Cergy Paris Université

Institut Économique et de Gestion

Master 2 Professionnel

Ingénierie Économique et Analyse de Données

Projet : Prédiction de non solvabilité d'un client

Étudiants :
SYLLA Mamadou Daya
DJRAMEDO Chris
YUZUAK Ali

Enseignant Référent :
KENGNE William

10 janvier 2022

Résumé

Ce présent document constitue un rapport d'études en Python dans le cadre la validation de notre 2ème année de Master Professionnel spécialité Ingénierie Économique et Analyse de données.

Ce projet d'étude a pour ambition d'améliorer la stratégie d'attribution de crédit d'une banque. En effet, cette dernière souhaite prédire la non solvabilité d'un client et de construire un nouveau score de risque ; car son score disponible étant devenu obsolète.

Ce document présente également les différentes étapes effectuées dans la réalisation de ce projet, en commençant par l'évolution de la couche présentation, jusqu'à l'élaboration du score de risque permettant de définir la priorité dans l'attribution de crédit.

A titre informatif, toutes les sorties de ce document ont été réalisées avec le logiciel Python.

Table des matières

Introduction

Le succès de la numérisation et l'émergence des objets inter-connectés poussent Internet à un essor sans précédent. Avec les progrès de la technologie, le réseau devient un important référentiel de données et, grâce à l'expansion des informations, il continue de croître chaque jour. C'est dans cet environnement complexe que les entreprises doivent émerger, faire face à la diversification des sources de données et à la quantité massive d'informations qu'elles doivent analyser pour prendre des décisions établies. Par conséquent, le «data mining» a tout son sens entre les mains de l'entreprise et constitue un outil puissant pour explorer et analyser les données décisionnelles.

Ce projet fait partie du cours de data mining. Grâce à ses méthodes théoriques, nous aiderons les banques à analyser leurs clients en fonction de leur solvabilité et à créer des scores de risque.

Pour cela, nous allons d'abord nettoyer et préparer une base de données que nous pourrons exploiter, grâce à une analyse descriptive de chaque variable. Nous allons ensuite estimer différents modèles pour retenir le modèle qui prédit le mieux la capacité du client à rembourser ou à ne pas rembourser le prêt bancaire. Enfin, nous établirons un score de risque pour chaque client.

Chapitre 1

Définitions et objectifs

1.1 Objectifs pratiques

L'objectif principal de notre travail est de prédire la solvabilité des clients d'une banque. Il s'agira donc de distinguer notre population de recherche (clients) en fonction du risque de crédit, c'est-à-dire de classer les clients solvables et les clients peu fiables. Deuxièmement, nous devons développer un score de risque à attribuer aux nouveaux clients et aux demandeurs de prêt, ce qui permettra à la banque de leur octroyer des prêts sur la base

— i) du score

— ii) des ressources disponibles de la banque.

Par conséquent, notre unité statistique est la clientèle de la banque.

1.2 Problématique de l'étude

Nous sommes confrontés au problème de la supervision de la classification et de la notation dans notre travail actuel. Il s'agit en effet de classer les clients en fonction de leur solvabilité. Plus tard, nous créerons un score de risque qui nous permettra de classer les nouveaux documents de demande de prêt.

1.3 Application

Les résultats de cette recherche permettront à la banque de construire un modèle pour prédire la probabilité qu'un de ses clients rembourse le prêt. Il peut alors rejeter des documents qui comportent un risque considérable d'insolvabilité. C'est grâce à son nouveau score de risque.

Chapitre 2

Explorations et analyse des données

Pour notre projet, nous disposons d'une base de données à partir de l'historique des clients qui ont fait une demande de crédit. Par conséquent, la banque nous a fourni des informations sur les caractéristiques de ces clients.
Il est important de souligner que tous les remboursements sont dus, ainsi nous pouvons comprendre les clients qui peuvent rembourser le crédit sans impact et les clients qui remboursent avec impact.

Pour la réussite de notre projet et des analyses et modèles fiables, nous allons d'abord explorer et préparer nos données. La description des variables de recherche sera effectuée avant l'analyse bivarié pour déterminer la corrélation entre elles, afin de sélectionner les variables à considérer dans la suite de l'analyse. Après avoir terminé ce travail, le fractionnement des données nous permettra de construire deux échantillons : des échantillons d'apprentissage et des échantillons de test. Nous utiliserons ces deux échantillons lors de la construction de nos différents modèles prédictifs.

2.1 Statistiques descriptives

2.1.1 Description et renommage de variables

Dans le but d'avoir un aperçu général et rapide des données et d'en juger la fiabilité, il est important de procéder à une analyse préliminaire via des statistiques descriptives. Nous entamons ainsi notre projet par la prise de connaissance des différentes variables et de leurs types et nous procédons également à leurs renommages pour des questions de simplicité de lecture. Notre base de données **Tab 2.1** est composée de 12 variables allant de A à L et d'un échantillon de 3 574 individus. Le tableau ci-dessous (Tableau descriptif des variables) montre le renommage adopté ainsi que le type de chaque variable.
Nous avons utilisé la fonction *Rename* de Pandas pour effectuer ce renommage.

NB : Noter bien qu'à cette étape nous procédons à un nouveau formatage et recodage des variables afin de faciliter l'intuition et l'interprétation qui en découle. Nous transformerons ainsi la variable *Age_cred* en années. Et la variable qualitative *Incident_r* prendra la modalité **"OUI"** lorsque *Incident_r* sera égal à **"1"** et **"NON"** lorsque *Incident_r* sera égal à **"0"**.

Var	Signification	Type	Renommage
A	Remboursement du crédit sans incident	Quantitative discrète	Incident_r
B	Montant de la demande de prêt	Quantitative continue	Montant_pret
C	Montant hypothèque	Quantitative continue	Montant_hypotheque
D	Valeur propriété	Quantitative continue	Val_propriete
E	Motif du prêt	Qualitative	Motif_pret
F	Profession du demandeur	Qualitative	Profession
G	Nombre d'années dans le travail actuel	Quantitative continue	Nb_annees_travail
H	Nombre de demande de report d'échéances de prêt	Quantitative continue	Nb_report_pret
I	Nombre de litiges	Quantitative discrète	Nb_litiges
J	Âge du plus ancien crédit	Quantitative continue	Age_cred
K	Nombre de demandes récentes de crédit	Quantitative discrète	Nb_demandes_cred
L	Ratio dette sur revenu	Quantitative continue	Ratio_dette_revenu

TABLE 2.1 – *Tableau descriptif des variables*

2.1.2 Distribution des variables

Variables	N	Moy	Écart-Type	Q1	Q2	Q3	Q4
Montant_pret	3574	18478.1198	11182.0003	11000	16100	23000	89800
Montant_hypotheque	3265	74081.5625	44517.512	46731	65372	92241	399412
Val_propriete	3506	102466.356	58529.3985	66782	89640.5	120470	854114
Nb_annees_travail	3268	8.9620716	7.56808897	3	7	13	41
Nb_report_pret	3143	0.26057906	0.89442465	0	0	0	10
Nb_litiges	3220	0.45465839	1.13822294	0	0	0	15
Age_cred	3391	180.76624	88.1121203	115.08	174.03	232.32	1168.23
Nb_demandes_cred	3265	1.18591118	1.71553527	0	1	2	17
Ratio-dette-revenu	2797	33.6307276	8.05350423	29.058	34.634	38.974	133.528

TABLE 2.2 – *Tableau de distribution des variables quantitatives*

(TABLE 2.2) :
La fonction *dfp.describe(include="all")* nous permet d'avoir la distribution de nos variables. Les statistiques descriptives de nos variables révèlent que l'âge du plus ancien crédit d'un client est en moyenne 181 mois (15 ans) alors que le nombre d'années dans le travail actuel est en moyenne pour notre population presque 9 ans. Soulignons que les banques se basent généralement dans la décision d'octroi de crédit sur la stabilité financière du client (du demandeur) et sur son ancienneté dans son poste professionnel.

Par ailleurs, il n'y a presque pas de report de prêt et moins de 2 demandes de crédit par client en moyenne.

2.1.3 Fréquences de valeurs manquantes

(TABLE 2.3) : *(fréquences des valeurs manquantes)*
Pour la détection des valeurs manquantes nous utilisons respectivement les fonctions *isna().any()* et *isnull().sum()* sur Python. En considérant un niveau de tolérance de présence des valeurs manquantes de 10%, nous remarquons que les variables *Nb_report_pret* (12,06%) et *Ratio_dette_revenu* (21,29%) ont des proportions de valeurs manquantes bien au-dessus du seuil défini. La variable définissant l'âge du crédit (9,90%), le nombre de demande du prêt et le montant de l'hypothèque (8,65%) n'en sont pas bien loin . Il s'avèrent donc important de procéder à une imputation des variables.
Dans la suite de notre étude, nous déciderons d'imputer toutes les variables ayant des valeurs manquantes, quelques soit la proportion, avec une méthode adaptée que nous ne manquerons d'expliquer.

