test ,select=-c(new,amount\_cat,step\_cat))  
df\_small* = *subset(df\_small ,select=-c(amount\_cat,step\_cat,new))*

# Modèle logistique

Nous construisons un modèle logistique par le biais du “Train\_small”, ce modèle est nommé “model\_log\_small” et nous construisons un autre modèle basé sur le grand training set “Train\_big” nommé “model\_log\_big”.

*## Call:  
## glm(formula = isFraud ~ ., family = binomial, data = Train\_small)  
##  
## Coefficients:  
## Estimate Std. Error  
## (Intercept) -7.8994327711799713e+14 1.1504812760211306e+06  
## step 6.5540649153889832e+11 3.0752893220012515e+03  
## typeTRANSFER 4.6324360141676594e+14 1.3627398025354960e+06  
## amount -1.5488475576605036e+09 1.1952770012382885e+00  
## oldbalanceOrg 2.7535761221271753e+09 1.0317785178994725e+00  
## newbalanceOrig -2.9432554087715368e+09 1.1068548688897897e+00  
## oldbalanceDest 4.9151308222373493e+07 6.9916177996587958e-01  
## newbalanceDest -4.8310439952837177e+07 6.6758472295306948e-01  
## z value Pr(>|z|)   
## (Intercept) -686619846.478479981 < 2.22e-16 \*\*\*  
## step 213120270.294569999 < 2.22e-16 \*\*\*  
## typeTRANSFER 339935474.516020000 < 2.22e-16 \*\*\*  
## amount -1295806374.636110067 < 2.22e-16 \*\*\*  
## oldbalanceOrg 2668766672.650829792 < 2.22e-16 \*\*\*  
## newbalanceOrig -2659115925.219460011 < 2.22e-16 \*\*\*  
## oldbalanceDest 70300336.246600002 < 2.22e-16 \*\*\*  
## newbalanceDest -72366005.829400003 < 2.22e-16 \*\*\*  
## ---  
## AIC: 152769.0030630785  
##   
## Number of Fisher Scoring iterations: 25  
## Call:  
## glm(formula = isFraud ~ ., family = binomial, data = Train\_big)  
##   
## Coefficients:  
## Estimate Std. Error  
## (Intercept) -7.2427388581403571e+00 5.0151606124058126e-02  
## step 5.2327700184924483e-03 1.0469348480814869e-04  
## typeTRANSFER 1.5923276588683986e+00 3.8813748190374177e-02  
## amount -1.5309618651108923e-05 4.8315410300082378e-07  
## oldbalanceOrg 2.8779951165448935e-05 4.8311650347781141e-07  
## newbalanceOrig -3.2145368846807167e-05 5.1914649280812398e-07  
## oldbalanceDest 5.7848101608698457e-06 1.1599878020508574e-07  
## newbalanceDest -6.1030767156235014e-06 1.1511133565651106e-07  
## z value Pr(>|z|)   
## (Intercept) -144.416889999999995 < 2.22e-16 \*\*\*  
## step 49.981810000000003 < 2.22e-16 \*\*\*  
## typeTRANSFER 41.024839999999998 < 2.22e-16 \*\*\*  
## amount -31.686820000000001 < 2.22e-16 \*\*\*  
## oldbalanceOrg 59.571449999999999 < 2.22e-16 \*\*\*  
## newbalanceOrig -61.919649999999997 < 2.22e-16 \*\*\*  
## oldbalanceDest 49.869579999999999 < 2.22e-16 \*\*\*  
## newbalanceDest -53.018900000000002 < 2.22e-16 \*\*\*  
## ---  
## AIC: 30617.853925780197  
##   
## Number of Fisher Scoring iterations: 18*

Nous remarquons que toutes les variables sont significatives (trois étoiles dans le modèle \*\*\* avec une très petite p-value).

Nous ne pouvons rien déduire du AIC vu que les modèles sont entrainés sur des différentes datasets.

