27/10/2024

Rapport Projet Scoring M2 MoSEF

LEBRETON Louis – Master 2 MoSEF – Université Paris 1 Panthéon-Sorbonne

Table des matières

I/ Données et objectif

Présentation du dataset

Le dataset contient 5960 lignes et 13 colonnes, dont une variable binaire à prédire appelée BAD. Chaque ligne représente un prêt sur valeur domiciliaire, probablement aux États-Unis, ce qui suggère que le cadre légal américain s'applique, notamment avec des spécificités telles que la possibilité d'emprunter sur des périodes allant jusqu'à 50 ans.

Objectif

L'objectif principal de ce projet est d'évaluer la solvabilité d'un prêt et de déterminer la probabilité que le prêt soit remboursé. Pour cela, différentes caractéristiques du profil de l'emprunteur sont analysées afin de prédire s'il y a un risque de défaut de paiement. Ainsi, le but sera de proposer un modèle expliquant la variable « BAD » par les autres variables disponibles.

Variables disponibles et hypothèses a priori

BAD (Variable à prédire) :

Type : Catégorielle binaire

Modalités :

1 : Le demandeur a fait défaut sur le prêt ou est sérieusement en retard de paiement.

0 : Le demandeur a remboursé le prêt.

Hypothèse a priori : Cette variable est fortement influencée par des facteurs de stabilité financière et des historiques de crédit.

LOAN : Montant du prêt demandé par l'emprunteur.

Type : Numérique

Hypothèse a priori : Non lié à BAD. Il semble peu probable que le montant du prêt demandé influence directement la probabilité de défaut, car cela dépendrait davantage de la capacité de remboursement que du montant en lui-même.

MORTDUE : Montant restant à rembourser sur une hypothèque existante.

Type : Numérique

Hypothèse a priori : Non lié à BAD. Le montant d'une hypothèque existante n'est pas forcément un indicateur direct de la probabilité de défaut.

VALUE : Valeur actuelle de la propriété détenue par l'emprunteur.

Type : Numérique

Hypothèse a priori : Non lié à BAD. La valeur de la propriété n'indique pas nécessairement la capacité à rembourser le prêt.

REASON : Raison de l'emprunt (DebtCon = consolidation de dettes, HomeImp = amélioration de la maison).

Type : Catégorielle

Hypothèse a priori : Potentiellement lié à BAD. Un emprunt pour une consolidation de dettes (DebtCon) peut indiquer des difficultés financières préexistantes, augmentant ainsi le risque de défaut.

JOB : Catégorie professionnelle de l'emprunteur.

Type : Catégorielle

Hypothèse a priori : Lié à BAD. Certaines professions offrent plus de stabilité financière (exemple : les cadres ou employés de bureau) tandis que d'autres (comme les travailleurs indépendants) sont plus sujets aux fluctuations de revenus, augmentant ainsi le risque de défaut.

YOJ : Nombre d'années dans l'emploi actuel.

Type : Numérique

Hypothèse a priori : Lié à BAD. Une plus grande durée dans un emploi pourrait indiquer une meilleure stabilité professionnelle et donc financière, réduisant le risque de défaut.

DEROG : Nombre de rapports de dérogation majeurs (événements négatifs dans l'historique de crédit).

Type : Numérique

Hypothèse a priori : Fortement lié à BAD. Plus il y a de rapports de dérogation, plus l'emprunteur est susceptible d'avoir des difficultés à rembourser.

DELINQ : Nombre de lignes de crédit en retard de paiement.

Type : Numérique

Hypothèse a priori : Fortement lié à BAD. Un nombre élevé de retards de paiement est un indicateur clé de difficultés financières passées et présentes, augmentant ainsi le risque de défaut.

CLAGE : Âge de la plus ancienne ligne de crédit (en mois).

Type : Numérique

Hypothèse a priori : Non lié à BAD. L'âge de la ligne de crédit peut ne pas être directement lié à la probabilité de défaut, bien qu'un historique de crédit long puisse être vu comme un signe de stabilité.

NINQ : Nombre de demandes récentes d'enquête de crédit.

Type : Numérique

Hypothèse a priori : Lié à BAD. Un nombre élevé de demandes récentes pourrait signaler que l'emprunteur est à la recherche de nouvelles sources de crédit, ce qui peut indiquer un risque accru de surendettement.

CLNO : Nombre total de lignes de crédit.

Type : Numérique

Hypothèse a priori : Non lié à BAD. Le nombre total de lignes de crédit n'est pas forcément un indicateur direct de défaut, sauf si combiné avec d'autres facteurs comme le taux d'endettement.

DEBTINC : Ratio dettes/revenus (Debt-to-income ratio). Le ratio dettes/revenus est le pourcentage des revenus mensuels bruts d'un emprunteur consacré au remboursement de ses dettes.

Type : Numérique

Hypothèse a priori : Fortement lié à BAD. Un ratio élevé suggère que l'emprunteur consacre une part importante de ses revenus à rembourser ses dettes, ce qui augmente la probabilité de défaut en cas d'imprévu financier.

II/ Exploratory Data Analysis (EDA)

Analyse univariée

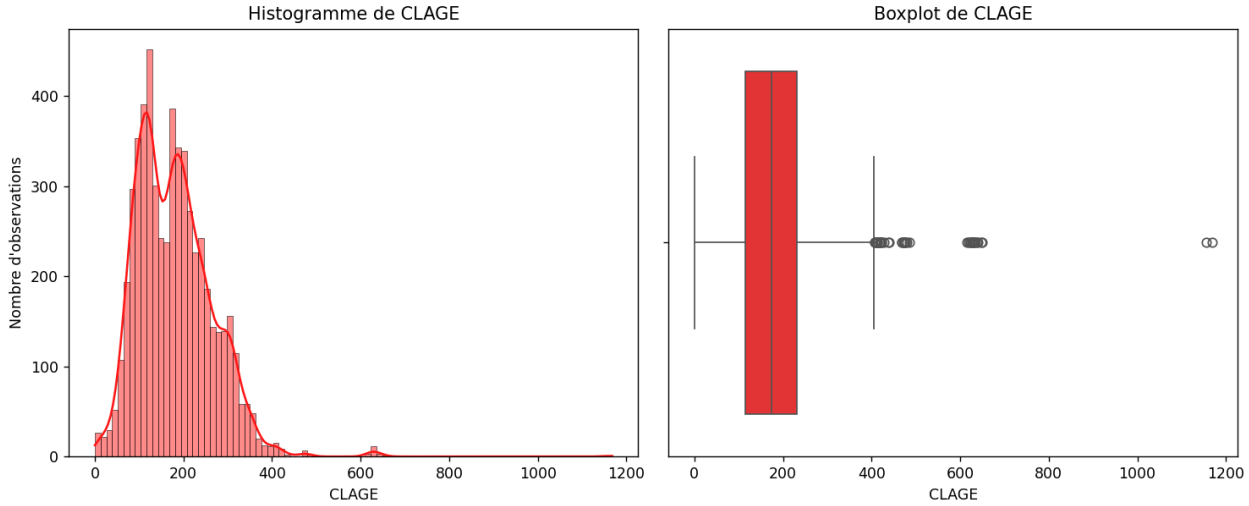
Commençons par observer la distribution univariée des variables du dataset.

Les variables LOAN, MORTDUE et VALUE affichent des valeurs minimales respectives de 1100, 2063 et 8000. Ces valeurs sont cohérentes, car elles sont supérieures à 0, ce qui suggère qu'il n'y a pas de valeurs aberrantes pour ces montants minimaux. Concernant la variable YOJ, sa valeur minimale est 0, ce qui est possible si elle correspond aux clients récemment embauchés ou sans emploi. Il faudra donc vérifier l'impact du fait d'avoir un YOJ de 0 sur BAD dans la suite des analyses.

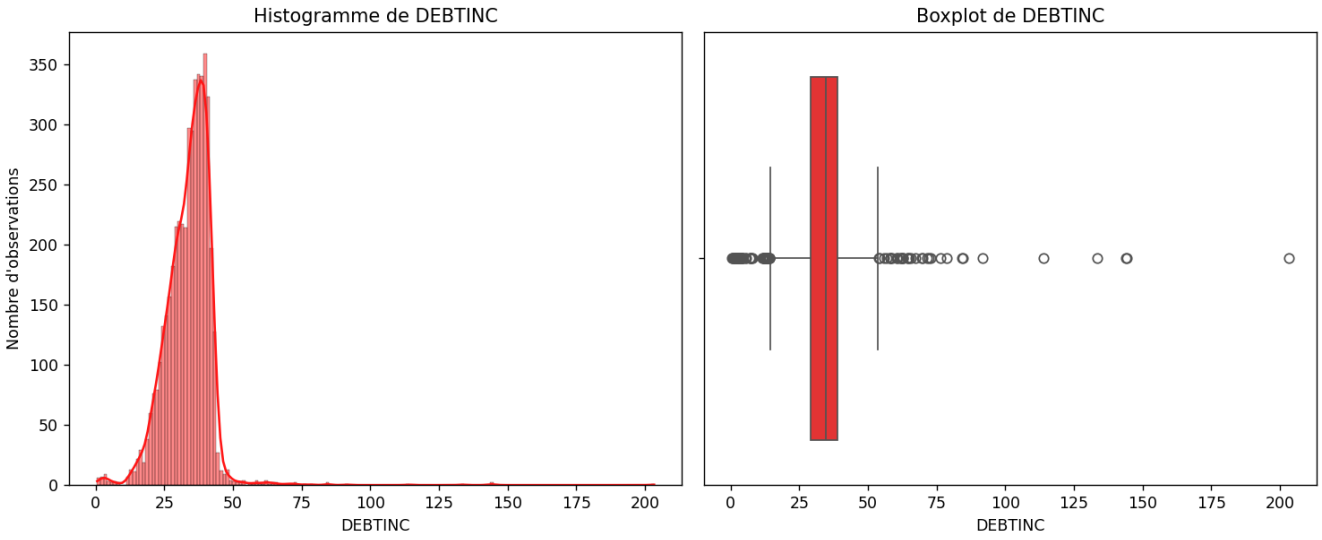
En revanche, certaines valeurs des variables DEBTINC et CLAGE semblent aberrantes.

CLAGE (âge de la plus ancienne ligne de crédit) présente 2 valeurs bien supérieures à 54 ans (soit 650 mois). Ces valeurs semblent incohérentes car elles dépassent la durée maximale légalement possible pour un prêt. En général, les crédits immobiliers et personnels aux États-Unis ne dépassent pas une durée de 30 à 50 ans. Sachant que ces prêts dépassent les 96 ans, on peut les considérer comme des valeurs aberrantes qui seront à traiter par la suite.