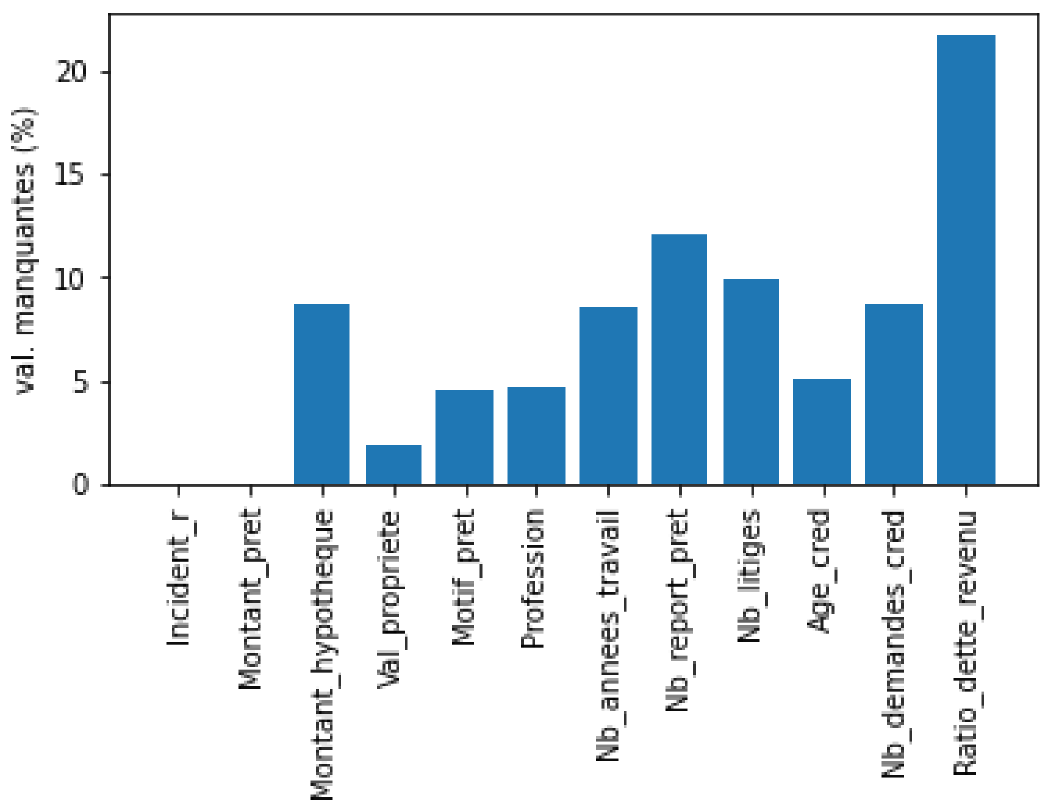


FIGURE 2.1 – Histogramme des fréquences des valeurs manquantes

2.2 Détection de valeurs manquantes et aberrantes

Pour la détection des valeurs aberrantes, nous utilisons la fonction boxplot() sur Python.

Fig 2.2 : *(Montant_pret)*
En observant la boîte à moustache, nous notons une présence d'un nombre significatif de valeurs dépassant le maximum des *Montant_pret*. Sans prendre en compte les variables qui sont au voisinage du maximum, nous concluons qu'il y a bel et bien présence de valeurs aberrantes.

Fig 2.3 : *(Montant_hypotheque)*
Nous observons un bloc de points dépassant légèrement le voisinage du maximum et puis nous remarquons la présence de 2 blocs de points se détachant complètement et qui sont beaucoup plus excentrés. D'où la présence de valeurs aberrantes.

Fig 2.4 : *(Val_propriete)*
Il existe un nombre significatif de valeur au voisinage du maximum de *(Val_propriete)* mais nous ne les prendront pas en compte. Il apparaît juste un petit groupe de points excentrés que nous considérons comme valeurs aberrantes.

Fig 2.5 : *(Age_cred)*
Nous observons deux blocs de points proche du voisinage du maximum que nous ne prendront pas en compte. Il existe en effet un bloc de points se détachant du maximum de *(Age_cred)*. Il y a donc présence de valeurs aberrantes.

Fig 2.6 : *(Ratio_dette_revenu)*
Nous remarquons que parmi les points proches du voisinage du maximum et minimum de *(Ratio_dette_revenu)* il y a des points se détachant complètement. Ce détachement est synonyme de la présence de valeurs aberrantes.

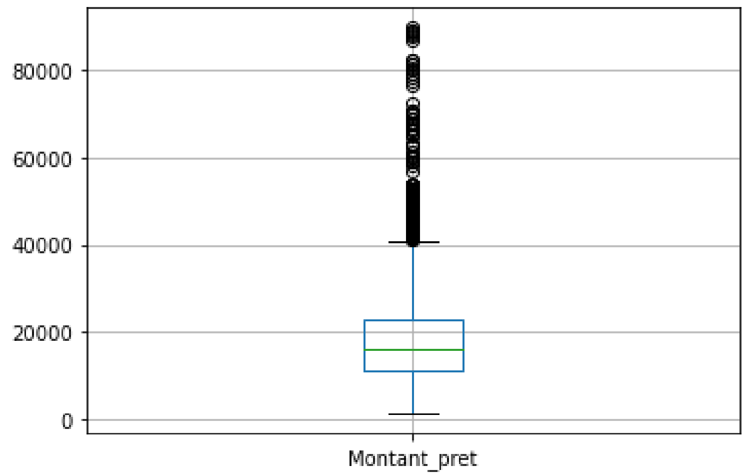


FIGURE 2.2 – Boxplot de *Montant_pret*

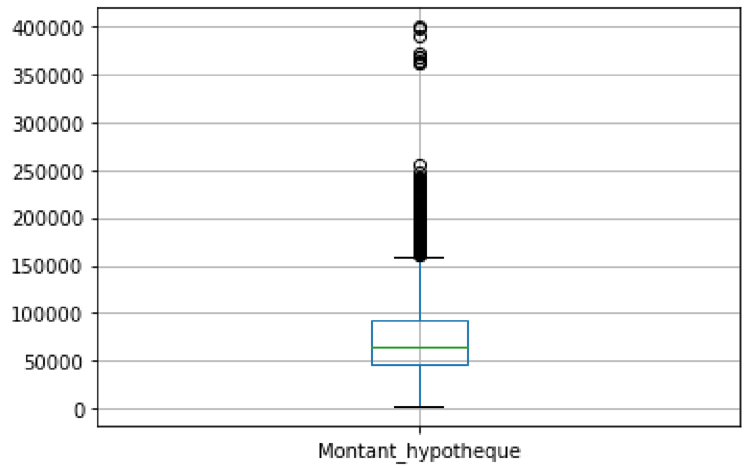


FIGURE 2.3 – Boxplot de *Montant_hypotheque*

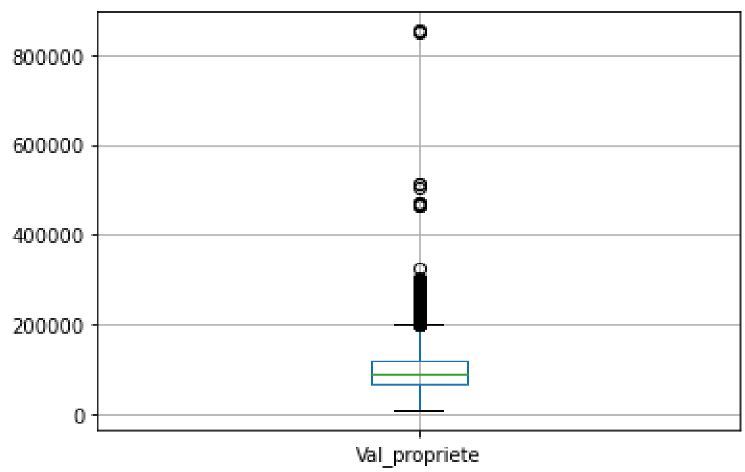


FIGURE 2.4 – Boxplot de *Val_propriete*

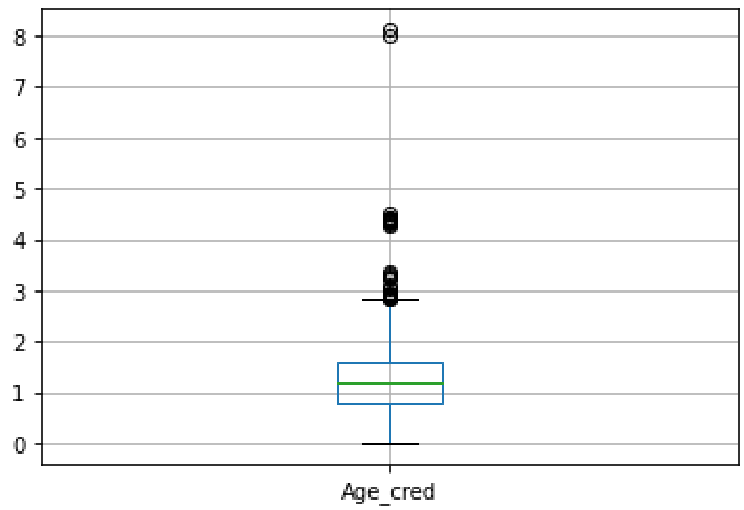


FIGURE 2.5 – Boxplot de *Age_ cred*

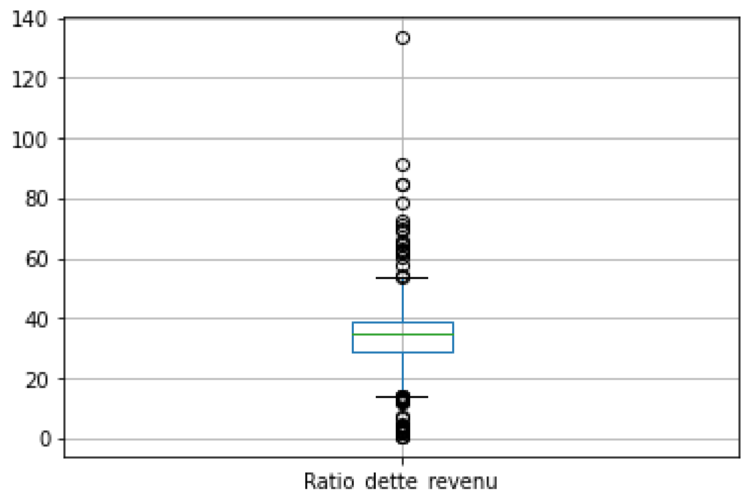


FIGURE 2.6 – Boxplot de *Ratio_ dette_ revenu*

Certaines variables peuvent être fortement corrélées entre elles. Lorsque 2 variables sont fortement corrélées, elles peuvent avoir une dépendance c'est-à-dire l'une ou l'autre auraient le même rôle dans le modèle. Il est donc important de capter cette possible redondance des informations. Pour cela, nous regarderons la matrice de corrélation des variables.