Pour déterminer le meilleur threshold, nous traçons la courbe ROC (true positives rate and false positives rate in function of threshold)

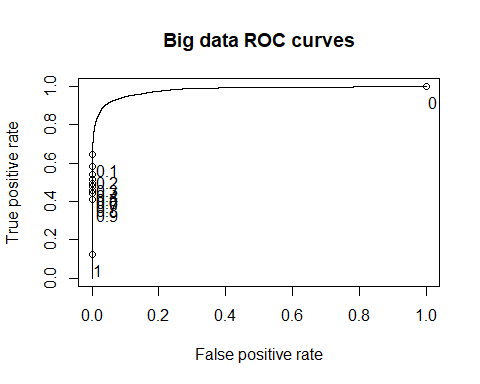


Figure 5.4‑1 : Courbe ROC de la big Data

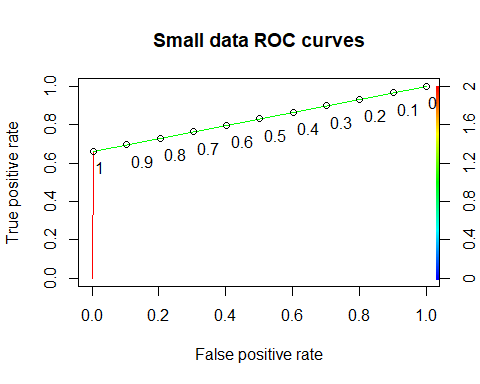


Figure 5.4‑2 : Courbe ROC de la Small Data

Pour maximiser “true positive rate” et minimiser “false positive rate”, nous optons un threshold de 0.7.

Nous expliquons la densité des thresholds dans le premier graph “big data ROC curves”, par le fait que le nombre des non-fraudes (2.77 millions) est très grand par rapport aux fraudes (8213) (unbalancing data issues).

Alors, même si le nombre des faux positifs est grand, son pourcentage reste toujours très faible, autrement dit “false positives rate” reste toujours très petit quel que soit le threshold.

Nous calculons la surface sous la courbe (AREA UNDER CURVE AUR) qui donne une idée à propos de la qualité absolue du modèle (100% pour un modèle parfait qui explique toutes les observations).

*as.numeric(performance(ROCRpred\_big, "auc")@y.values)*

*## [1] 0.98 for big model*

*as.numeric(performance(ROCRpred\_small, "auc")@y.values)*

*## [1] 0.82 for small model*

Le premier modèle explique 98% des observations, et le deuxième explique juste 82.8%.

*threshold*=0.7

*## [1] "big data predictions on big testing set"##   
## FALSE TRUE  
## 0 690492 57  
## 1 1114 935*

*## [1] "the overall accuracy is: 99.83 "*

*## [1] "the overall error is: 0.17 "*

*## [1] "the rate of detected fraud is: 45.63 "*

*## [1] "-----------------------------------------------------"*

*## [1] "big data predictions on small testing set"*

*##   
## FALSE TRUE  
## 0 2512 0  
## 1 1114 935*

*## [1] "the overall accuracy is: 75.58 "*

*## [1] "the overall error is: 24.42 "*

*## [1] "the rate of detected fraud is: 45.63 "*

*## [1] "-----------------------------------------------------"*

*## [1] "small data predictions on big testing set"*

*##   
## FALSE TRUE  
## 0 687348 3201  
## 1 699 1350*

*## [1] "the overall accuracy is: 99.44 "*

*## [1] "the overall error is: 0.56 "*

*## [1] "the rate of detected fraud is: 65.89 "*

*## [1] "-----------------------------------------------------"*

*## [1] "small data predictions on small testing set"*

*##   
## FALSE TRUE  
## 0 2505 7  
## 1 699 1350*

*## [1] "the overall accuracy is: 84.52 "*

*## [1] "the overall error is: 15.48 "*

*## [1] "the rate of detected fraud is: 65.89 "*

**Overall accuracy [= (tpr+fpr) /N]** n’est pas suffisante en tant que critère d’évaluation, vu qu’elle est trop liée à la taille du testing set, ils croissent simultanément. Et de même **overall error** décroît inversement à la taille du testing set. Mais, ces deux critères sont très importants quant aux testings sets qui ont la même taille.