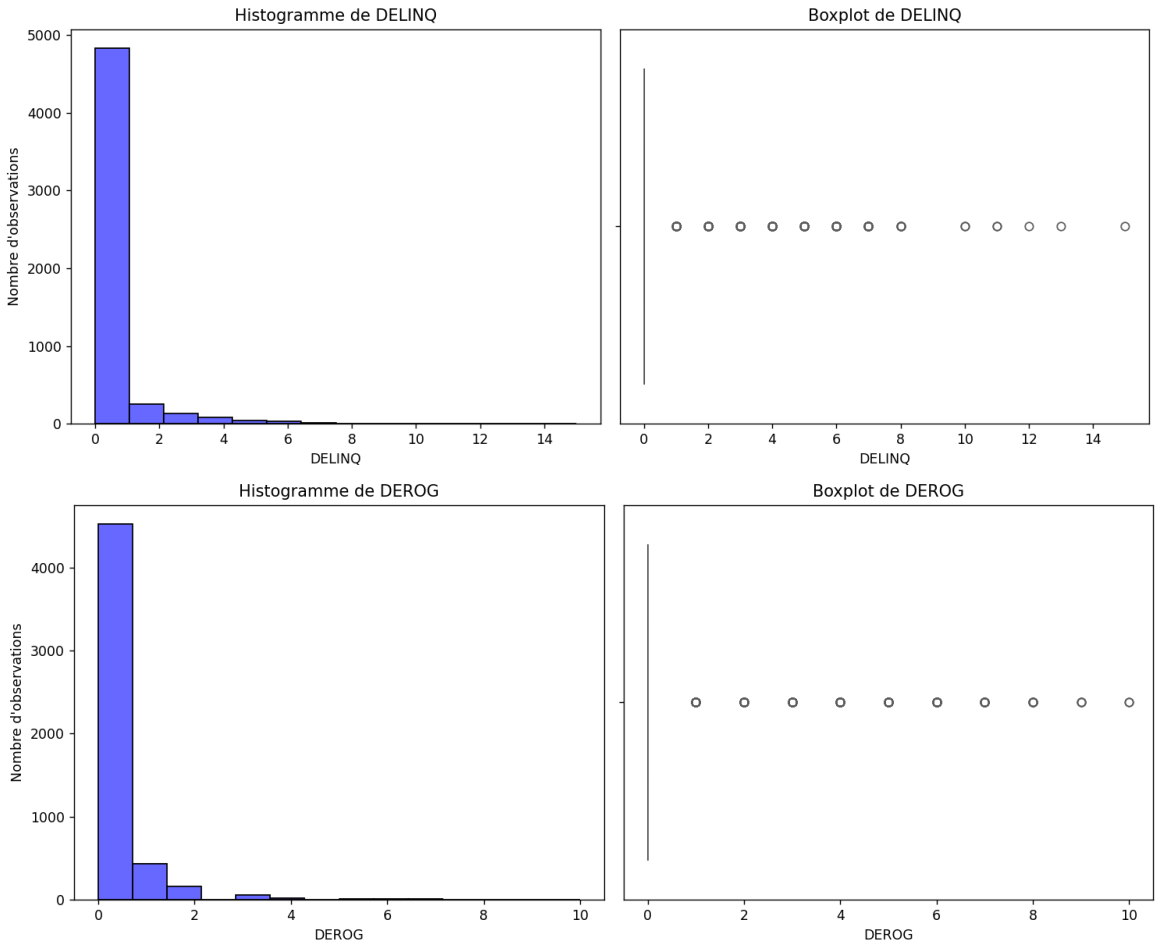
	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC
3097	1	16800	87300.0	155500.0	DebtCon	Other	3.0	0.0	0.0	1154.633333	0.0	0.0	NaN
3679	1	19300	96454.0	157809.0	DebtCon	Other	3.0	0.0	0.0	1168.233561	0.0	0.0	40.206138



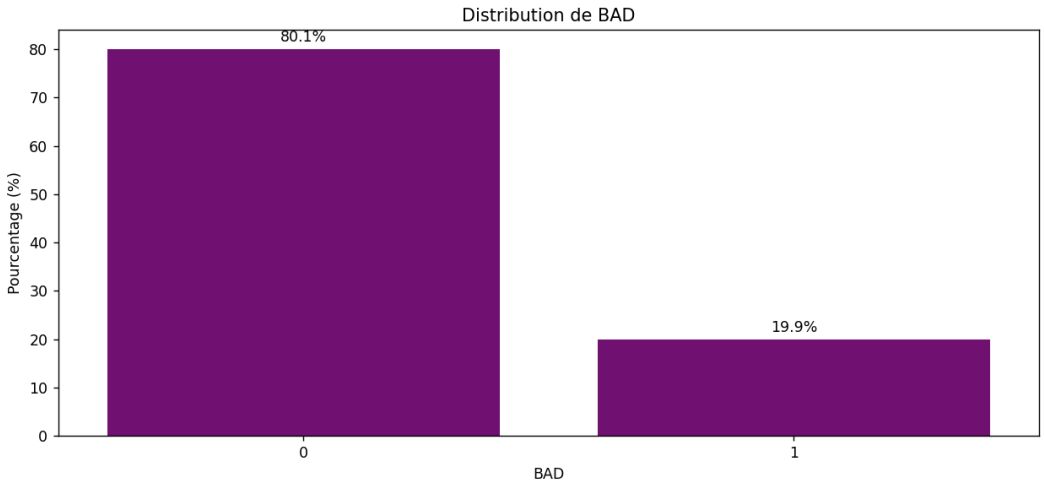
DEBTINC (ratio dettes/revenus) présente également des valeurs élevées, avec 33 valeurs supérieures à 55 % et 5 cas au-delà de 100 %. Ces valeurs sont particulièrement anormales, car un ratio supérieur à 100 % indique que la dette mensuelle d'un client dépasse son revenu mensuel. Ces valeurs extrêmes sont observées exclusivement pour des clients en défaut de paiement ou en retard. Il est possible que ces emprunteurs aient vu leur revenu diminuer entre le moment de l'emprunt et la fin du prêt, ce qui pourrait expliquer cette situation. On peut donc considérer ces variables comme non aberrantes. De façon générale, le ratio DEBTINC des prêts observés se situe entre 10 % et 50 %.



Pour les variables DEROG et DELINQ, qui sont probablement corrélées avec la variable cible BAD, la majorité des clients a un score nul (76 % pour DEROG et 70 % pour DELINQ). Un score supérieur à zéro pour l'une de ces variables pourrait donc indiquer un risque accru de défaut ou de retard de paiement.



Enfin, il est notable que seulement 20 % des clients sont en situation de défaut ou de retard de paiement. Pour modéliser la probabilité de défaut, l'objectif sera de viser une précision supérieure à 80 % afin de surpasser un modèle naïf qui prédirait simplement l'absence de défaut dans la totalité des cas.

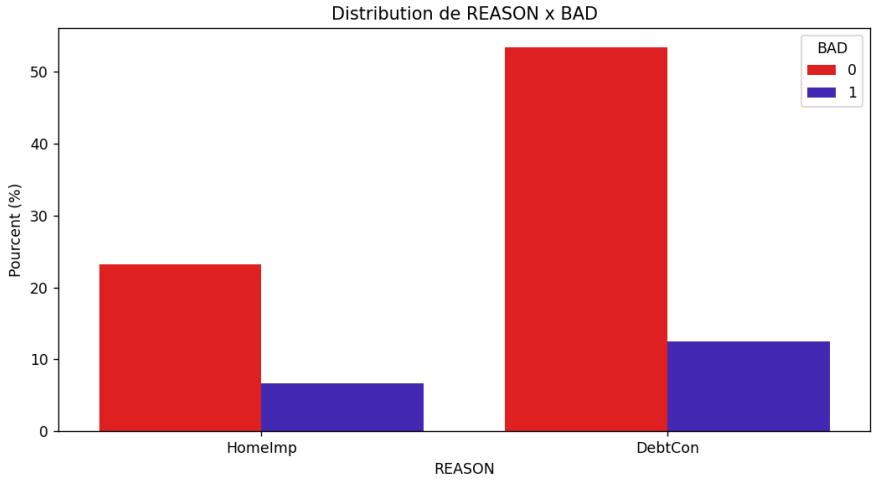


Analyse bivariée

Procédons maintenant à une analyse bivariée en observant les relations entre la variable cible BAD et nos variables explicatives.

Nous commencerons par les deux variables catégorielles REASON et JOB.

En comparant les distributions de BAD en fonction de REASON, il apparaît visuellement que la proportion de défaut (BAD = 1) est similaire quel que soit le motif de l'emprunt.



Les tableaux de fréquence suivants confirment cette observation en affichant les pourcentages de BAD par type de REASON :

Tableau de fréquence pour REASON x BAD sachant REASON

BAD	0	1
DebtCon	81.03%	18.97%
HomeImp	77.75%	22.25%

On observe ici une différence minimale entre les catégories de REASON : le pourcentage de défaut (BAD = 1) est de 19 % pour les emprunts à des fins de consolidation de dettes (DebtCon) et de 22 % pour les prêts d'amélioration de maison (HomeImp). Cela suggère que REASON n'est probablement pas un facteur déterminant de défaut.

Passons à l'analyse de JOB et BAD pour voir si certaines catégories professionnelles sont davantage associées au défaut de paiement.

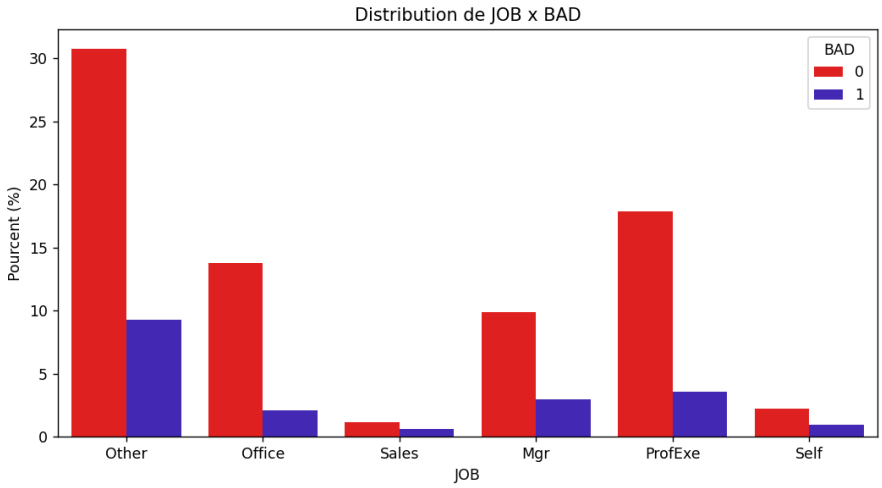


Tableau de fréquence pour JOB x BAD sachant JOB :

JOB	BAD = 0	BAD = 1
Mgr	76.66%	23.34%
Office	86.81%	13.19%
Other	76.80%	23.20%
ProfExe	83.39%	16.61%
Sales	65.14%	34.86%
Self	69.95%	30.05%

Ici, des différences plus notables apparaissent. Par exemple, le taux de défaut est significativement plus élevé pour les emprunteurs ayant des professions dans la vente (Sales) (34 %) et travailleurs indépendants (Self) (30 %), par rapport aux professionnels de bureau (Office) (13 %) et les cadres exécutifs (ProfExe) (17 %). Cela pourrait indiquer que certains métiers, comme Sales et Self, présentent un risque de défaut plus élevé.