La corrélation entre les variables est forte lorsque le coefficient de corrélation est supérieur ou égal à 0.8 (ou bien inférieur ou égal à 0.8). La corrélation peut même être parfaite lorsque le coefficient de corrélation est égal à 1 ou -1.
Enfin la corrélation est nulle lorsque le coefficient de corrélation vaut 0.

Nous utilisons la fonction suivante *corr.style.background_ gradient(cmap='coolwarm').set_ precision(2)* pour avoir en sortie, la matrice de corrélation sous le format de l'image suivante avec une précision des nombre décimaux à 2 chiffres après la virgule.

Index	Incident_r	Montant_pret	ntant_hypothe	Val_propriete	b_annees_trav	Vb_report_pre	Nb_litiges	Age_cred	o_demandes_cr	tio_dette_reve
Incident_r	1	-0.083964	-0.0472966	-0.0319677	-0.0540615	0.290327	0.351471	-0.179098	0.172803	0.187137
Montant_pret	-0.083964	1	0.228632	0.331467	0.10034	0.00227975	-0.0288669	0.0919563	0.0436001	0.069771
Montant_hypotheque	-0.0472966	0.228632	1	0.879263	-0.0740483	-0.0558589	0.00564569	0.135285	0.0264561	0.136512
Val_propriete	-0.0319677	0.331467	0.879263	1	0.0188743	-0.0456679	-0.0134419	0.166388	-0.00714122	0.114459
Nb_annees_travail	-0.0540615	0.10034	-0.0740483	0.0188743	1	-0.0608039	0.0578732	0.199575	-0.0802344	-0.0504636
Nb_report_pret	0.290327	0.00227975	-0.0558589	-0.0456679	-0.0608039	1	0.217693	-0.0894159	0.198983	0.0448799
Nb_litiges	0.351471	-0.0288669	0.00564569	-0.0134419	0.0578732	0.217693	1	0.023154	0.0641888	0.044349
Age_cred	-0.179098	0.0919563	0.135285	0.166388	0.199575	-0.0894159	0.023154	1	-0.131706	-0.0496582
Nb_demandes_cred	0.172803	0.0436001	0.0264561	-0.00714122	-0.0802344	0.198983	0.0641888	-0.131706	1	0.146842
Ratio_dette_revenu	0.187137	0.069771	0.136512	0.114459	-0.0504636	0.0448799	0.044349	-0.0496582	0.146842	1

FIGURE 2.7 – Matrice de Corrélation **avant** Imputation

La matrice représenté sur la figure *(Fig 2.7)* révèle la présence d'une forte corrélation entre les variables *Montant_ hypotheque* et *Val_ propriete*. Le coefficient de corrélation est estimé à 0.879263, soit environ 88% de corrélation.
Nous pouvons ainsi déduire que ces 2 variables apportent la même information. Par la suite, parmi les variables corrélées, nous ne garderons que celle qui présentera le plus faible pourcentage de valeurs manquantes.
Nous décidons donc de garder la variable *Val_ propriete* et nous supprimerons la variable *Montant_ hypotheque* pour la suite de l'analyse.

2.3 Traitement des valeurs manquantes et aberrantes

Une valeur aberrante est une valeur ou une observation qui est distante des autres observations effectuées sur le même phénomène. Ces données aberrantes peuvent apparaître par hasard dans n'importe quelle distribution et elles indiquent souvent une erreur de mesure ou une période marquée par un événement particulier.

Une fois repérées, les valeurs aberrantes doivent être traitées. Pour réaliser ces traitements, on effectue une correction des valeurs marquantes en utilisant des techniques d'imputation comme pour les valeurs manquantes ou encore les éliminer définitivement.

Plusieurs méthodes existent pour traiter les données manquantes à savoir l'imputation simple, l'imputation multiple, l'imputation par la moyenne, l'imputation par les KNN ou bien l'imputation par le mode.
Dans notre projet, nous distinguerons le traitement de nos variables. Nous utiliserons l'imputation par les **KNN** pour les variables quantitatives.
Pour réaliser l'imputation sur les variables quantitatives, nous avons transformer les valeurs aberrantes en valeurs manquantes.

En pratique, en utilisant l'approche KNN, nous spécifions une distance par rapport aux valeurs manquantes. Les valeurs manquantes seront prédites en fonction de la moyenne des voisins.
Par la suite, nous avons définis une fonction

optimize_k dans le but de déterminer le *k-optimal* des voisins candidats pour l'imputation, puis utiliser la librairie **KNNImputer** de Python pour faire l'imputation.

Concernant les variables qualitatives nous utilisons l'imputation par le **mode** via la fonction *fillna*. Le mode correspond à la valeur la plus représentée dans la population de nos variables.

Imputation des variables quantitatives :

```
In [17]: print(dfp.describe())
        Incident_r  Montant_pret  Montant_hypotheque  Val_propriete  \
count  3574.000000   3574.000000         3265.000000    3506.000000
mean      0.204253  18478.119754        74081.562484  102466.355687
std       0.403211  11182.000312        44517.511972   58529.398471
min       0.000000   1300.000000         2619.000000    8000.000000
25%       0.000000  11000.000000        46731.000000   66787.000000
50%       0.000000  16100.000000        65372.000000   89640.500000
75%       0.000000  23000.000000        92241.000000  120465.250000
max       1.000000  89800.000000       399412.000000  854114.000000

       Nb_annees_travail  Nb_report_pret  Nb_litiges     Age_cred  \
count        3268.000000     3143.000000  3220.000000  3391.000000
mean            8.962072        0.260579     0.454658    15.063810
std             7.568089        0.894425     1.138223     7.342625
min             0.000000        0.000000     0.000000     0.000000
25%             3.000000        0.000000     0.000000     9.590000
50%             7.000000        0.000000     0.000000    14.500000
75%            13.000000        0.000000     0.000000    19.360000
max            41.000000       10.000000    15.000000    97.350000

       Nb_demandes_cred  Ratio_dette_revenu
count       3265.000000         2797.000000
mean           1.185911           33.630728
std            1.715535            8.053504
min            0.000000            0.524000
25%            0.000000           29.058000
50%            1.000000           34.634000
75%            2.000000           38.974000
max           17.000000          133.528000
```

FIGURE 2.8 – ↑ Variables quantitatives **avant** imputation

Nous avons décidé de considérer comme valeurs aberrantes :

— Toute valeur supérieure à 50 pour la variable Age_cred
— Toute valeur supérieure à 320 000 pour la variable Valeur_propriete
— Toute valeur supérieure à 270 000 pour la variable Montant_hypotheque
— Toute valeur supérieure à 55 000 pour la variable Montant_pret
— Toute valeur supérieure à 75 et inférieur à 17 pour la variable Ratio_dette_revenu

```
In [48]:
    ...:
    ...: rmse = lambda y, yhat: np.sqrt(mean_squared_error(y, yhat))
    ...:
    ...: def optimize_k (data, target):
    ...:     errors = [] # liste vide qui va contenir les erreurs calculées
    ...:     for k in range(1, 20, 1):
    ...:         imputer = KNNImputer(n_neighbors=k)
    ...:         imputed = imputer.fit_transform(data)
    ...:         quant_var_imputed = pd.DataFrame(imputed, columns=quant_var.columns)
    ...:
    ...:         X = quant_var_imputed.drop(target, axis=1)
    ...:         y = quant_var_imputed[target]
    ...:         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
    ...:
    ...:         model = RandomForestRegressor()
    ...:         model.fit(X_train, y_train)
    ...:         preds = model.predict(X_test)
    ...:         error = rmse(y_test, preds)
    ...:         errors.append({'K': k, 'RMSE': error})
    ...:     return errors

In [49]: k_errors = optimize_k(data=quant_var, target='Incident_r')
    ...: k_errors
Out[49]:
[{'K': 1, 'RMSE': 0.3192925765144712},
 {'K': 2, 'RMSE': 0.309520947244636},
 {'K': 3, 'RMSE': 0.29863465529415256},
 {'K': 4, 'RMSE': 0.29882426952298247},
 {'K': 5, 'RMSE': 0.3023254856949697},
 {'K': 6, 'RMSE': 0.29747374568292473},
 {'K': 7, 'RMSE': 0.2998733998607678},
 {'K': 8, 'RMSE': 0.3010150659519718},
 {'K': 9, 'RMSE': 0.2984470294216871},
 {'K': 10, 'RMSE': 0.29931623709750493},
 {'K': 11, 'RMSE': 0.2939214014150555},
 {'K': 12, 'RMSE': 0.301156047648001},
 {'K': 13, 'RMSE': 0.3036403835419171},
 {'K': 14, 'RMSE': 0.2980277830011146},
 {'K': 15, 'RMSE': 0.2986810165550359},
 {'K': 16, 'RMSE': 0.29680560377723064},
 {'K': 17, 'RMSE': 0.29853488392996375},
 {'K': 18, 'RMSE': 0.2998789965759732},
 {'K': 19, 'RMSE': 0.2995374055817693}]
```

FIGURE 2.9 – Imputation par la méthode **K-NN** Neighbors des variables qualitatives

Fig 2.9 - Fig 2.10 : *(Variables quantitatives)*
Le nombre d'observations après imputation est égal 3574 pour chacune des variables (5 variables citées précédemment en section **2.2**).
L'écart type après imputation a baissé pour l'ensemble des variables. Lorsque l'écart type diminue, la variance diminue également, ce qui entraîne une augmentation de la précision de l'estimation. Ce qui réponds à l'objectif de l'imputation.