*The rate of detected fraud* est aussi un critère très important mais n’est pas suffisant. Puisqu’il donne une idée à propos du pourcentage des fraudes détectées mais il ne note pas le taux des non-fraudes qui sont détectées comme fraudes. C’est pour cela, il faut toujours comparer tous ces critères pour choisir le meilleur modèle.

A cet égard, nous remarquons que le ***small model*** donne toujours une meilleure accuracy que celle du ***big model*.**

En effet, quand nous entraine notre model sur la grande data, il perd de la crédibilité surtout parce que cette data est déséquilibrée, en revanche la small model est plus crédible et de plus équilibrée.

# Deuxième technique d’équilibrèrent de la data

Nous divisons la data des non-fraudes à plusieurs datasets de 10000 observations concaténées avec 75% des fraudes qui vont être tous des trainings sets.

**Ps.** Les mêmes fraudes se trouvent dans tous ces sous-ensembles.

Et finalement, nous prenons une base de données ou il y a le reste des non-fraudes (un nombre inférieur à 10000) et 25% des fraudes qui ne se trouvent pas dans les trainings sets.

*notFraud\_c*=*df\_isNotFraud  
i*=1 *dt*=*list()  
w****hile*** *(dim(notFraud\_c)[1]>10000){*  ***split***=***s****ample.split(notFraud\_c$step,SplitRatio =* 10000*/nrow(notFraud\_c))   
part* = *subset(notFraud\_c, split==TRUE)  
notFraud\_c* = *subset(notFraud\_c, split==FALSE)  
dt[[i]]*=*rbind(part,df\_isFraud\_train)  
dt[[i]]*=*dt[[i]][order(dt[[i]]$step),]  
 i*=*i+*1 *}  
  
test\_rest*=*rbind(notFraud\_c,df\_isFraud\_test)  
test\_rest*=*test\_rest[order(test\_rest$step),]*

Pour éviter l’exécution de ces split à chaque fois, nous exportons ces datasets dans un dossier extérieur et nous les importons à chaque fois.

Nous construisons 274 modèles logistiques, chacun s’entraine sur une nouvelle data set équilibrée où il y a 10000 observations non-fraudes et 6092 fraudes (75% des fraudes) la prédiction finale sera la moyenne des prédictions des autres modèles sur le même testing set (qui contient tous les non-fraudes et juste 25% des fraudes).

*model\_log*=*list()*for *(i* in1*:*270*){  
model\_log[[i]]*=*glm(isFraud ~ ., data=dt[[i]],family = binomial)}*

*sum\_pred*=0for *(i* in1*:*270*){  
sum\_pred*=*sum\_pred+predict(model\_log[[i]],type="response",newdata = Test\_big)}  
sum\_pred*=*sum\_pred/*270

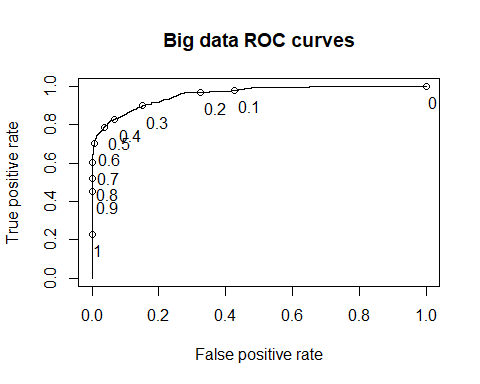


Figure 5.5‑1 : Courbe ROC de la big Data

*a*=*table(Test\_big$isFraud,(sum\_pred)>*0.6*)  
a*

*##   
## FALSE TRUE  
## 0 686157 4392  
## 1 611 1438*

*## [1] "the overall accuracy is: 99.28 "*

*## [1] "the overall error is: 0.72 "*

*## [1] "the rate of detected fraud is: 70.18 "*

*## [1] "the rate of not fraud predicted as fraud is: 0.64 "*

Nous remarquons que les résultats de cette technique sont plus favorables que le modèle logistique avec la small data.

Dans cette technique, chaque modèle donne une probabilité qu’une observation sera fraude, et nous calculons la moyenne de ces probabilités pour la même observation. Le résultat sera notre probabilité finale que l’observation soit une fraude. Et, nous choisissons 0.6 en tant que threshold.