Pour confirmer ces observations, nous utilisons le V de Cramer, qui mesure la force de l'association entre les variables catégorielles et BAD. Les résultats sont les suivants :

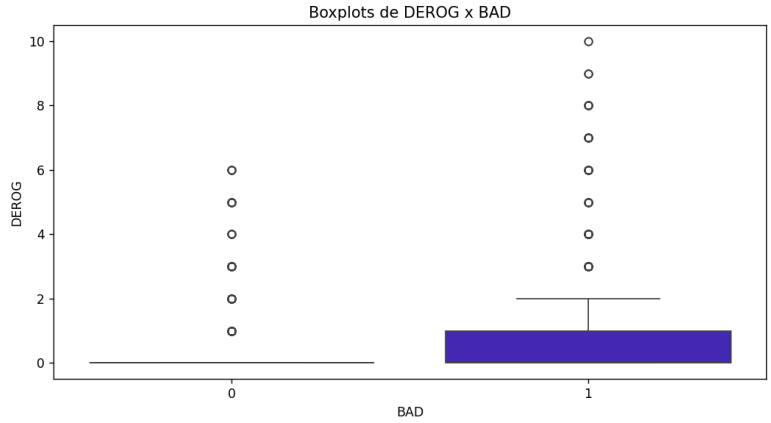
REASON et BAD : V de Cramer = 0.15 donc association faible

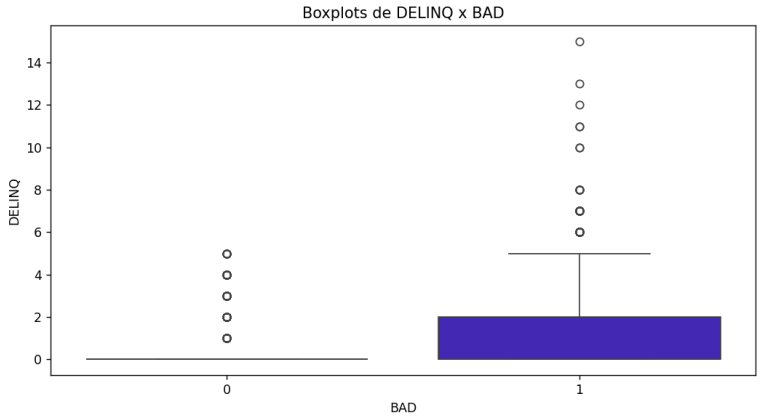
JOB et BAD : V de Cramer = 0.12 donc association faible

Les deux valeurs du V de Cramer sont faibles, indiquant une association limitée entre BAD et les variables REASON et JOB.

Observons maintenant le lien entre BAD et les variables explicatives numériques.

D'après nos hypothèses initiales, les variables DEROG et DELINQ semblent probablement liées à BAD. Cette intuition est confirmée par les boxplots ci-dessous : les valeurs de DEROG ou DELINQ supérieures à 0 sont rares lorsque BAD = 0 mais beaucoup plus fréquentes lorsque BAD = 1.





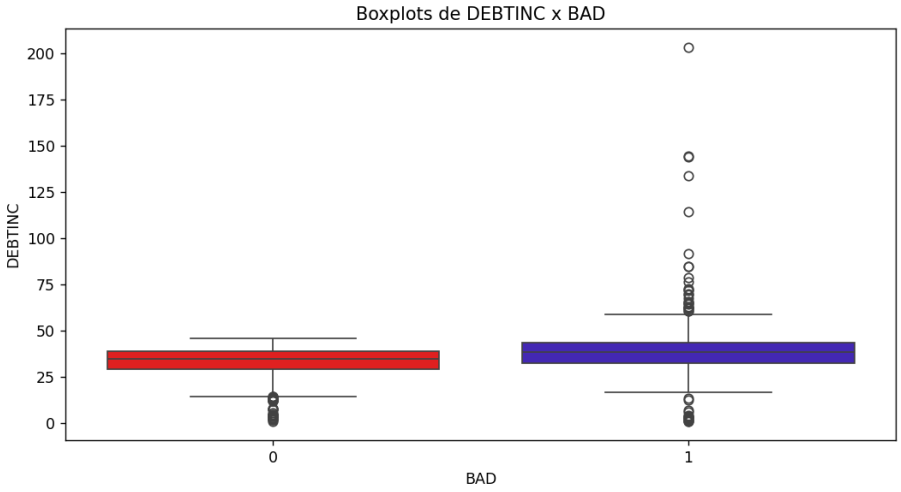
En détail, environ 76 % des clients ont un score de DEROG nul et 70 % ont un score de DELINQ nul, indépendamment de BAD. Cependant, ces proportions augmentent pour les clients qui ne sont pas en défaut de paiement : 79 % des emprunteurs sans défaut (BAD = 0) ont un score nul pour DEROG, soit 3 % de plus, et 75 % ont un score nul pour DELINQ, soit 5 % de plus.

Ces différences confirment que DEROG et DELINQ sont de bons indicateurs du risque de défaut ou de retard de paiement, rendant ces deux variables pertinentes pour prédire la variable cible BAD.

Les valeurs significatives du ratio de corrélation (eta) entre BAD et ces deux variables confirment cette relation :

- Ratio de corrélation entre DEROG et BAD : 0,29
- Ratio de corrélation entre DELINQ et BAD : 0,37

Comme anticipé dans les hypothèses a priori, il existe une corrélation notable entre DEBTINC (ratio dettes/revenus) et la variable cible BAD, avec un ratio de corrélation (eta) de 0,33. Ce coefficient indique une association modérée, confirmant que DEBTINC pourrait être un indicateur pertinent du risque de défaut de paiement. En effet, un ratio dettes/revenus élevé est généralement associé à une pression financière accrue, augmentant la probabilité de défaut.



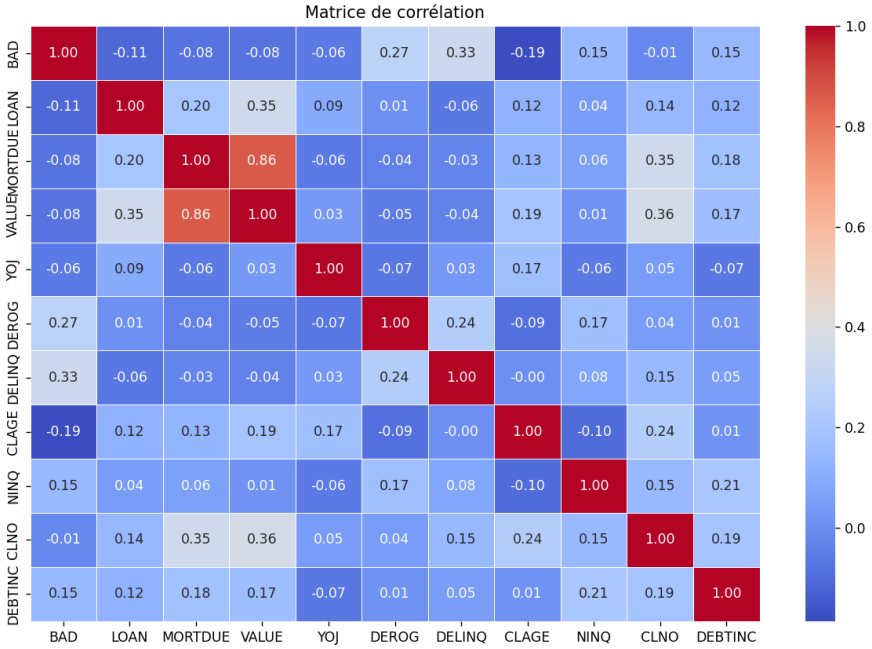
En revanche, aucune corrélation significative n'est observée entre BAD et les autres variables numériques. Voici les ratios de corrélation entre BAD et les autres variables :

LOAN : 0,08, MORTDUE : 0,05, VALUE : 0,03, YOJ : 0,06, CLAGE : 0,18, NINQ : 0,18, CLNO : 0,004

Ces valeurs montrent que les variables LOAN, MORTDUE, VALUE, YOJ, et CLNO n'affichent pratiquement aucune association avec BAD. CLAGE et NINQ présentent une corrélation légèrement plus élevée (environ 0,18), mais restent trop faibles pour être considérées comme de bons prédicteurs du défaut.

Analyse multivariée

En observant la matrice de corrélation de Spearman, on constate que la plupart des variables sont faiblement corrélées entre elles, à l'exception de VALUE et MORTDUE, qui présentent une corrélation élevée de 0,86.



Cette forte corrélation est logique, car VALUE représente la valeur de la propriété, tandis que MORTDUE correspond au montant restant à rembourser sur l'hypothèque. Dans la plupart des cas, une propriété de grande valeur est associée à un montant hypothécaire élevé, ce qui explique cette relation linéaire.

Une telle corrélation peut indiquer une redondance d'information entre ces deux variables. Par conséquent, pour éviter la multicolinéarité dans la modélisation, il pourrait être judicieux d'envisager l'utilisation d'une seule de ces variables.

III/ Prétraitement des données

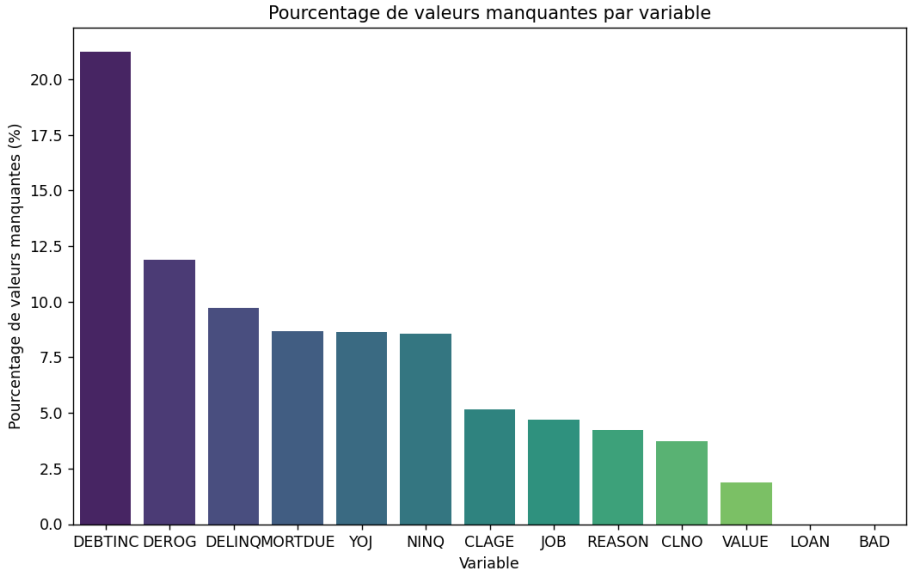
Traitement des valeurs manquantes et aberrantes

Traitement des valeurs aberrantes par plafonnement

Pour la variable CLAGE (âge de la plus ancienne ligne de crédit), deux valeurs sont extrêmement élevées, dépassant 96 ans, ce qui dépasse largement les durées habituelles de crédit. Nous avons donc plafonné CLAGE à 660 mois.