Fig 2.11 - Fig 2.12 : *(Variables qualitatives)*
Nous pouvons remarqué que la fréquence des modalités des deux variables qualitatives (*Motif_pret* et *Profession*) a augmenté puisqu'il y avait des valeurs manquantes avant imputation.

```
In [13]: print(quant_var.describe())    # Resumé statistiques de la base de
données après imputation
        Incident_r  Montant_pret  Montant_hypotheque  Val_propriete  \
count  3574.000000   3574.000000         3574.000000    3574.000000
mean      0.204253  17761.191942        72520.961277  100627.643160
std       0.403211   9317.174534        40421.425247   50600.276834
min       0.000000   1300.000000         2619.000000    8000.000000
25%       0.000000  11000.000000        46743.250000   66663.500000
50%       0.000000  16000.000000        65062.500000   89000.000000
75%       0.000000  22800.000000        90105.750000  119397.750000
max       1.000000  53800.000000       256431.000000  324987.000000

       Nb_annees_travail  Nb_report_pret  Nb_litiges     Age_cred  \
count        3574.000000     3574.000000  3574.000000  3574.000000
mean            8.930381        0.269026     0.509233    14.847147
std             7.302566        0.851860     1.118736     6.553137
min             0.000000        0.000000     0.000000     0.000000
25%             3.000000        0.000000     0.000000     9.760000
50%             7.000000        0.000000     0.000000    14.400000
75%            13.000000        0.000000     1.000000    19.007500
max            41.000000       10.000000    15.000000    40.500000

       Nb_demandes_cred  Ratio_dette_revenu
count       3574.000000         3574.000000
mean           1.189097           33.746783
std            1.654805            6.307040
min            0.000000           17.126000
25%            0.000000           29.601000
50%            1.000000           34.027583
75%            2.000000           38.374750
max           17.000000           72.670000
```

FIGURE 2.10 – ↑ Variables quantitatives **après** imputation

corr - DataFrame

Index	Incident_r	Montant_pret	Montant_hypotheque	Val_propriete	Nb_annees_travail	Nb_report_pret	Nb_litiges	Age_cred	Nb_demandes_cred	Ratio_dette_revenu
Incident_r	1	-0.0839609	-0.0593934	-0.0855912	-0.0506662	0.271619	0.319614	-0.016…	0.165793	0.0715971
Montant_pret	-0.0839…	1	0.248462	0.306023	0.0577532	0.0205462	-0.032…	-0.110…	0.0657885	0.18428
Montant_hypotheque	-0.0593…	0.248462	1	0.825834	-0.0532804	-0.0658541	0.0182…	-0.596…	0.0149109	0.115298
Val_propriete	-0.0855…	0.306023	0.825834	1	0.0151465	-0.0574989	-0.024…	-0.656…	-0.0224158	0.110178
Nb_annees_travail	-0.0506…	0.0577532	-0.0532804	0.0151465	1	-0.0569872	0.0405…	0.0446…	-0.0762192	-0.0665271
Nb_report_pret	0.271619	0.0205462	-0.0658541	-0.0574989	-0.0569872	1	0.181267	0.0311…	0.19091	0.00208589
Nb_litiges	0.319614	-0.032094	0.0182824	-0.0242465	0.0405334	0.181267	1	0.0027…	0.057789	-0.0109418
Age_cred	-0.0162…	-0.11052	-0.596698	-0.656888	0.0446097	0.0311561	0.0027…	1	-0.0141968	-0.0300718
Nb_demandes_cred	0.165793	0.0657885	0.0149109	-0.0224158	-0.0762192	0.19091	0.057789	-0.014…	1	0.128205
Ratio_dette_revenu	0.07159…	0.18428	0.115298	0.110178	-0.0665271	0.00208589	-0.010…	-0.030…	0.128205	1

FIGURE 2.11 – ↑ Matrice de Corrélation **après** Imputation

Imputation des variables qualitatives :

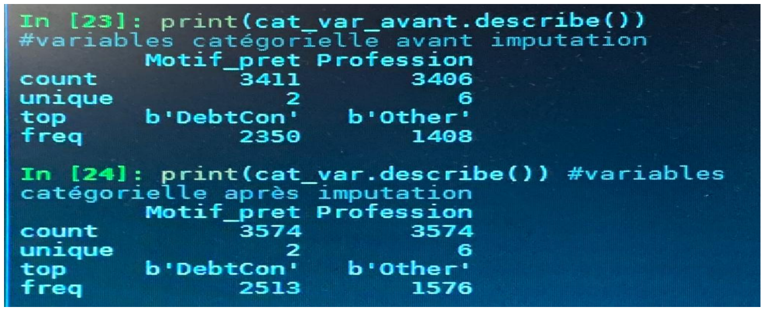


FIGURE 2.12 – ↑ variables qualitative **avant après** Imputation

2.4 Création des bases apprentissage et test

Après le traitement de notre base de données, il est fondamental de la diviser en deux échantillons, respectivement en échantillon d'**apprentissage** et en échantillon **test** pour estimer nos modèles. Le **1er** nous servira pour la construction du modèle alors que le **2nd** nous permettra de valider le modèle.
Il est nécessaire de constituer ces 2 échantillons de façon à avoir la même probabilité de défaut que celle de la base de données, à savoir une même proportion de défaillance. Ces échantillons doivent nous fournir une représentation claire de la population dans sa globalité.
Nous effectuerons un tirage simple aléatoire sans remise ou nous prendrons en compte un échantillon d'apprentissage de l'ordre de 80% et un échantillon test de 20%.

Afin d'effectuer des transformations sur nos données, nous utilisons la fonction train_test_split() de **scikit-learn**.

Chapitre 3

Estimation des modèles

3.1 Classification par les k plus proches voisins

La méthode des K plus proches voisins ou K-nearest neighbors (kNN) est un algorithme appartenant à la classe des algorithmes d'apprentissages supervisés qui est facile à mettre en œuvre et dont l'usage est primordial pour résoudre les problèmes de classification ainsi que de régression. Il est facile à mettre en application et ne nécessite en aucun cas l'élaboration de modèles ou la formulation d'hypothèses supplémentaires. L'inconvénient reste néanmoins un ralentissement de la procédure lorsque le nombre d'observations et de variables indépendantes augmente.

Il faut souligner ici que le nombre de voisins k n'est pas nécessairement intuitif. Surtout lorsque l'ensemble de données est volumineux. De même, il n'y a pas de nombre unique k possible. Cependant on choisira le meilleur, c'est à dire celui pour lequel le k sera le plus petit.

— L'algorithme calcule les distances entre notre individu io et les individus i.
— Par la suite, les distances sont rangées par ordre croissant et sont retenus uniquement les individus qui sont les k-plus proches voisins de notre individu io.
— Enfin l'algorithme affecte l'individu io à la classe majoritaire de ces k voisins.

Le choix de l'entier k est donc primordial, selon le choix du k, les résultats peuvent différer. L'entier k le plus efficace se fait en minimisant l'erreur de classement calculé par validation croisée.
Nous cherchons donc le k qui nous fournit le taux d'erreur le plus **faible**.

En faisant un balayage avec pour valeur de départ k=3, et en la faisant varier jusqu'à k=16, nous trouvons que le k-optimal est celui égal à (k=16), car c'est lui qui nous fournit le taux d'erreur le plus faible à l'issue de validation croisée sur l'échantillon test.

Incident_r	**0**	**1**
0	547	14
1	134	20

TABLE 3.1 – Matrice de Confusion des **K-NN**

Le taux d'erreur moyen estime la probabilité de classer un individu pris au hasard dans l'échantillon lorsqu'on fait le modèle de prédiction. On trouve que $\tau = (134 + 14)/715 = 0.206$, soit 20.6%

3.2 Arbre de décision

L'arbre de décision est une méthode de prise de décision qui fait partie de l'une des plus efficaces. Elle permet non seulement de présenter visuellement les informations mais aussi de les hiérarchiser. C'est un outil qui facilite grandement nos décisions et limite le sentiment de surcharge informationnelle.

Un arbre de décision commence généralement par un nœud d'où découlent plusieurs résultats possibles. Chacun de ces résultats mènent à d'autres nœuds, d'où émanent d'autres possibilités. Le schéma ainsi obtenu rappelle la forme d'un arbre.

En utilisant la fonction *DecisionTreeClassifier* du package **sklearn.tree**, nous pouvons générer l'arbre de décision. Après estimation de ce modèle, l'erreur de prédiction est de :

Incident_r	**0**	**1**
0	547	14
1	134	20

Table 3.2 – Matrice de Confusion de l'**arbre de décision**

3.3 Régression logistique

Ce modèle nous permettra de déterminer la probabilité qu'un client soit solvable, au travers de sa fonction « Logit » (sous python).
La régression logistique est un modèle de régression multiple. Elle est utilisée lorsque la variable à expliquer (variable dépendante Y) est qualitative, le plus souvent binaire. La variable explicative (variable indépendante Xi) peut être qualitative ou quantitatif. Dans notre cas, la variable à expliquer est la personne qui n'a pas remboursé le prêt, évènement "valeur 0" ou événement "valeur 1" (personne ayant remboursé le prêt). La régression logistique s'avère être, par ailleurs l'une des méthodes fiables de modélisation, où plusieurs indicateurs statistiques permettent vérifiez facilement sa robustesse (rapport LR, R-carré). En pratique, cette approche ne nécessite pas nécessairement une distribution normale des variables, ni homogénéité de la variance. Cependant, avoir un grand échantillon est important.

La prédiction de notre variable d'intérêt qui est une variable binaire se prête à une régression logistique. Les versions pénalisées (Ridge, Lasso, elastic net, lars) du modèle linéaire général sont les algorithmes les plus développés dans scikit-learn au détriment de ceux plus classique de sélection de variables. Nous avons utilisé la version Lasso de la régression logistique qui introduit la sélection automatique des variables.
Les packages **sklearn.linear_model** et **sklearn.model_selection** sont les deux utilisés.