# Arbres de décisions

Nous construisons deux modes d’arbres de décisions, le premier sur small training set et le deuxième sur le big traning set.

*model\_tree\_small* = *rpart(isFraud ~ ., data = Train\_small, method="class",cp=*0.001*)  
prp(model\_tree\_small)*

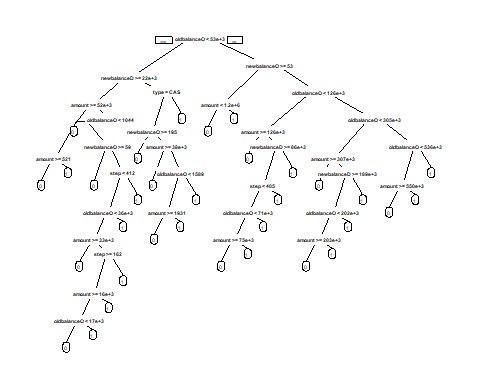


Figure 5.6‑1 : Arbres de décisions de small training Set

*model\_tree\_big* = *rpart(isFraud ~ . , data = Train\_big, method="class",cp=*0.1*)  
prp(model\_tree\_big)*

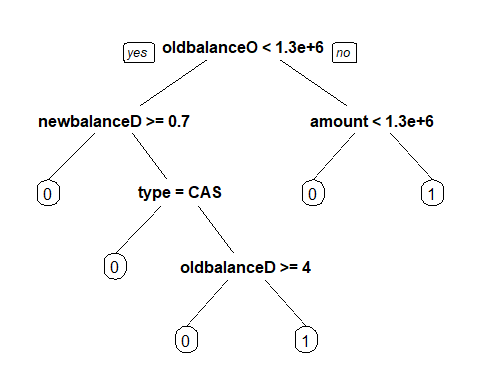


Figure 5.6‑2 : Arbres de décisions de big training Set

Les arbres au-dessus donnent une interprétation logique et développée du problème.

Nous remarquons que celle du « small model » est plus développée que du « big model », même si la deuxième demande plus de temps pour l’exécution. Et, nous expliquons cette remarque par la taille géante du deuxième dataset.

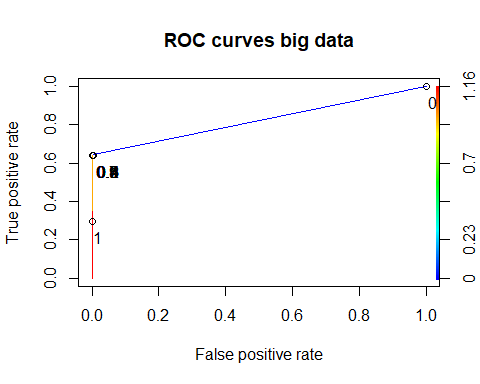


Figure 5.6‑3 : Courbe ROC de la big Data

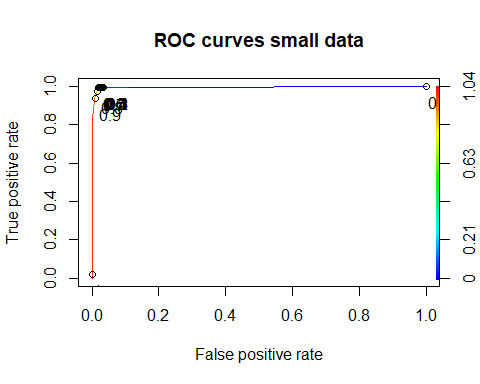


Figure 5.6‑4 : Courbe ROC de la Small Data

Pour maximiser les Tpr et minimiser les fpr, nous choisissons un threshold de 0.5 en prenant compte le ROC curves.

*as.numeric(performance(ROCRpred\_big, "auc")@y.values)*

*## [1] 0.82 for big model*

*as.numeric(performance(ROCRpred\_small, "auc")@y.values)*

*## [1] 0.99 for small model*

Le premier modèle explique 82% des observations, et le deuxième modèle (small data) explique 99 % des observations. Autrement dit, le modèle sur la small data est plus efficace que celui sur la big data.