Traitement des valeurs manquantes :

Passons maintenant à l'examen des valeurs manquantes pour nos variables.



Pourcentage de valeurs manquantes par variable (en %) :

Variable	% de valeurs manquantes
DEBTINC	21.0
DEROG	12.0
DELINQ	10.0
MORTDUE	9.0
YOJ	9.0

NINQ	9.0
CLAGE	5.0
JOB	5.0
REASON	4.0
CLNO	4.0
VALUE	2.0
LOAN	0.0
BAD	0.0

La variable cible BAD n'a pas de valeurs manquantes, tandis que DEBTINC présente le pourcentage le plus élevé de valeurs manquantes (21 %). Les autres variables ont entre 2 % et 12 % de valeurs manquantes.

Tout d'abord, nous choisissons de supprimer 4,61 % de lignes (275 lignes sur 5960) ayant cinq valeurs manquantes ou plus, considérant qu'elles sont trop incomplètes pour être fiables.

Pour la variable REASON, nous créons une catégorie « Other » pour les valeurs manquantes, afin d'inclure une possibilité de raison de prêt différente des options initiales. Pour la variable JOB, les valeurs manquantes sont également imputées par « Other », une modalité déjà existante.

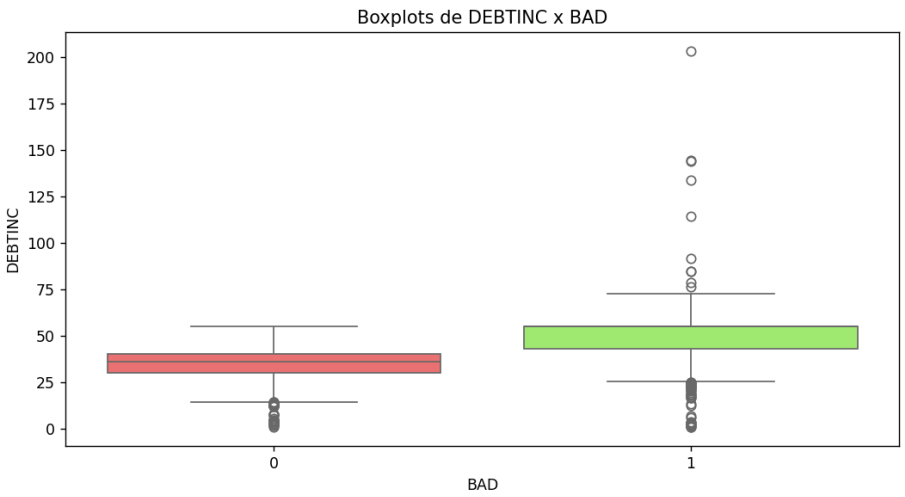
Pour les variables numériques sans forte corrélation avec BAD (telles que MORTDUE, YOJ, NINQ, CLAGE, CLNO, et VALUE), nous utilisons la médiane pour l'imputation, car elle est moins sensible aux valeurs extrêmes.

DEROG et DELINQ, étant modérément corrélées avec BAD, nécessitent un traitement plus attentionné. Pour ces variables, la valeur la plus fréquente (0) est utilisée pour l'imputation, car elle correspond à 76 % des valeurs pour DEROG et 70 % pour DELINQ. L'analyse montre que les individus ayant une valeur manquante pour DEROG ou DELINQ ont un taux de défaut similaire ou inférieur à ceux ayant un score de 0, ce qui justifie l'imputation par 0 sans introduire de biais significatif.

DEBTINC est particulièrement importante, car elle est fortement corrélée avec BAD. Cependant, avec 21 % de valeurs manquantes, l'imputation classique (par médiane ou moyenne) ou par le biais d'un modèle de Machine Learning pourrait réduire la pertinence de cette variable.

En examinant les données, nous avons constaté que les emprunteurs pour lesquels DEBTINC est manquant présentent un taux de défaut de 61 %, bien plus élevé que le taux global de défaut de 20 %. Cette information est précieuse et indique que les valeurs manquantes de DEBTINC pourraient elles-mêmes signaler un risque de défaut élevé. En imputant DEBTINC par le plafond de 55 % (représentant le ratio dettes/revenus maximum accepté dans des cas particuliers aux États-Unis), nous préservons cette relation sans introduire de biais défavorable.

Après imputation, la corrélation entre DEBTINC et BAD est passée de 0,33 à 0,49, renforçant l'association entre ces deux variables et améliorant ainsi leur contribution potentielle dans la prédiction du défaut de paiement. On peut observer ce lien par biais du boxplot post-imputation suivant :



Feature Engineering

Pour améliorer les performances de notre modèle, nous avons créé trois nouvelles variables basées sur des ratios et des calculs entre les variables existantes, qui pourraient être pertinentes pour mieux capturer la relation entre les emprunteurs et leur capacité de remboursement :

LOAN_VALUE_RATIO : Ratio Loan / Value. Il pourrait fournir des informations sur le niveau de risque de l'emprunt par rapport à la garantie immobilière, puisque des ratios plus élevés indiquent un emprunt plus important par rapport à l'actif.

MORTDUE_VALUE_RATIO : Ratio Mortdue / Value. Ce ratio entre le montant restant dû sur l'hypothèque et la valeur de la propriété peut être un indicateur de l'équilibre financier de l'emprunteur.

CREDIT_PER_MONTH : Ratio CLNO / CLAGE donne une idée de la fréquence d'acquisition de crédit. Une valeur plus élevée pourrait indiquer une tendance accrue à contracter des crédits, potentiellement liée à des comportements financiers à risque.

Bien que ces nouvelles variables semblent intuitivement pertinentes, leurs corrélations avec BAD se révèlent faibles (corrélations respectives de 0,03 pour LOAN_VALUE_RATIO et MORTDUE_VALUE_RATIO et de 0,06 pour CREDIT_PER_MONTH), indiquant qu'elles n'apportent pas de lien direct fort avec la variable cible.

En parallèle, nous avons également créé de nouvelles variables issues d'interactions entre les variables présentant des corrélations plus élevées avec BAD : DEBTINC, DEROG, et DELINQ.

Après cette phase, nous disposons de 21 variables explicatives, contre 12 initialement.

Encodage et Normalisation

Pour préparer les variables catégorielles pour la modélisation, nous appliquons un One-Hot Encoding, car toutes les variables catégorielles sont nominales (pas de hiérarchie entre les catégories). Nous supprimons une modalité par variable pour éviter les problèmes de multicolinéarité. Ensuite, nous appliquons une normalisation Z-score sur les variables numériques.

IV/ Modélisation

Les étapes de notre processus de modélisation sont la sélection des variables, les tests statistiques, le choix du modèle puis optimisation des hyperparamètres et enfin l'interprétation des résultats.

Sélection des variables

Pour sélectionner les variables explicatives les plus pertinentes, nous avons appliqué une méthode de sélection progressive appelée stepwise forward. Cette approche sélectionne les variables de façon itérative, en ajoutant ou en retirant des variables en fonction de leur significativité statistique, évaluée par leur p-valeur dans un modèle logistique.

Le processus se poursuit jusqu'à ce qu'aucune autre variable ne remplisse les critères d'ajout ou de retrait, garantissant ainsi un modèle avec les variables les plus pertinentes.

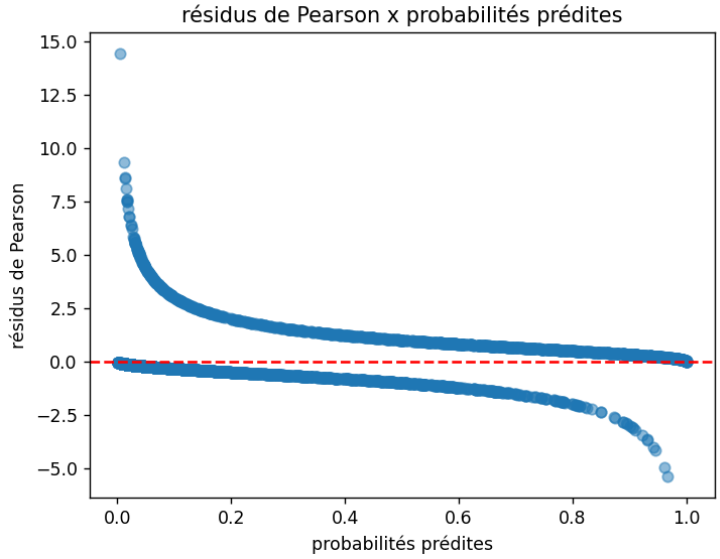
Résultat de la sélection de variables

La sélection progressive a conservé 11 variables spécifiques :

- DEBTINC et DEBTINC^2 : La variable du ratio dettes/revenus (DEBTINC) ainsi que son carré ont été retenus, confirmant son importance comme indicateur de risque.
- DEROG et DELINQ : Ces variables, corrélées à BAD, sont également retenues en raison de leur potentiel prédictif.
- LOAN_VALUE_RATIO : Ce ratio, créé lors du feature engineering, est retenu au lieu des variables brutes LOAN et VALUE, ce qui indique qu'il représente mieux la relation entre le montant emprunté et la garantie.
- REASON : La variable catégorielle REASON a été convertie en une variable binaire indiquant si le prêt est destiné à l'amélioration de la maison (REASON_HomeImp = 1) ou non (REASON_HomeImp = 0).
- CLAGE, YOJ, NINQ, CLNO, JOB

Tests statistiques

La variance des résidus est stable, excepté pour les probabilités très faibles et très élevées prédites par le modèle logistique.



La statistique de Durbin-Watson est de 1.68, proche de 2, ce qui suggère l'absence d'autocorrélation significative entre les résidus.