Les résultats de l'estimation nous permettent d'avoir les interceptes et les coefficients suivant :

En utilisant la fonction *metrics* du package **sklearn**, le taux de bonne prédiction du modèle estimé est de **79%**. La probabilité que le modèle se trompe dans sa prédiction est d'environ **21%** comme le présente l'image suivante :

```
In [22]: succes = metrics.accuracy_score(Y_test,Y_pred)
    ...: print(succes)
0.793006993006993

In [23]: err = 1.0 - succes
    ...: print(err)
0.20699300699300704
```

FIGURE 3.1 – Taux d'erreur **Régression Logistique**

3.4 Forêt aléatoire

Les forêts aléatoires sont une combinaison de prédicteurs d'arbres de sorte que chaque arbre dépend des valeurs d'un vecteur aléatoire échantillonné indépendamment et avec la même distribution pour tous les arbres de la forêt. Leur construction est basée sur des algorithmes permettant d'optimiser un critère afin d'obtenir des séparations les plus « pures » possibles. L'apprentissage par arbre de décision consiste en l'apprentissage de l'arbre en utilisant des données. Chaque branche est créée de manière à séparer le mieux possibles les différentes modalités de la cible de votre apprentissage supervisé.

On fait usage d'une sélection aléatoire de caractéristiques pour diviser chaque nœud. Cette sélection produit des taux d'erreur qui se comparent favorablement à ceux d'Adaboost **(Freund et Schapire [1996])**, mais sont plus robustes en ce qui le concerne par rapport au bruit.
Par ailleurs, les estimations internes permettent de contrôler l'erreur, la force et la corrélation et celles-ci sont utilisées pour montrer la réponse à l'augmentation du nombre de caractéristiques utilisées dans le fractionnement. Elles sont également utilisées pour mesurer l'importance des variables. Notons également que l'erreur de généralisation des forêts converge vers une limite lorsque le nombre d'arbres de la forêt devient important. Cet erreur d'arbres classificateurs dépend de la force des arbres individuels de la forêt et de la corrélation entre eux.
Pour remarque, ces idées sont également applicables à la régression.

Dans le cadre de ce projet, l'estimation de ce modèle prend en compte les paramètres suivants :
- 500 arbres (n_estimators)
- 5 division au minimum (min_samples_split)
- la méthode d'estimation de l'erreur considéré est celle standard : out-of-bag.

Sous python, cette estimation fût possible grâce à la fonction *RandomForestClassifier* du package **sklearn.ensemble**. Après avoir construit le premier jet du modèle avec les données de l'échantillon d'apprentissage, l'erreur de prévision sur l'échantillon test est d'environ **12%**.

L'élaboration de la matrice de confusion, grâce à la fonction ***crosstab*** du package **pandas** nous permet d'avoir le résultat suivant :

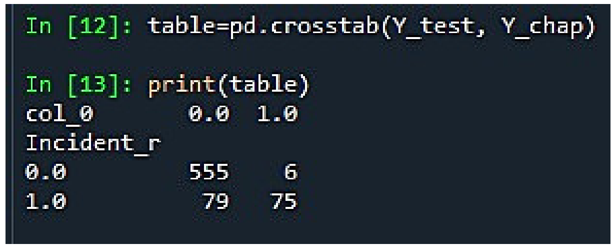


FIGURE 3.2 – Matrice de confusion du **Random Forest**

Le taux d'erreur moyen obtenu est égale à $\tau = (79 + 6)/715 = 0.118$
Ainsi la probabilité de classer un individu pris au hasard hors l'échantillon, une fois l'estimation du modèle de prédiction faite est de 11.8%.

3.5 Analyse du meilleur modèle

Model	**Error ratio (%)**
KNN-Neighbors	20.6
Logistic Regression	20.6
Decision Tree	16.6
Random Forest	11.8

TABLE 3.3 – Tableau de comparaison des taux d'erreurs des modèles

Après comparaison, nous remarquons que la méthode qui fournit le taux d'erreur le plus faible est la méthode des Forêts aléatoires avec un taux de **11.8%**.
Donc modèle *(Random Forest)* est le meilleur modèle.

Chapitre 4

Scoring

Il a pour objectif d'attribuer des notes (ou scores) aux emprunteurs potentiels pour estimer les performances futures de son emprunt en vue de lui accorder un crédit et d'assurer sa solvabilité appelé le ***score de risque***.

L'un des avantages de cette approche est qu'elle augmente la productivité. En effet, les décideurs se basent sur ces scores dans un premier temps, scores calculés sur la base des informations des clients. Un temps considérable est économisé. Dans un second temps, le décideur peut faire recours aux dossiers des clients, pour s'assurer de sa décision, ou lorsque l'un des clients fait une réclamations.

Dans le cadre de notre projet, le score des clients a été calculé à la suite de l'estimation du modèle par la régression logistique. De ce fait, le score maximal d'un client est de **0.68**.

Conclusion

La réalisation de ce projet était pour nous l'occasion de découvrir le puissant logiciel d'analyse de données qu'est python. Cet outil est un langage très couramment utilisé par les Data Scientists, nous avons vu pourquoi nous devons remercier ses différentes bibliothèques.
Il est probablement en raison de son écosystème riche et de sa simplicité, l'apprentissage des language de programmation le plus important à utiliser dans l'avenir.

Le principal avantage de Python est sa largeur. Par exemple, R peut exécuter l'algorithme Apprentissage automatique sur des ensembles de données prétraités, mais Python est meilleur pour le traitement de l'information.
Pandas est une bibliothèque très utile qui peut essentiellement faire tout ce que SQL fait et plus encore. En termes d'utilisabilité de l'algorithme. Il existe de nombreux algorithmes prêts à l'emploi disponibles via **scikit-learn**.
De plus, avec bibliothèque scikit-learn, tout est prêt pour que les utilisateurs puissent travailler dans la mise en place de leur modèle.

Le *«risque»* s'applique au secteur bancaire et peut prédire l'insolvabilité des clients.
Utiliser ces scores pour établir des règles de prise de décision. Sur la base des données disponibles, nous avons appliqué les différentes méthodes statistiques vues dans le cours, à savoir la régression logistique qui calcule un score de risque pour chaque client.

Bibliographie

[1] https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

[3] https://scikit-learn.org/stable/modules/model_evaluation.html.

[4] https://www.stat4decision.com/fr/faire-une-regression-logistique-avec-python/.

[5] http://www.xavierdupre.fr/app/ensae_teaching_cs/helpsphinx/notebooks/td2a_cenonce_session_3B.html.

Annexe A

Annexes

Variables	**Effectif**	**Pourcentage** (%)
Incident_r	0	0
Montant_pret	0	0
Montant_hypotheque	309	8.65
Val_propriete	68	1.90
Motif_pret	163	4.56
Profession	168	4.70
Nb_annees_travail	306	8.56
Nb_report_pret	431	12.06
Nb_litiges	354	9.90
Age_cred	183	5.12
Nb_demandes_cred	309	8.65
Ratio_dette_revenu	777	21.74