*threshold\_tree*=0.5

*## [1] "big data predictions on big testing set*

*##   
## FALSE TRUE  
## 0 690479 70  
## 1 753 1296*

*## [1] "the overall accuracy is: 99.88 "*

*## [1] "the overall error is: 0.12 "*

*## [1] "the rate of detected fraud is: 63.25 "*

*## [1] "big data predictions on small testing set"*

*##   
## FALSE TRUE  
## 0 2511 1  
## 1 753 1296*

*## [1] "the overall accuracy is: 83.47 "*

*## [1] "the overall error is: 16.53 "*

*## [1] "the rate of detected fraud is: 63.25 "*

*## [1] "------------------------------------------"*

*## [1] "small data predictions on big testing set"*

*##   
## FALSE TRUE  
## 0 672274 18275  
## 1 15 2034*

*## [1] "the overall accuracy is: 97.36 "*

*## [1] "the overall error is: 2.64 "*

*## [1] "the rate of detected fraud is: 99.27 "*

*## [1] "------------------------------------------"*

*## [1] "small data predictions on small testing set"*

*##   
## FALSE TRUE  
## 0 2443 69  
## 1 15 2034*

*## [1] "the overall accuracy is: 98.16 "*

*## [1] "the overall error is: 1.84 "*

*## [1] "the rate of detected fraud is: 99.27 "*

Nous remarquons que le “small model” donne toujours un bon résultat que celle du “big model”. Nous considérons cette interprétation comme celle de la partie logistique.

En guise de conclusion, les arbres de décisions donnent une accuracy « rate of detected fraud » très importante par rapport à celle de logistique modèle, mais malheureusement on sacrifie « overall accuracy », qui se diminue légèrement, cette diminution coute beaucoup de false positives surtout quand le nombre des non-fraudes est très grand par rapport au fraud ( unblanced data)

Et par la technique de la moyenne des predictions (ou bien voting) nous trouvons :

*model\_tree*=*list()  
sum\_pred*=0for *(i* in1*:*270*){  
model\_tree[[i]]*=*rpart(isFraud ~ ., data = dt[[i]], method="class",cp=*0.001*)  
}*for *(i* in1*:*270*){  
sum\_pred*=*sum\_pred+predict(model\_tree[[i]],type="prob",newdata = Test\_big)[,*2*]}*

*##   
## FALSE TRUE  
## 0 679335 11214  
## 1 18 2031*

*## [1] "the overall accuracy is: 98.38 "*

*## [1] "the overall error is: 1.62 "*

*## [1] "the rate of detected fraud is: 99.12 "*

Cette fois-ci, la méthode du voting donne des résultats plus importants que les précédents avec *rate of detected fraud* de 99.42% et *overall accuracy* de 98.64%.

# Gradient boosting method

Nous optons cette fois pour le « gradient boosting method » algorithme, pour modéliser notre problème.

Nous commençons par la construction de deux modèles l’un sur la « small dataset » et l’autre sur la « big dataset »

*model\_gauss\_small*=*gbm(isFraud~. ,data=Train\_small, distribution="bernoulli",n.trees =*5000*,interaction.depth=*4*,shrinkage=*0.05*)*

*## var rel.inf  
## oldbalanceOrg oldbalanceOrg 60.1014724  
## newbalanceDest newbalanceDest 16.2117811  
## newbalanceOrig newbalanceOrig 10.4502132  
## amount amount 9.0727050  
## step step 2.7417269  
## oldbalanceDest oldbalanceDest 0.7527265  
## type type 0.6693749*

*model\_gauss\_big*=*gbm(isFraud~. ,data=Train\_big, distribution="bernoulli",n.trees =*1000*,interaction.depth=*2*)  
summary(model\_gauss\_big)*

*## var rel.inf  
## type type 68.82085552  
## newbalanceDest newbalanceDest 31.10625021  
## oldbalanceOrg oldbalanceOrg 0.04192312  
## amount amount 0.03097115  
## step step 0.00000000  
## newbalanceOrig newbalanceOrig 0.00000000  
## oldbalanceDest oldbalanceDest 0.00000000*

D’après le rapport, nous remarquons que la variable oldbalanceOrg est la plus significative dans les deux modèles “small model” et “big model”.