En examinant le facteur d'inflation de la variance (VIF) pour chaque variable, on n'observe pas de colinéarité majeure, à l'exception de DEBTINC et DEBTINC^2, avec des VIF de 6.53 et 6.35 respectivement. Pour résoudre ce problème de multicolinéarité, DEBTINC^2 est retirée du modèle logistique afin d'améliorer la robustesse des estimations du modèle logistique.

variable	VIF
const	1.68
DELINQ	1.09
CLAGE	1.15
DEBTINC	1.11
DEROG	1.08
JOB_Office	1.01
NINQ	1.09
JOB_Sales	1.02
CLNO	1.16
LOAN_VALUE_RATIO	1.07
YOJ	1.06
REASON_HomeImp	1.06

variable	VIF
const	1.68
DELINQ	1.09
CLAGE	1.15
DEBTINC	6.53
DEROG	1.08
JOB_Office	1.01
NINQ	1.09
JOB_Sales	1.02
CLNO	1.16
LOAN_VALUE_RATIO	1.07
YOJ	1.06
REASON_HomeImp	1.06
DEBTINC**2	6.35

Choix du modèle

Plusieurs modèles ont été testés. Le Random Forest s'avère être celui qui offre les meilleures performances sur l'échantillon de test, selon les métriques d'évaluation (Accuracy, Precision, Recall, et F1 Score).

	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.883905	0.784314	0.547945	0.645161
Decision Tree	0.861038	0.639269	0.639269	0.639269
Random Forest	0.912049	0.814815	0.703196	0.754902
Gradient Boosting	0.901495	0.805714	0.643836	0.715736

Modèles

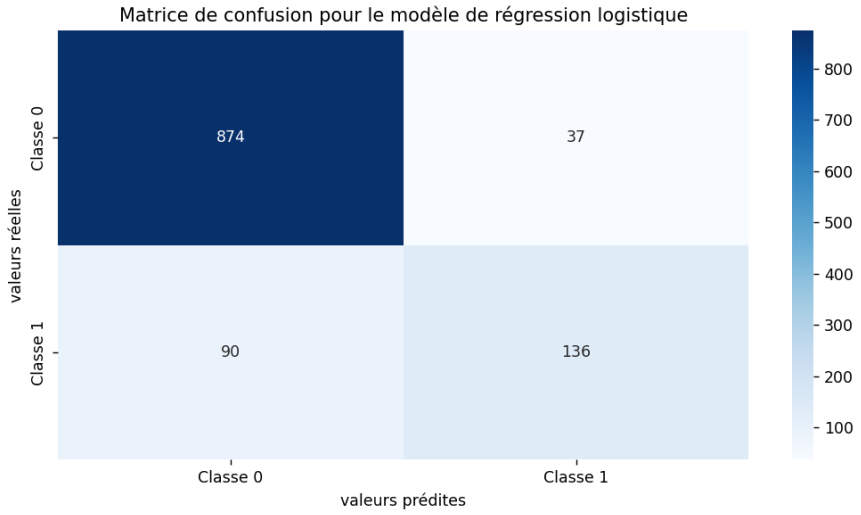
Modèle logistique :

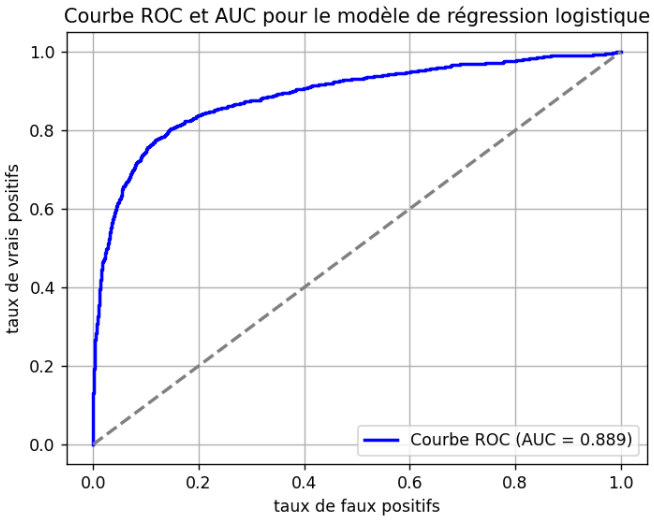
	Odds Ratio	p-value
const	0.12	0.000
DELINQ	2.27	0.000
CLAGE	0.61	0.000
DEBTINC	4.33	0.000
DEROG	1.61	0.000
JOB_Office	0.48	0.000
NINQ	1.20	0.000
JOB_Sales	2.41	0.001
CLNO	0.82	0.000
LOAN_VALUE_RATIO	0.92	0.056
YOJ	0.91	0.046
REASON_HomeImp	1.21	0.054

Les résultats du modèle logistique montrent que toutes les variables incluses sont significatives (p-value < 0.05) car sélectionnées préalablement via une méthode stepwise.

Les résultats ne sont pas directement interprétables en raison de la normalisation des données, mais ils permettent d'observer l'impact de chaque variable sur la probabilité de défaut. Par exemple, une augmentation du ratio dette/revenu (DEBTINC), ainsi qu'une hausse du nombre de lignes de crédit en retard de paiement (DELINQ) ou le nombre de rapports de dérogation (DEROG), augmentent de manière significative la probabilité de défaut. De plus, le fait de travailler dans le secteur des ventes est associé à une probabilité plus élevée de défaut.

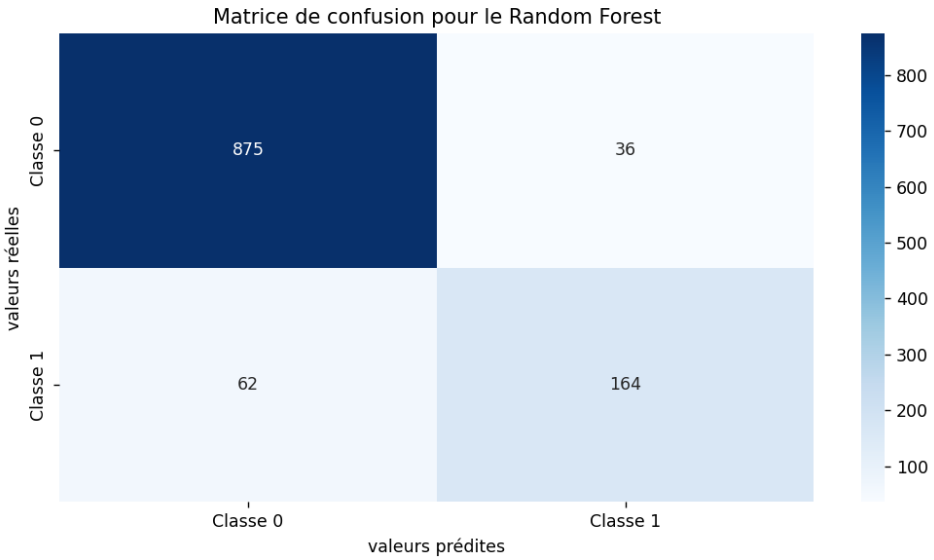
En revanche, certaines variables ont un effet protecteur. Le fait de travailler dans un emploi de bureau (JOBOFFICE) réduit la probabilité de défaut, tout comme l'augmentation de l'âge de la plus ancienne ligne de crédit (CLAGE), qui semble également diminuer ce risque.





Les performances du modèle logistique sont les suivantes : Accuracy : 0.89, Precision : 0.85, Recall : 0.78, F1 Score : 0.81, AUC : 0.89.

Modèle Random Forest (optimisé avec gridsearch et validation croisée) :



Les résultats pour le Random Forest sont : Accuracy : 0.91, Precision : 0.88, Recall : 0.84, F1 Score : 0.86. On obtient donc de meilleurs performances avec ce modèle optimisé que précédemment.

Enfin, aucun point influent n'a été détecté, la distance de Cook maximum observée étant de 0.011 (aucune valeur supérieure à 1).

V/ Segmentation

Nous avons segmenté les clients en classes de risque en appliquant un K-Means sur les probabilités de défaut prédites . En utilisant la méthode Elbow, nous avons déterminé que trois classes étaient optimales.

Les centroïdes de ces classes indiquent trois niveaux de risque distincts, que nous avons nommés :

AAA : faible risque (4473 prêts)

BBB : risque modéré (231 prêts)

CCC : risque élevé (981 prêts)

Moyenne de probabilité de défaut préidte par classe pour la variable BAD :

AAA : 0.0058 (très faible probabilité de défaut)

BBB : 0.6234 (risque modéré)

CCC : 0.9796 (risque très élevé)

VI/ Annexes (scripts)

Le script main.py utilisent des fonctions issues des scripts functions_EDA.py, functions_pretraitement.py, functions_tests_statistiques.py, functions_modelisation.py.