TABLE A.1 – *Tableau de fréquences de valeurs manquantes*

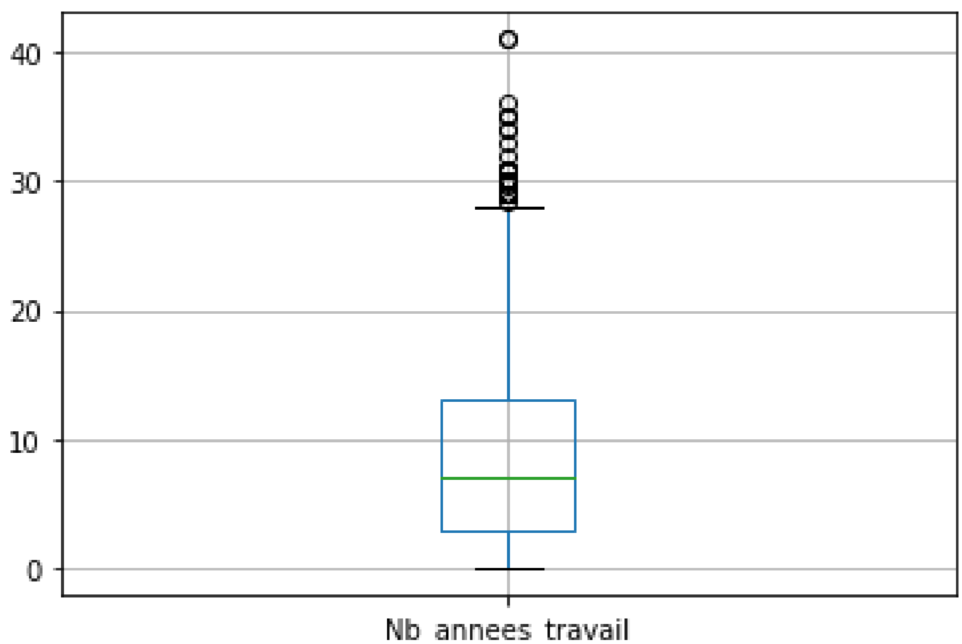


FIGURE A.1 – ↑ Boîte à Moustache *Nb_ annees_ travail*

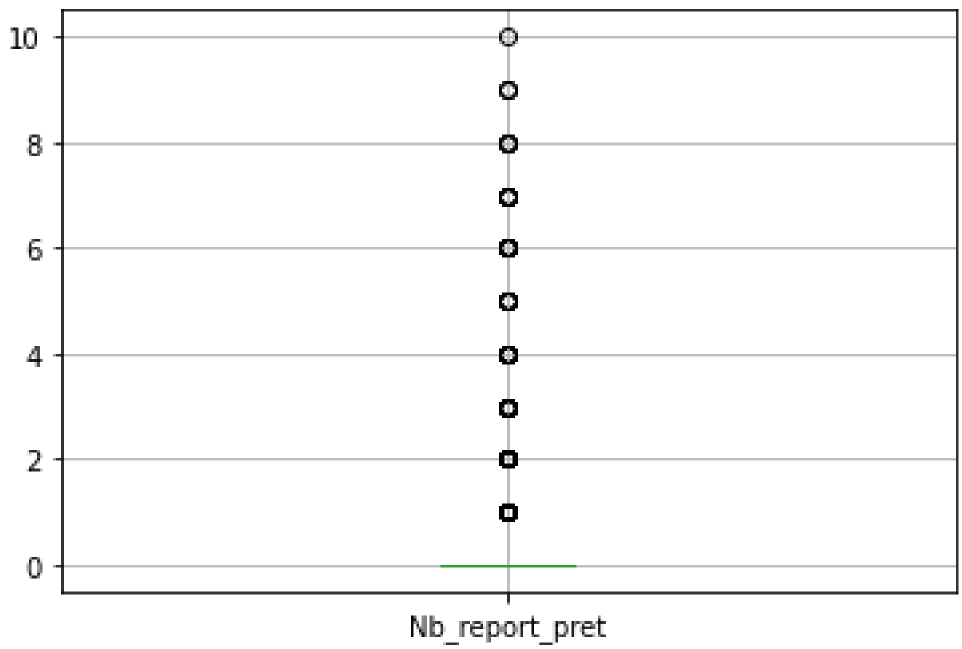


Figure A.2 – ↑ Boîte à moustache *Nb_ report_ pret*

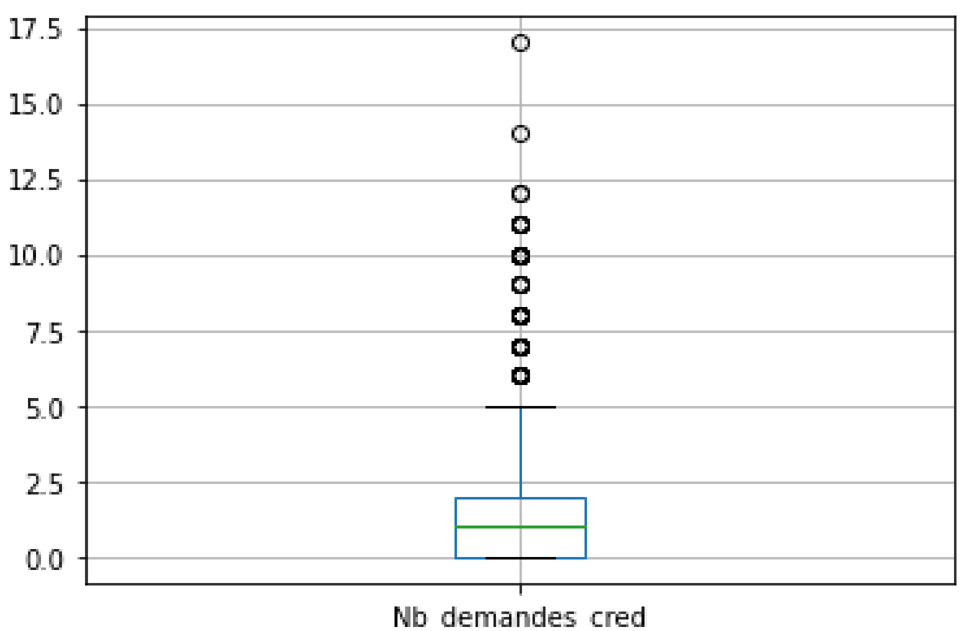


Figure A.3 – ↑ Boîte à moustache *Nb_ demandes_ cred*

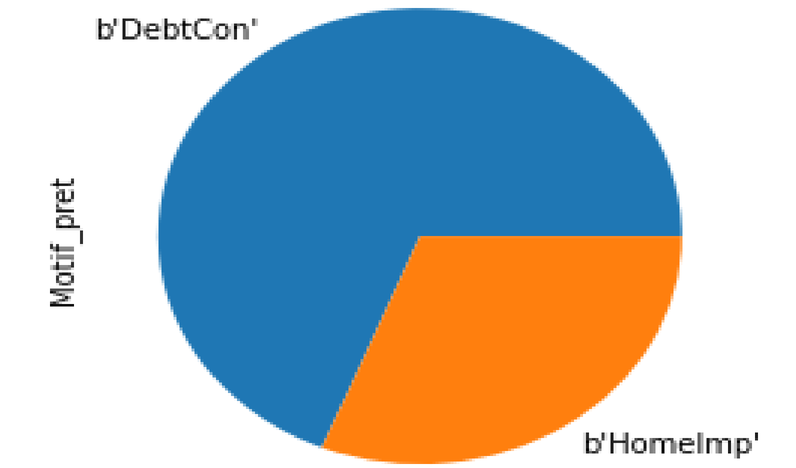


FIGURE A.4 – ↑ Diagramme en camembert *Motif_pret*

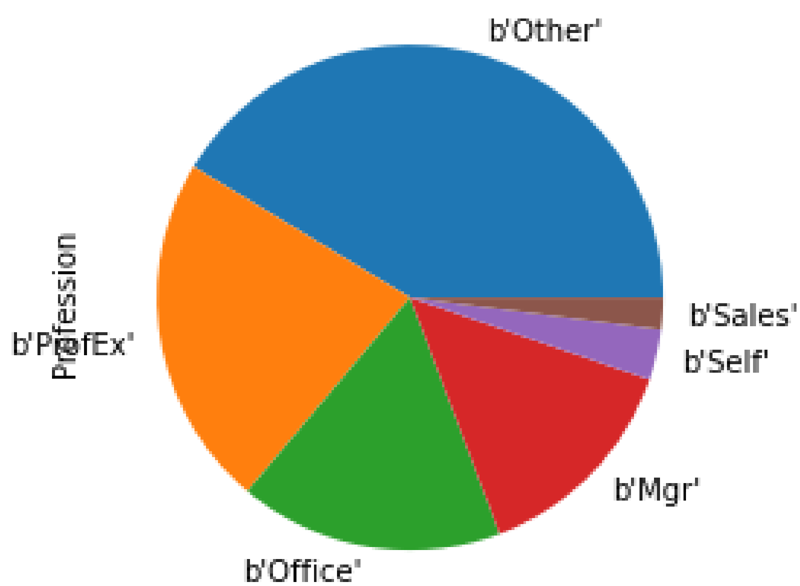


FIGURE A.5 – ↑ Diagramme en camembert *Profession*

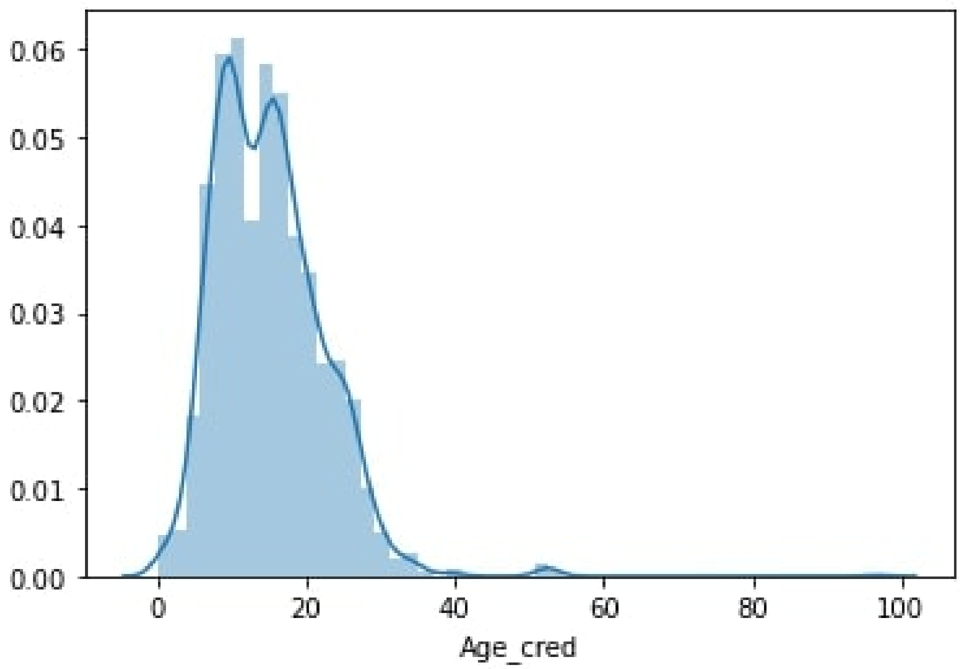


FIGURE A.6 – ↑ Densité de probabilités *Age_ cred*

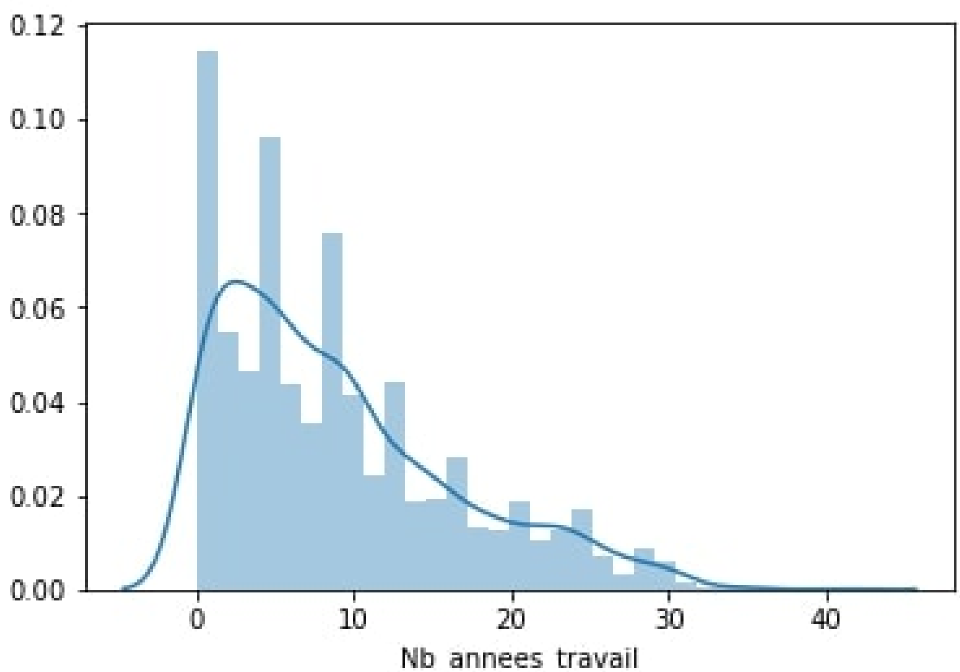


FIGURE A.7 – ↑ Densité de probabilités *Nb_ annees_ travail*

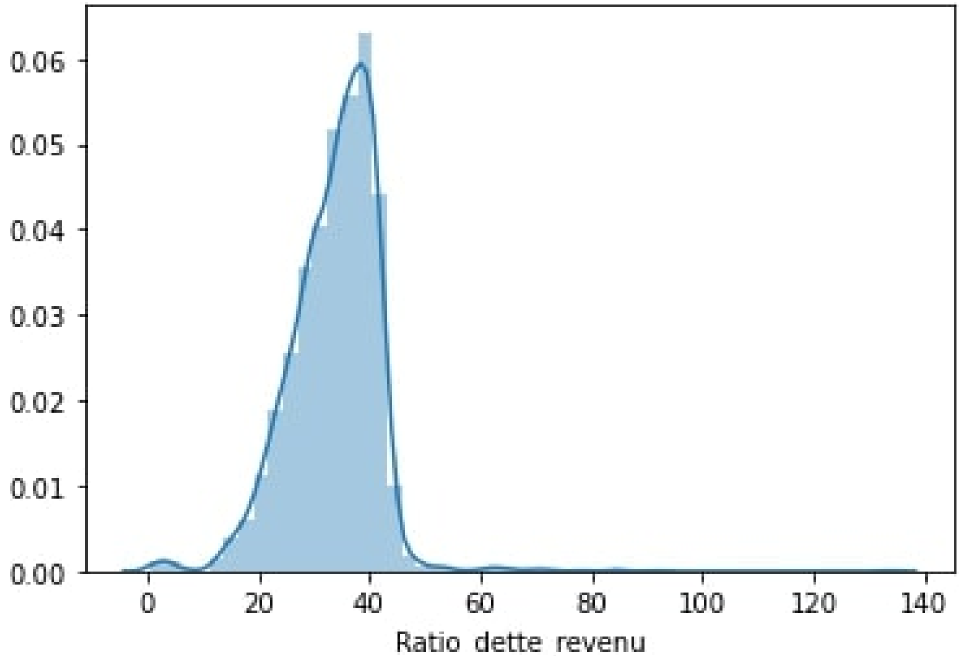


FIGURE A.8 – ↑ Densité de probabilités *Ratio_ dette_ revenu*

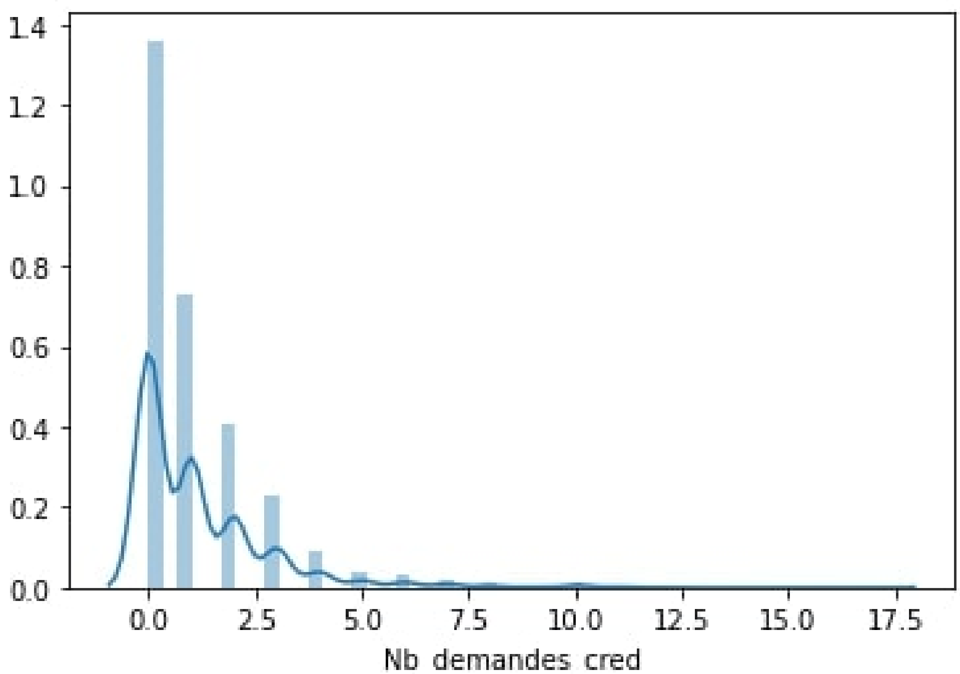


FIGURE A.9 – ↑ Densité de probabilités *Nb_ demandes_ cred*

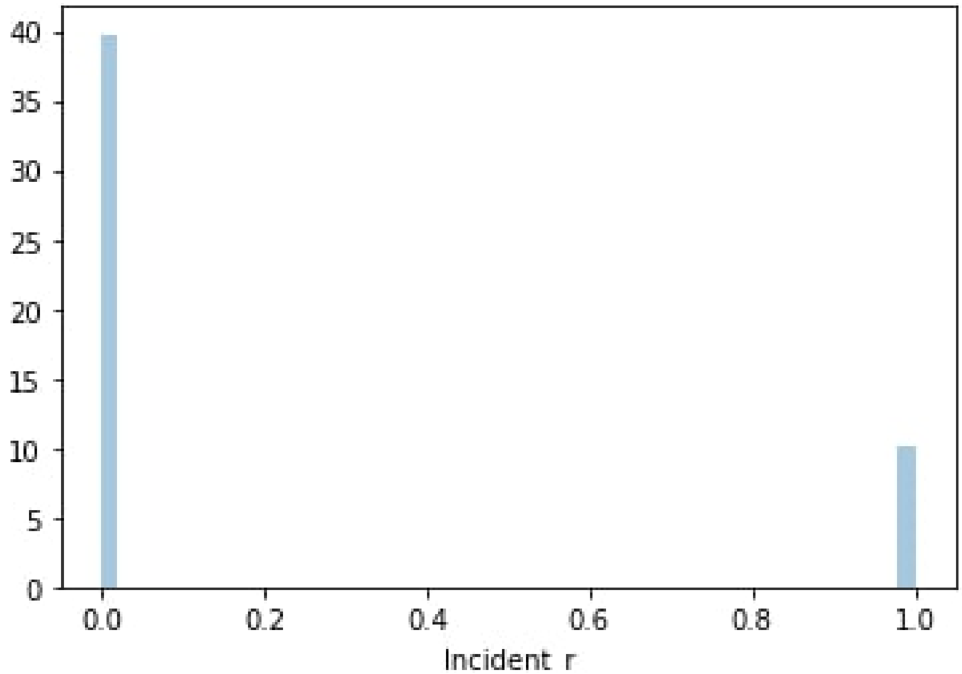


FIGURE A.10 – ↑ Densité de probabilités *Incident_ r*

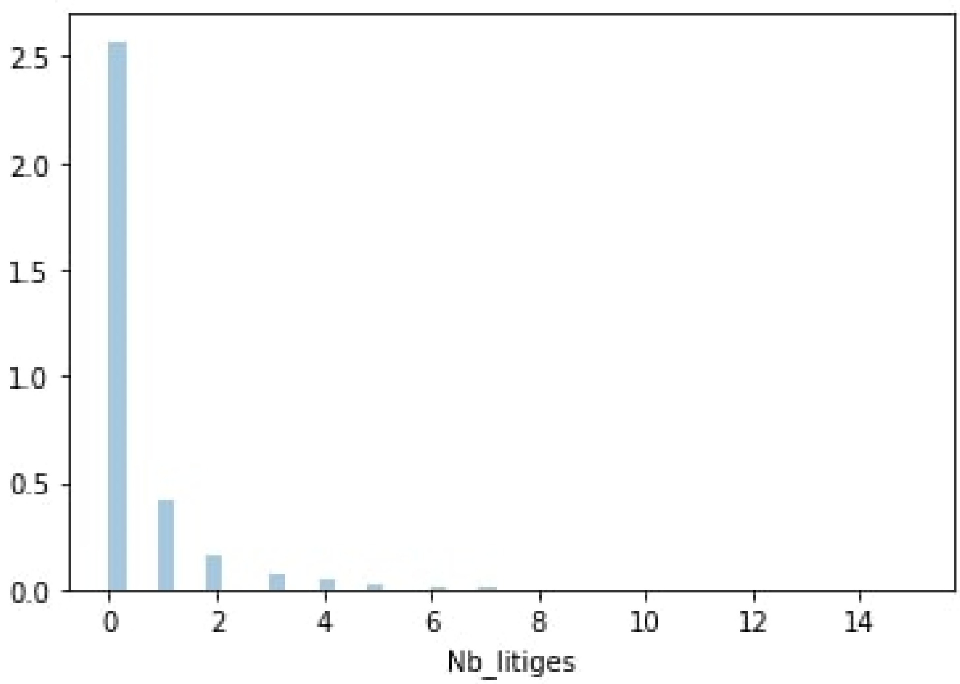
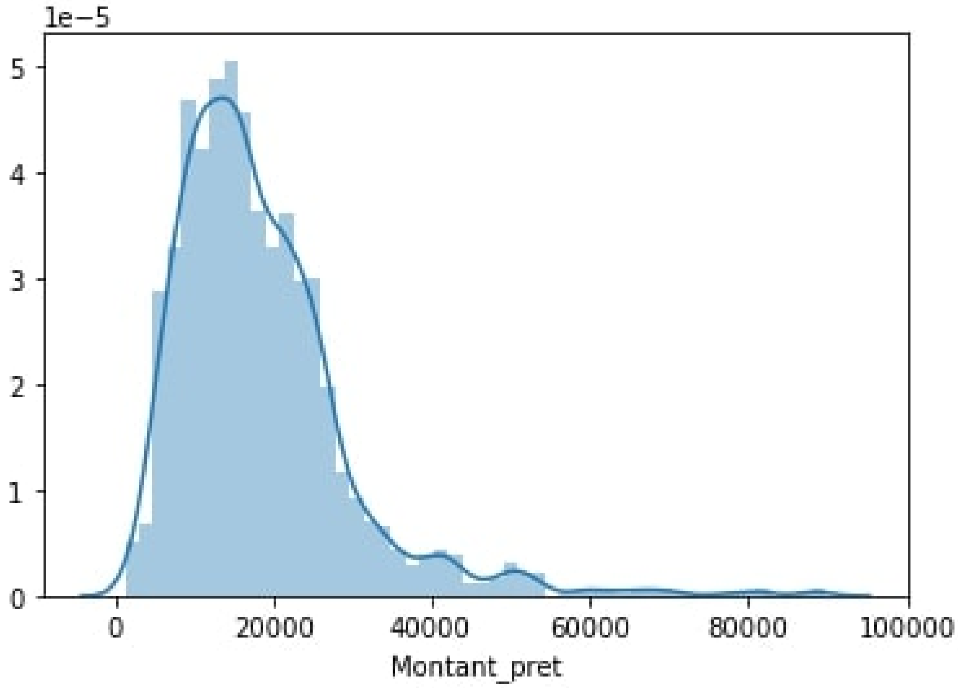
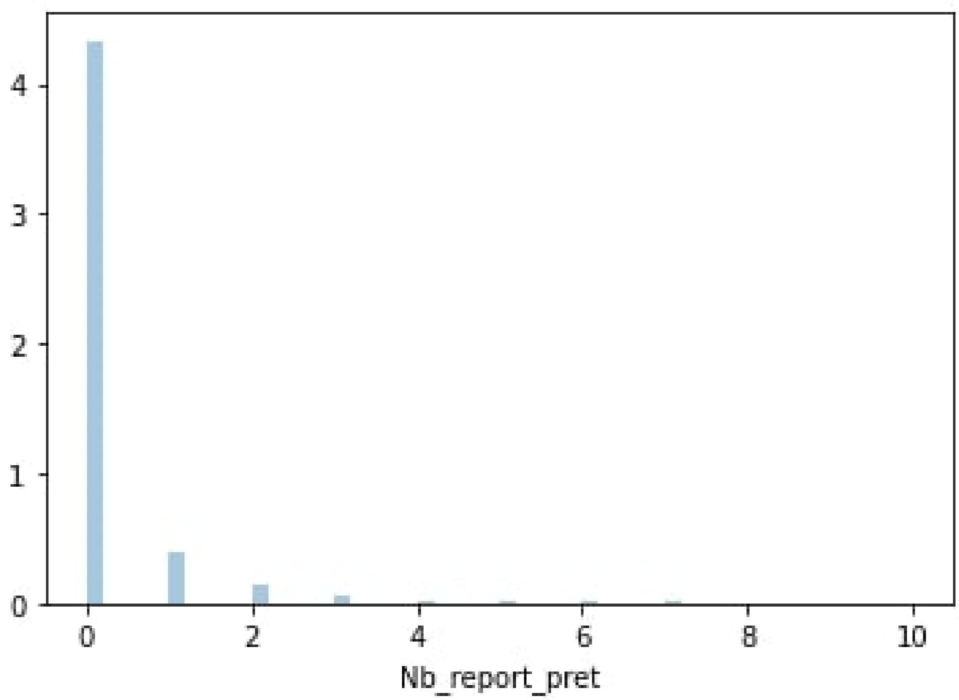


FIGURE A.11 – ↑ Densité de probabilités *Nb_ litiges*



1e−5
5
4
3
2
1
0
0
20000
40000
60000
80000
100000
Montant_pret



4
3
2
1
0
0
2
4
6
8
10
Nb_report_pret

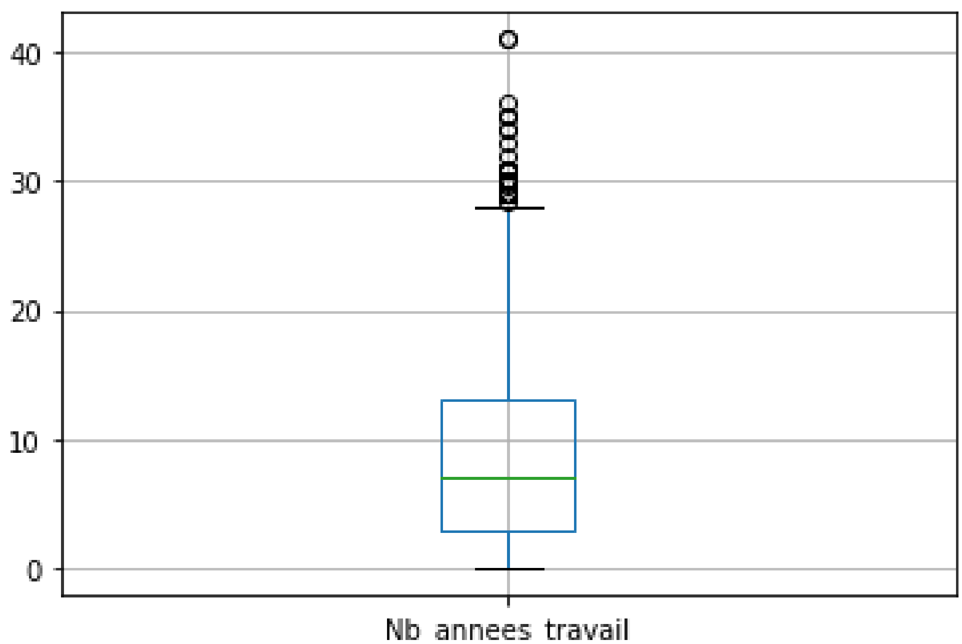


Figure A.12 – ↑ Boîte à Moustache *Nb_ annees_ travail*

Annexe B

Code Python