Ensuite, nous traçons les « ROC curves » afin qu’on puisse déterminer le meilleur threshold et en même temps visualiser le comportement de notre modèle.

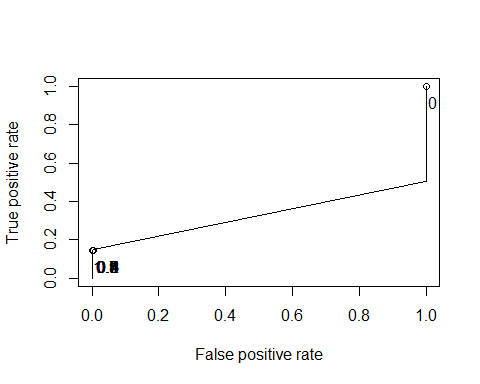


Figure 5.7‑1 : Courbe ROC de la Big Data

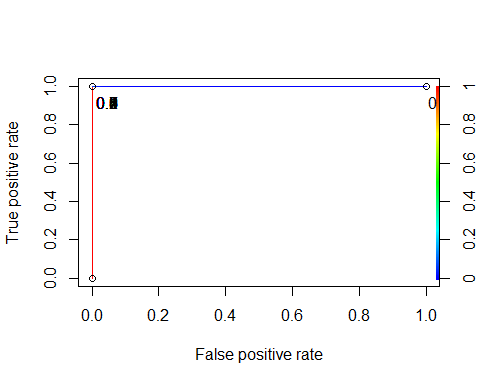


Figure 5.7‑2 : Courbe ROC de la Small Data

D’après, ROC curves nous prenons le threshold = 0.9 (parce que le pourcentage des fraudes est déjà très peu par rapport à celui des non-fraudes, donc nous maximisons le threshold pour éviter de nous pénaliser par « un petit pourcentage supplémentaire des fpr » et qui représente beaucoup de fausses observations). En fait, un petit taux de fpr coûte beaucoup d’observation détectées comme fraudes même s’elles ne sont pas, c’est pour cela nous minimisant le fpr même par un petit taux.

Bien évidemment, le modèle “small\_gauss\_model” est presque parfait puisque son **ROC curves** a un angle de 90°.

*as.numeric(performance(ROCRpred\_big, "auc")@y.values)*

*## [1] 0.32 for big model*

*as.numeric(performance(ROCRpred\_small, "auc")@y.values)*

*## [1] 1 for small model*

Le modèle “big\_gauss model” explique plus de 32% de la data (parce que les nombres des arbres et de learning rate sont inferieur mais avec ce nombre il prend plus de temps pour l’exécution). En revanche, le modèle “small\_gauss\_model” explique 100% de la data

*threshold\_gbm*=0.9

*## [1] "big data predictions on big testing set"*

*##   
## FALSE TRUE  
## 0 690516 33  
## 1 1769 280*

*## [1] "the overall accuracy is: 99.74 "*

*## [1] "the overall error is: 0.26 "*

*## [1] "the rate of detected fraud is: 13.67 "*

*## [1] "----------------------------------------"*

*## [1] "big data predictions on small testing set"##   
## FALSE TRUE  
## 0 2512 0  
## 1 1769 280*

*## [1] "the overall accuracy is: 61.21 "*

*## [1] "the overall error is: 38.79 "*

*## [1] "the rate of detected fraud is: 13.67 "*

*## [1] "----------------------------------------"*

*## [1] "small data predictions on big testing set"*

*##   
## FALSE TRUE  
## 0 686143 4406  
## 1 18 2031*

*## [1] "the overall accuracy is: 99.36 "*

*## [1] "the overall error is: 0.64 "*

*## [1] "the rate of detected fraud is: 99.12 "*

*## [1] "----------------------------------------"*

*## [1] "small data predictions on small testing set"*

*##   
## FALSE TRUE  
## 0 2497 15  
## 1 18 2031*

*## [1] "the overall accuracy is: 99.28 "*

*## [1] "the overall error is: 0.72 "*

*## [1] "the rate of detected fraud is: 99.12 "*

Finalement nous trouvons que **le gradient boosting method** est le plus efficace, puisqu’il peut détecter 100% des fraudes avec une grande *overall accuracy* qui égale à 99.12% (sur le big testing set) sauf avec le **small training set** qui s’exécute dans le minimum du temps par rapport aux autres, et bien évidement sans **voting methode** (275 modèles).