main.py

```
"""
Projet Scoring
M2 MoSEF
Louis LEBRETON
Dataset: hmeq.csv

main.py: Script principal
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import functions_EDA as f_eda
import functions_pretraitement as f_pre
import functions_modelisation as f_m
import functions_tests_statistiques as f_ts


# EDA
#############################################################################################
#########

df = pd.read_csv("data/hmeq.csv")

df.head()
df.shape
df.info()
df.describe()

vars_cat = df.select_dtypes(include=['object', 'category']).columns
print(vars_cat) #vars cat sans 'BAD'

for var in vars_cat:
    print(df[var].nunique(),' modalités issues de la variable', str(var) + " :
", df[var].unique())

vars_num = df.select_dtypes(include=['float', 'int']).columns
vars_num = list(vars_num)
vars_num.remove('BAD')
print(vars_num)
```

```
# Univarie

f_eda.afficher_df_head(df)
f_eda.afficher_cat_vars_univarie_graph(df, 'BAD', palette=["purple"])
f_eda.afficher_cat_vars_univarie_tableau(df, 'BAD')
f_eda.afficher_num_vars_univarie_graph(df, palette=["#3336ff"])
df.columns

# Observations de valeurs extrêmes
df[df['CLAGE']>650]
len(df[df['DEBTINC']>55])
len(df[df['DEBTINC']>100])

# Bivarie
ma_palette = ["#ff0000", "#3511ca"]

f_eda.afficher_cat_vars_bivarie_graph(df, 'BAD', palette = ma_palette)
f_eda.afficher_num_vars_bivarie_graph(df, 'BAD', palette = ma_palette)
f_eda.afficher_cat_vars_bivarie_tableau(df, 'BAD')

# Observations de DEROG X BAD et de DELINQ X BAD
len(df[df['DEROG'] == 0]) / len(df)
len(df[df['DELINQ'] == 0]) / len(df)
df_bad_0 = df[df['BAD'] == 0]

len(df[df['YOJ']==0.0]) / len(df)
len(df_bad_0[df_bad_0['YOJ']==0.0]) / len(df_bad_0)

len(df_bad_0[df_bad_0['DEROG'] == 0]) / len(df_bad_0)
len(df_bad_0[df_bad_0['DELINQ'] == 0]) / len(df_bad_0)

# correlation entre variables categorielles
f_eda.v_cramer_test_khi2(df['REASON'], df['JOB']) # association faible mais
stats significative
f_eda.v_cramer_test_khi2(df['BAD'], df['JOB']) # association faible mais stats
significative


# correlation entre 1 var cat x 1 var num
for var in vars_num:
    print('correlation_ratio entre ',var ,' et BAD: ',
f_eda.correlation_ratio(df['BAD'], df[var]))


# Multivarie

f_eda.afficher_matrice_correlation_num(df, methode='spearman')
f_eda.afficher_matrice_correlation_num(df, methode='kendall')
f_eda.afficher_matrice_correlation_num(df)
```

```
# Derog et Delag corrélés à la variable y
# Value et Mortdue fortement et postiviement corrélés
sns.regplot(x=df['VALUE'], y=df['MORTDUE'], data=df, color = '#27006c')
plt.xlabel('VALUE')
plt.ylabel('MORTDUE')
plt.title('VALUE x MORTDUE')
plt.show()

# Prétraitement des données
################################################################################
#########

# Traitement des valeurs aberrantes par plafonnement

df.loc[df['CLAGE'] > 660, 'CLAGE'] = 660 # 55 ans
# df.loc[df['DEBTINC'] > 55, 'DEBTINC'] = 55


# Traitement des valeurs manquantes
f_pre.afficher_pourcentage_valeurs_manquantes(df)
f_pre.afficher_valeurs_manquantes_barplot(df)

# Supression des lignes avec un nb de valeurs manquantes > 5
tab_nb_va_manquantes = f_pre.afficher_nb_valeurs_manquantes(df)
# lignes avec au moins nb_vm valeurs manquante
for nb_vm in range(len(df.columns)):
    pourcent_vm = len(tab_nb_va_manquantes[tab_nb_va_manquantes > nb_vm]) /
len(df)
    print('Plus de ',nb_vm, 'valeurs manquantes pour ' ,round(100 *
pourcent_vm,2)," % des lignes")

# Observons ces lignes avec de nombreuses valeurs manquantes
index_a_enlever = tab_nb_va_manquantes[tab_nb_va_manquantes > 4].index
df_vm_4 = df.loc[index_a_enlever]
f_eda.afficher_cat_vars_univarie_graph(df_vm_4, 'BAD', palette=["blue"])
f_eda.afficher_num_vars_univarie_graph(df_vm_4, palette=["#3336ff"])


df = df.drop(index=index_a_enlever)
df = df.reset_index(drop=True)

# Imputation
df["REASON"].fillna("Other", inplace=True)
df["JOB"].fillna('Other',inplace=True)

# imputation par la mediane
for var in ["MORTDUE", "YOJ", "NINQ", "CLAGE", "CLNO", "VALUE"]:
    df[var] = df[var].fillna(value=df[var].median())
```

```
# imputation par 0 pour DEROG et DELINQ
# Observation préliminaire : DEROG NaN vs O
df_DEROG_0 = df[df['DEROG'] == 0]
df_DEROG_na = df[df['DEROG'].isna()]

len(df_DEROG_0[df_DEROG_0['BAD'] == 1]) / len(df_DEROG_0)
len(df_DEROG_na[df_DEROG_na['BAD'] == 1]) / len(df_DEROG_na)

# Observation préliminaire : DELINQ NaN vs O
df_DELINQ_0 = df[df['DELINQ'] == 0]
df_DELINQ_na = df[df['DELINQ'].isna()]
len(df_DELINQ_0[df_DELINQ_0['BAD'] == 1]) / len(df_DELINQ_0)
len(df_DELINQ_na[df_DELINQ_na['BAD'] == 1]) / len(df_DELINQ_na)

for var in ["DEROG", "DELINQ"]:
    df[var] = df[var].fillna(value=0)

# imputation de DEBTINC

# test de plusieurs méthodes
# observation des valeurs manquantes pour cette variables
df_DEBTINC_na = df[df['DEBTINC'].isna()]
len(df_DEBTINC_na[df_DEBTINC_na['BAD'] == 1]) / len(df_DEBTINC_na)
len(df[df['BAD'] == 1]) / len(df)

# test d'imputation sur DEBTINC
df_impute = f_pre.random_forest_imputation(df, 'DEBTINC')
print('imputation random forest : ')
print('df précédent -> correlation_ratio entre DEBTINC et BAD: ',
f_eda.correlation_ratio(df['BAD'], df['DEBTINC']))
print('df imputé -> correlation_ratio entre DEBTINC et BAD: ',
f_eda.correlation_ratio(df_impute['BAD'], df_impute['DEBTINC']))
print('-'*100)
df_impute = df.copy()
df_impute['DEBTINC'] =
df_impute['DEBTINC'].fillna(value=df_impute['DEBTINC'].median())
print('imputation mediane : ')
print('df précédent -> correlation_ratio entre DEBTINC et BAD: ',
f_eda.correlation_ratio(df['BAD'], df['DEBTINC']))
print('df imputé -> correlation_ratio entre DEBTINC et BAD: ',
f_eda.correlation_ratio(df_impute['BAD'], df_impute['DEBTINC']))
print('-'*100)
df_impute = df.copy()
df_impute['DEBTINC'] =
df_impute['DEBTINC'].fillna(value=df_impute['DEBTINC'].mean())
print('imputation moyenne : ')
print('df précédent -> correlation_ratio entre DEBTINC et BAD: ',
f_eda.correlation_ratio(df['BAD'], df['DEBTINC']))
```

```
print('df imputé -> correlation_ratio entre DEBTINC et BAD: ',
f_eda.correlation_ratio(df_impute['BAD'], df_impute['DEBTINC']))
print('-'*100)

df_impute = df.copy()
df_impute['DEBTINC'] = df_impute['DEBTINC'].fillna(value=0)
print('imputation par 0 : ')
print('df précédent -> correlation_ratio entre DEBTINC et BAD: ',
f_eda.correlation_ratio(df['BAD'], df['DEBTINC']))
print('df imputé -> correlation_ratio entre DEBTINC et BAD: ',
f_eda.correlation_ratio(df_impute['BAD'], df_impute['DEBTINC']))
print('-'*100)

df_impute = df.copy()
df_impute['DEBTINC'] = df_impute['DEBTINC'].fillna(value=55)
print('imputation par 55 : ')
print('df précédent -> correlation_ratio entre DEBTINC et BAD: ',
f_eda.correlation_ratio(df['BAD'], df['DEBTINC']))
print('df imputé -> correlation_ratio entre DEBTINC et BAD: ',
f_eda.correlation_ratio(df_impute['BAD'], df_impute['DEBTINC']))
print('-'*100)

ma_palette = ['#fd5d5d','#9afd5d']
f_eda.afficher_num_vars_bivarie_graph(df_impute, 'BAD', palette = ma_palette)


f_pre.afficher_pourcentage_valeurs_manquantes(df_impute) # verification



# Feature Engineering

# nouvelles variables
df_impute['LOAN_VALUE_RATIO'] = df_impute['LOAN'] / df_impute['VALUE']
df_impute['MORTDUE_VALUE_RATIO'] = df_impute['MORTDUE'] / df_impute['VALUE']
df_impute['CREDIT_PER_MONTH'] = df_impute['CLNO'] / df_impute['CLAGE']

# gestion des ['CLAGE'] == 0
mask_inf = np.isinf(df_impute['CREDIT_PER_MONTH'])
df_impute.loc[mask_inf, 'CREDIT_PER_MONTH'] = df_impute['CLNO']

for var in ['LOAN_VALUE_RATIO', 'MORTDUE_VALUE_RATIO', 'CREDIT_PER_MONTH']:
    print('correlation_ratio entre ',var ,' et BAD: ',
f_eda.correlation_ratio(df_impute['BAD'], df_impute[var]))


colonnes_pour_interaction = ["DEROG", "DELINQ","DEBTINC"]
df_modelisation = f_pre.creation_variables_interaction(df_impute, cols =
colonnes_pour_interaction)
```

```
df_modelisation.info()
len(df.columns)
len(df_modelisation.columns)

# Encodage
df_encoded = pd.get_dummies(df_modelisation, drop_first=True)
df_encoded.columns

# Normalisation / Standardisation

df_normalise = f_pre.normaliser_variables(df_encoded, 'BAD', scaler =
"standard")
f_eda.afficher_df_head(df_normalise)

# transformation des variables booléennes en 0 et 1
df_clean = df_normalise.applymap(lambda x: int(x) if isinstance(x, bool) else
x)
f_eda.afficher_df_head(df_clean)
df_clean.info()

# Modélisation
################################################################################
#########

# Choix des variables
vars_selectionne = f_m.stepwise_selection(df_clean, 'BAD')
vars_selectionne

# Test statistiques
f_ts.test_homoscedasticite(df_clean[vars_selectionne], df_clean['BAD'])
dw_stat = f_ts.test_independance_erreurs(df_clean[vars_selectionne],
df_clean['BAD'])
vif_df= f_ts.test_independance_variables(df_clean[vars_selectionne])
vif_df

vars_selectionne.remove('DEBTINC**2')
# vars_selectionne.append('DEBTINC**2')
vif_df= f_ts.test_independance_variables(df_clean[vars_selectionne])
vif_df

# Summary d'une reg log

summary_reg_log = f_m.regression_logistique_simple_summary(df_clean,
vars_selectionne, var_y ='BAD')
print(summary_reg_log)

# Analyse points influents
```

```
summary_p_influents = f_m.detecte_points_influents(df_clean, vars_selectionne,
var_y ='BAD', seuil_cook_d = 0.01)
print(summary_p_influents)

# Fine-tuning d'une reg log
best_model, best_params = f_m.regression_logistique_kfold_gridsearch(df_clean,
vars_selectionne, 'BAD')

X = df_clean[vars_selectionne]
y = df_clean['BAD']
y_pred_proba = best_model.predict_proba(X)[:, 1]
y_pred_proba = y_pred_proba.reshape(-1, 1)


f_m.plot_courbe_roc(y, y_pred_proba, color='blue', title= 'Courbe ROC et AUC
pour le modèle de régression logistique')

# Choix du meilleur modèle
resultats_test_modeles = f_m.tester_modeles(df_clean,  vars_selectionne,
target_variable = 'BAD')

# Fine-tuning d'une random forest
best_model, best_params = f_m.random_forest_kfold_gridsearch(df_clean,
vars_selectionne, 'BAD')

y_pred_proba = best_model.predict_proba(X)[:, 1]

f_m.plot_courbe_roc(y, y_pred_proba, color='red', title= 'Courbe ROC et AUC
pour le modèle de Random Forest')

# Clustering
##########################################################################################
#########

X = df_clean[vars_selectionne]
y_pred_proba = best_model.predict_proba(X)[:, 1]
y_pred_proba = y_pred_proba.reshape(-1, 1)


f_m.elbow_method(y_pred_proba)
km, labels, centroids = f_m.K_means(y_pred_proba, 3)

# ajout des clusters du Kmeans à mon df
df_clean['Classe_de_risque'] = labels
centroids
cluster_mapping = {0: 'AAA', 1: 'CCC', 2: 'BBB'}
df_clean['Classe_de_risque'] =
df_clean['Classe_de_risque'].map(cluster_mapping)
```

```
# BAD X Classe de risque
vars_a_traiter = vars_selectionne + ['BAD']

for var in vars_a_traiter:
    print(var)
    df_clean.groupby('Classe_de_risque')[var].mean()
    print()
```