Pandas : est une librairie Python permettant la manipulation et l'analyse de données. Elle propose en particulier des structures de données et des opérations de manipulation de tableaux numériques et de séries temporelles.

NumPy : est une librairie qui permet de manipuler des matrices ou tableaux multidimensionnels et aussi des fonctions mathématiques qui agissent sur ces tableaux

Os : est dédié aux besoins de gestion de fichiers et de dossiers

Scipy : a pour objectif d'unifier et fédérer un ensemble de bibliothèques Python à usage scientifique. SciPy utilise les tableaux et matrices du module NumPy.

Matplotlib : est une librairie de Python permettant de tracer et visualiser des données sous formes de graphiques. Elle peut-être combinée avec les bibliothèques Python de calcul scientifique qui sont NumPy et SciPy.

Seaborn : vient s'ajouter à Matplotlib, remplace certains réglages par défaut et fonctions, et lui ajoute de nouvelles fonctionnalités. Ainsi toutes ces fonctions énumérées vont être utilisées pour l'exploration de données ainsi que d'autres si nécessaires.

```
[language=Python]

/**************************************************************************************/
/*                                Script Projet Python                                */
/**************************************************************************************/
# -*- coding: utf-8 -*-
"""
APPLICATION : PROJET DE PYTHON

@author: SYLLA Daya Mamadou
         DJRAMEDO Chris
         YUZUAK Ali
"""

#####################################
```

```