Ces résultats sont grâce à la particularité du **gradient boosting method** qui construit plusieurs arbres dépendantes (au contraire de **random\_forest** qui construit des arbres indépendants entre elles). Et dans notre cas c’est un avantage parce que:

*GBM construit les arbres un par un, où chaque nouvel arbre aide à corriger les erreurs commises par l’arbre préalablement formé.*

Cela permet de trouver les anomalies avec la plus grande précision sans donner trop d’exemples authentiques aux experts.

L’augmentation du gradient s’est avérée être une méthode puissante sur des ensembles de données réels pour résoudre les problèmes d’apprentissage du classement en raison de ses deux caractéristiques principales :

* Il effectue l’optimisation dans l’espace des fonctions (plutôt que dans l’espace des paramètres), ce qui facilite grandement l’utilisation des fonctions de perte personnalisées.
* Le boosting se concentre pas à pas sur des exemples difficiles qui donnent une belle stratégie pour gérer les ensembles de données déséquilibrés en renforçant l’impact de la classe positive.

Le boosting a montré des performances très élevées par rapport aux autres méthodes pour la détection des anomalies surtout dans des bases de données déséquilibrées, ce qui veut dire que ce modèle est le meilleur pour la détection des fraudes même s’il est relativement plus long que les autres en termes du temps d’exécution. Cependant grâce à notre stratégie d’équilibrèrent de la base de données, nous pouvions réduire énormément ce temps d’exécution en gardant les meilleurs résultats.

# Conclusion :

Un modèle bien amélioré pour la détection de fraude s’avère nécessaire de renforcer la protection des systèmes bancaires contre le vol.

En explorant l'ensemble des données, nous avons découvert des modèles qui nous ont permis de construire des caractéristiques importantes et d'écarter celles qui étaient inutiles.

Nous avons appliqué quelques algorithmes d'apprentissage automatique classiques et constaté que les méthodes consistant à générer plusieurs arbres de décision et à regrouper leurs résultats étaient plus performantes.

Entre Logistic Model, Decison Tree et Gradient Boosting Method, des critères pratiques ont été utilisés pour décider que le Gradient Boosting Method était un meilleur modèle.

Certes, Ce modèle est précis et explique parfaitement la majorité de la base de données pour détecter les transactions bancaires frauduleuse, mais le choix de travailler avec l’un et pas l’autre dans des conditions réelles reste toujours dépendant du coût d’un faux négatif par rapport au coût d’un faux positif.

Ces coûts peuvent être évalués en termes de la réputation ou l’image de la banque, des dépenses téléphoniques ou d’énergie pour la vérification des informations, ou encore du temps de travail.

# Annexe

Code R pour l’importation des bibliothèques :

*library(reticulate) # permet d'utiliser Rmarkdown  
library(caTools) #call library for split data  
set.seed(*123*)#random seed to mix data  
library(ModelMetrics)  
options(digits=*20*)  
library(dplyr)   
library(ggplot2)  
library(mltools)  
library(data.table)  
# Load CART packages  
library(rpart)  
library(rpart.plot)  
library(caret)  
#install.packages("e1071")  
library(e1071)  
library(randomForest)  
library(arules)  
library(gbm)  
options(reticulate.repl.quiet =* TRUE*)*

1. [https://forbesafrique.com/pourquoi-les-banques-africaines-sont-elles-victimes-de-piratage/](https://forbesafrique.com/pourquoi-les-banques-africaines-sont-elles-victimes-de-piratage%E2%80%89/)

   <https://www.grin.com/document/489536> [↑](#footnote-ref-1)
2. [*https://ntnuopen.ntnu.no/*](https://ntnuopen.ntnu.no/)*ntnu-xmlui/bitstream/handle/11250* [↑](#footnote-ref-2)
3. *Annexe p.30* [↑](#footnote-ref-3)