functions_EDA.py

```
"""
Projet Scoring
M2 MoSEF
Louis LEBRETON
Dataset: hmeq.csv

Fonctions pour EDA
"""
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
from scipy.stats import chi2_contingency

def afficher_cat_vars_univarie_graph(df, var_y, palette):

    vars_cat = df.select_dtypes(include=['object', 'category']).columns
    vars_cat = list(vars_cat)
    vars_cat.append(var_y)

    for var in vars_cat:
        plt.figure(figsize=(12, 5))

        total_counts = df[var].value_counts()
        pourcentages = total_counts / total_counts.sum() * 100

        ax = sns.barplot(x=pourcentages.index, y=pourcentages.values,
palette=palette)

        ax.set_ylabel("Pourcentage (%)")
        plt.title(f"Distribution de {var}")

        # pourcentage sur chaque barre
        for p in ax.patches:
            ax.annotate(f'{p.get_height():.1f}%',
                        (p.get_x() + p.get_width() / 2., p.get_height()),
```

```
                        ha='center', va='center', xytext=(0, 8),
textcoords='offset points')

        plt.show()

def afficher_cat_vars_univarie_tableau(df, var_y):
    vars_cat = df.select_dtypes(include=['object', 'category']).columns
    vars_cat = list(vars_cat)
    vars_cat.append(var_y)

    for var in vars_cat:
        tableau_contingence = df[var].value_counts(normalize=True) * 100  #
pourcentage
        print(f"Distribution en % de {var}:\n")
        print(tableau_contingence)
        print("\n" + "-"*50 + "\n")


def afficher_num_vars_univarie_graph(df, palette):
    vars_num = df.select_dtypes(include=['float', 'int']).columns

    for var in vars_num:
        plt.figure(figsize=(12, 5))

        # histo
        plt.subplot(1, 2, 1)
        if df[var].nunique()>30:
            kde_bool = True
        else :
            kde_bool = False
        sns.histplot(df[var], kde=kde_bool, color=palette[0])

        plt.title(f"Histogramme de {var}")
        plt.xlabel(var)
        plt.ylabel("Nombre d'observations")

        # boxplot
        plt.subplot(1, 2, 2)
        sns.boxplot(x=df[var], palette=palette)
        plt.title(f"Boxplot de {var}")
        plt.xlabel(var)

        plt.tight_layout()
        plt.show()



# Bivarie
```

```
def afficher_cat_vars_bivarie_graph(df, y, palette):
    vars_cat = df.select_dtypes(include=['object', 'category']).columns
    for var in vars_cat:
        plt.figure(figsize=(10, 5))
        ax = sns.countplot(data=df, x=var, hue=y, stat="percent",
palette=palette)
        ax.set_ylabel("Pourcent (%)")
        plt.title(f"Distribution de {var} x {y}")
        plt.show()

def afficher_cat_vars_bivarie_tableau(df, y):
    vars_cat = df.select_dtypes(include=['object', 'category']).columns
    for var in vars_cat:
        tableau_freq_rows= pd.crosstab(df[var], df[y], normalize='index') *
100  # pourcentage
        tableau_freq_columns = pd.crosstab(df[var], df[y],
normalize='columns') * 100  # pourcentage
        tableau_freq_all = pd.crosstab(df[var], df[y], normalize='all') *
100  # pourcentage
        print(f"Tableaux de fréquence /lignes pour {var} x {y}:\n")
        print(tableau_freq_rows, "\n")
        print(f"Tableaux de fréquence /colonnes pour {var} x {y}:\n")
        print(tableau_freq_columns, "\n")
        print(f"Tableaux de fréquence /tous pour {var} x {y}:\n")
        print(tableau_freq_all, "\n")
        print("\n" + "-"*50 + "\n")

def afficher_num_vars_bivarie_graph(df, y, palette):
    vars_num = df.select_dtypes(include=['float', 'int']).columns
    vars_num = list(vars_num)
    vars_num.remove(y)

    for var in vars_num:
        plt.figure(figsize=(10, 5))
        sns.boxplot(data=df, x=y, y=var, palette=palette)
        plt.title(f"Boxplots de {var} x {y}")
        plt.show()

def v_cramer_test_khi2(var1, var2):
    """Calcule le V de Cramer entre 2 variables catégorielles"""
    matrice_confusion = pd.crosstab(var1, var2)
    # test Chi-2
    chi2_stat, p_val, _ , _ = chi2_contingency(matrice_confusion)
    n = matrice_confusion.sum().sum()
    r, k = matrice_confusion.shape
    return round(np.sqrt(chi2_stat / (n*(min(r,k)-1))),3), round(p_val,3)

def correlation_ratio(var_cat, var_num):
    """Calcule le eta entre une var cat et une var num"""
```

```
    fcat, _ = pd.factorize(var_cat)
    cat_means = np.array([var_num[fcat == i].mean() for i in
range(len(np.unique(fcat)))])
    overall_mean = var_num.mean()
    numerateur = np.sum([len(var_num[fcat == i]) * (cat_mean - overall_mean)
** 2 for i, cat_mean in enumerate(cat_means)])
    denominateur = np.sum((var_num - overall_mean) ** 2)
    return np.sqrt(numerateur / denominateur)

# Multivarie

def afficher_df_head(df):
    for var in df.columns:
        print(f"{var}:\n{df[var].head()}\n")

def afficher_matrice_correlation_num(df, methode='pearson'):
    # methode: pearson, kendall ou spearman
    df_num = df.select_dtypes(include=['float64', 'int64'])
    matrice_correlation = df_num.corr(method = methode)

    plt.figure(figsize=(15, 10))
    sns.heatmap(matrice_correlation, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5)
    plt.title('Matrice de corrélation')
    plt.show()
```

functions_pretraitement.py

```
"""
Projet Scoring
M2 MoSEF
Louis LEBRETON
Dataset: hmeq.csv

Fonctions pour le prétraitement des données
"""
from itertools import combinations_with_replacement

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.ensemble import RandomForestRegressor

# Traitement des valeurs manquantes
```

```
def afficher_pourcentage_valeurs_manquantes(df):
    pourcentage_manquantes = df.isna().mean() * 100
    pourcentage_manquantes =
pourcentage_manquantes.sort_values(ascending=False)
    print("Pourcentage de valeurs manquantes par variable (en %) :\n")
    print(round(pourcentage_manquantes), 2)

def afficher_nb_valeurs_manquantes(df):
    lignes_avec_n_manquantes = df.isna().sum(axis=1) # nb de valeurs
manquantes par ligne
    return lignes_avec_n_manquantes

def afficher_valeurs_manquantes_barplot(df):

    missing_values = (df.isnull().sum() / len(df)) * 100
    missing_values = missing_values.sort_values(ascending=False)
    # barplot
    plt.figure(figsize=(10, 6))
    sns.barplot(x=missing_values.index, y=missing_values, palette='viridis')
    plt.xlabel('Variable')
    plt.ylabel('Pourcentage de valeurs manquantes (%)')
    plt.title('Pourcentage de valeurs manquantes par variable')
    plt.show()


def random_forest_imputation(df, colonne_a_imputer):
    df_complet = df[df[colonne_a_imputer].notna()]
    df_manquant = df[df[colonne_a_imputer].isna()]

    features =
df_complet.select_dtypes(include=[np.number]).drop(columns=[colonne_a_imputer,
'BAD']).columns.tolist()

    X = df_complet[features]
    y = df_complet[colonne_a_imputer]

    model = RandomForestRegressor(random_state=999)
    model.fit(X, y)

    df_manquant[colonne_a_imputer] = model.predict(df_manquant[features])

    # concatenation du df complet avec le df manquant imputé
    df_impute = pd.concat([df_complet, df_manquant]).sort_index()

    return df_impute

# Normalisation

def normaliser_variables(df, y, scaler = "standard" ):
```

```
    vars_num = df.select_dtypes(include=['float', 'int']).columns
    vars_num = list(vars_num)
    vars_num.remove(y)

    if scaler == 'standard':
        scaler = StandardScaler()
    elif scaler == 'minmax':
        scaler = MinMaxScaler()
    elif scaler == 'robust':
        scaler = RobustScaler()

    df[vars_num] = scaler.fit_transform(df[vars_num])

    return df

def creation_variables_interaction(df, cols, polynomial = True):
    """
    Cette fonction permet de créer des variables d'intéraction à partir de
colonnes données
    d'un df d'entrée
    """

    # boucle sur toutes les combinaisons avec remplacement
    for col1, col2 in combinations_with_replacement(cols, 2):

        if col1 == col2:
            if polynomial:
                df[f'{col1}**2'] = df[col1] ** 2
        else:
            # Si les deux colonnes sont différentes -> interaction
            df[f'{col1}*{col2}'] = df[col1] * df[col2]

    return df
```

functions_tests_statistiques.py

```
"""
Projet Scoring
M2 MoSEF
Louis LEBRETON
Dataset: hmeq.csv

Fonctions pour les tests statistiques
"""
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

```python
from sklearn.linear_model import LogisticRegression
from statsmodels.stats.stattools import durbin_watson
from statsmodels.stats.outliers_influence import variance_inflation_factor
import pandas as pd


# Test homoscedasticite

def test_homoscedasticite(X, y):

    # modele logistique
    log_modele = LogisticRegression(max_iter=1000)
    log_modele.fit(X, y)

    # probas predites
    y_pred_prob = log_modele.predict_proba(X)[:, 1]

    # résidus de Pearson
    residus_pearson = (y - y_pred_prob) / np.sqrt(y_pred_prob * (1 -
y_pred_prob))

    # graph des résidus vs probabilités prédites
    plt.scatter(y_pred_prob, residus_pearson, alpha=0.5)
    plt.axhline(0, color='red', linestyle='--')
    plt.xlabel('probabilités prédites')
    plt.ylabel('résidus de Pearson')
    plt.title('résidus de Pearson x probabilités prédites')
    plt.show()


# Test Indépendance des erreurs

def test_independance_erreurs(X, y):
    # modele logistique
    log_modele = LogisticRegression(max_iter=1000)
    log_modele.fit(X, y)

    # probas predites
    y_pred_prob = log_modele.predict_proba(X)[:, 1]

    # residus
    residuals = y - y_pred_prob

    # stats de durbin watson
    dw_stat = durbin_watson(residuals)

    # analyse du résultat
    if dw_stat < 1.5:
        print(f'autocorrélation positive détectée (DW = {dw_stat:.2f}).')
```

```
    elif dw_stat > 2.5:
        print(f'autocorrélation négative détectée (DW = {dw_stat:.2f}).')
    else:
        print(f'pas d\'autocorrélation significative détectée (DW =
{dw_stat:.2f}).')

    return dw_stat


# Test indépendance des variables

def test_independance_variables(X):
    # ajout constante
    X = sm.add_constant(X)
    # calcul du VIF
    vif_data = pd.DataFrame()
    vif_data["variable"] = X.columns
    vif_data["VIF"] = [round(variance_inflation_factor(X.values, i),2) for i
in range(X.shape[1])]

    return vif_data
```

functions_modelisation.py.

```
"""
Projet Scoring
M2 MoSEF
Louis LEBRETON
Dataset: hmeq.csv

Fonctions pour modélisation
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, accuracy_score,
precision_score, recall_score, f1_score, confusion_matrix
from xgboost import XGBClassifier
from sklearn.metrics import roc_curve, auc
```

```python
# Choix des variables

def stepwise_selection(df, y, threshold_in=0.05, threshold_out=0.05,
verbose=True):
    """
    Selectionne pas-à-pas une variable à ajouter dans le modèle et une autre à
enlever du modèle
    à chaque itération

    """
    X = df.drop(columns=[y])
    y = df[y]

    # liste des variables à selectionner
    list_var = []
    changement = True

    while changement:
        changement = False
        # vars restantes
        vars_restantes = list(set(X.columns) - set(list_var))
        new_pval = pd.Series(index=vars_restantes, dtype=float) # pvalue des
vars à tester

        for var_a_tester in vars_restantes:
            model = sm.Logit(y, sm.add_constant(pd.DataFrame(X[list_var +
[var_a_tester]]))).fit(disp=0)
            new_pval[var_a_tester] = model.pvalues[var_a_tester]

        best_pval = new_pval.min() # choix de la pvalue la plus basse
        if best_pval < threshold_in:
            changement = True
            best_var = new_pval.idxmin()
            list_var.append(best_var) # ajout d'1 var
            if verbose:
                print(f'ajout de {best_var} avec p-valeur = {best_pval:.4}')

        # modele logistique
        model = sm.Logit(y,
sm.add_constant(pd.DataFrame(X[list_var]))).fit(disp=0)

        pvalues = model.pvalues.iloc[1:]  # on ignore la constante
        worst_pval = pvalues.max()  # choix de la pvalue la plus haute

        if worst_pval > threshold_out:
            changement = True
            worst_var = pvalues.idxmax()
            list_var.remove(worst_var) # retrait d'1 var
```

```
            if verbose:
                print(f'retrait de {worst_var} avec p-valeur =
{worst_pval:.4}')

    if verbose:
        print("\n" + "-"*50 + "\n")
        print('nombre de variables dans le df: ', len(df.columns))
        print('nombre de variables selectionnées: ', len(list_var))

    return list_var



# regression logistique classique
def regression_logistique_simple_summary(df, vars_selectionne, var_y ='BAD'):
    """
    Summary d'une regression logistique classique
    Statsmodel plus adapté ici pour obtenir odds ratios etc
    """
    X = df[vars_selectionne]
    y = df[var_y]

    X = sm.add_constant(X)

    logit_model = sm.Logit(y, X)
    result = logit_model.fit()

    params = result.params  # coeffs
    summary = round(result.conf_int(),2)  # IC
    summary['Odds Ratio'] = params.apply(lambda x: round(np.exp(x),2))  # odds
ratios
    summary.columns = ['2.5%', '97.5%', 'Odds Ratio']
    summary['p-value'] = round(result.pvalues,3) # p-values

    return summary


def detecte_points_influents(df, vars_selectionne, var_y ='BAD', seuil_cook_d
= 0.1):
    """
    Identifie les points influents avec les distances de Cook
    """
    X = df[vars_selectionne]
    y = df[var_y]

    X = sm.add_constant(X)

    logit_model = sm.Logit(y, X)
    result = logit_model.fit()
```

```
    influence = result.get_influence()

    # distances de cook
    cooks_d = influence.cooks_distance[0]

    points_influents = np.where(cooks_d > seuil_cook_d)[0]
    summary_points_influents = pd.DataFrame({
        'index': points_influents,
        'cooks_Distance': cooks_d[points_influents]
    })

    return summary_points_influents.sort_values(ascending = False, by =
'cooks_Distance')

# Fine-tuning d'une reg log
def regression_logistique_kfold_gridsearch(df, var_x, var_y, k_folds = 5):
    """
    Permet de fine tuner une regression logistique grâce à une cross
validation (k-fold) gridsearch
    """
    X = df[var_x]
    y = df[var_y]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=999, stratify=y)

    # hyperparamètres
    param_gridsearch = {
        'penalty': ['l1', 'l2', 'elasticnet'],
        'C': [0.05, 0.5, 1],
        'solver': ['saga']
    }

    # modèle logistique
    log_reg = LogisticRegression(max_iter=1000)

    # gridSearch: k-fold cross-validation
    grid_search = GridSearchCV(log_reg, param_gridsearch, cv=k_folds,
scoring='f1')
    grid_search.fit(X_train, y_train)

    # meilleurs hyperparamètres
    best_params = grid_search.best_params_
    print(f"Meilleurs hyperparamètres : {best_params}")

    # evaluer le modèle sur l'ensemble de test
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)
```

```
    # rapport de classification
    print("\nRapport de classification sur l'ensemble de test :")
    print(classification_report(y_test, y_pred))

    print("accuracy score : ", accuracy_score(y_test,y_pred))
    print("precision score : ", precision_score(y_test, y_pred,
average="macro"))
    print("recall score : ", recall_score(y_test, y_pred, average="macro"))
    print("f1 score : ", f1_score(y_test, y_pred, average="macro"))

    # matrice de confusion
    conf_matrix = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 5))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=['Classe 0', 'Classe 1'], yticklabels=['Classe 0', 'Classe 1'])
    plt.title('Matrice de confusion pour le modèle de régression logistique')
    plt.xlabel('valeurs prédites')
    plt.ylabel('valeurs réelles')
    plt.show()

    return best_model, best_params

# Choix du modèle

def tester_modeles(df_norm, selected_variables, target_variable):
    X = df_norm[selected_variables]
    y = df_norm[target_variable]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=999)

    # modeles à tester
    modeles = {
        'Logistic Regression': LogisticRegression(max_iter=1000),
        'Decision Tree': DecisionTreeClassifier(),
        'Random Forest': RandomForestClassifier(),
        'Gradient Boosting': GradientBoostingClassifier(),
        'XGBoost': XGBClassifier(eval_metric='logloss')

    }

    resultats = {}

    for nom_model, modele in modeles.items():
        modele.fit(X_train, y_train)
        y_pred = modele.predict(X_test)

        # metriques
        accuracy = accuracy_score(y_test, y_pred)
```

```
        precision = precision_score(y_test, y_pred)
        rappel = recall_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)

        resultats[nom_model] = {
            'Accuracy': accuracy,
            'Precision': precision,
            'Recall': rappel,
            'F1-Score': f1
        }

    resultats_df = pd.DataFrame(resultats).T
    return resultats_df


# Fine-tuning d'un Random Forest
def random_forest_kfold_gridsearch(df, var_x, var_y, k_folds=5):
    """
    Permet de fine tuner une random forest grâce à une cross validation (k-fold) gridsearch
    """
    X = df[var_x]
    y = df[var_y]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=999, stratify=y)

    # hyperparametres testés
    param_gridsearch = {
        'n_estimators': [100, 200, 300],
        'max_depth': [10, 20, 30, None],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'bootstrap': [True, False]
    }

    rf = RandomForestClassifier(random_state=999)

    # gridSearch k-fold crossvalidation
    grid_search = GridSearchCV(rf, param_gridsearch, cv=k_folds, scoring='f1', n_jobs=-1)
    grid_search.fit(X_train, y_train)


    best_params = grid_search.best_params_
    print(f"Meilleurs hyperparamètres : {best_params}")

    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)
```

```
    # rapport de classification
    print("\nRapport de classification sur l'ensemble de test :")
    print(classification_report(y_test, y_pred))

    print("accuracy score : ", accuracy_score(y_test, y_pred))
    print("precision score : ", precision_score(y_test, y_pred,
average="macro"))
    print("recall score : ", recall_score(y_test, y_pred, average="macro"))
    print("f1 score : ", f1_score(y_test, y_pred, average="macro"))

    # Matrice de confusion
    conf_matrix = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 5))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=['Classe 0', 'Classe 1'], yticklabels=['Classe 0', 'Classe 1'])
    plt.title('Matrice de confusion pour le Random Forest')
    plt.xlabel('valeurs prédites')
    plt.ylabel('valeurs réelles')
    plt.show()

    return best_model, best_params

# Clustering : K-Means

def elbow_method(X):
    # elbow method
    inertia_list = []

    for k in range(1, 11):
        kmeans = KMeans(n_clusters=k, random_state=999)
        kmeans.fit(X)
        inertia_list.append(kmeans.inertia_)

    plt.figure(figsize=(10, 5))
    plt.plot(range(1, 11), inertia_list, marker='o')
    plt.xlabel('Nombre de clusters')
    plt.ylabel('Inertie intra-cluster')
    plt.title('Méthode du coude')
    plt.grid(True)
    plt.show()

def K_means(X, k):

    kmeans = KMeans(n_clusters=k, random_state=999)
    kmeans.fit(X)

    labels = kmeans.labels_
    centroids = kmeans.cluster_centers_
```

```
    return kmeans, labels, centroids

def plot_courbe_roc(y, y_pred_proba, title:str, color:str):
    """
    fonction pour tracer la courbe ROC
    """
    fpr, tpr, _ = roc_curve(y, y_pred_proba)
    roc_auc = auc(fpr, tpr)

    # traçage
    plt.figure()
    plt.plot(fpr, tpr, color=color, lw=2, label=f'Courbe ROC (AUC =
{roc_auc:.3f})')
    plt.plot([0, 1], [0, 1], color='grey', linestyle='--', lw=2)
    plt.xlabel('taux de faux positifs')
    plt.ylabel('taux de vrais positifs')
    plt.title(title)
    plt.legend(loc='lower right')
    plt.grid(True)
    plt.show()
```