##        Liste des variable        ##

# VarA = Incident_r_r : (qualitative)
# VarE = Motif_pret : (qualitative)
# VarF = Profession : (qualitative)
# VarB = Montant_pret
# VarC = Montant_hypotheque
# VarD = Val_propriete
# VarG = Nb_annees_travail
# VarH = Nb_report_pret
# VarI = Nb_litiges
# VarJ = Age_cred
# Vark = Nb_demandes_cred
# VarL = Ratio_dette_revenu
######################################


import os
import numpy as np
import pandas as pd
import scipy as sc
import matplotlib.pyplot as plt
import seaborn as sns
import pylab as pl

from os import chdir
from sklearn.impute import KNNImputer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

#-----------------------------------------------------------------------
# ### Chargement de la base de données ######
# chdir(r"C:\Users\e-msylla1\Downloads\Projet")
# pd.set_option('display.max_column', 12)
# dfp = pd.read_sas(r"C:\Users\e-msylla1\Downloads\Projet/Tab2.sas7bdat")
#-----------------------------------------------------------------------

### Chargement de la base de données ######
chdir(r"C:\Users\carrel\Downloads\Projet")   # work directory
pd.set_option('display.max_column', 12)
dfp = pd.read_sas(r"C:\Users\carrel\Downloads\Projet/Tab2.sas7bdat")


#1)
# Renommage des variables
```

```
dfp.rename(columns={'varA': 'Incident_r', 'varB': 'Montant_pret',
'varC': 'Montant_hypotheque',
'varD': 'Val_propriete','varE': 'Motif_pret','varF': 'Profession',
'varG': 'Nb_annees_travail','varH': 'Nb_report_pret', 'varI': 'Nb_litiges',
'varJ': 'Age_cred','varK': 'Nb_demandes_cred','varL': 'Ratio_dette_revenu'},
inplace=True)
print(dfp)


dfp.shape       #(Nb_ligne, Nb_col)
dfp.dtypes      #(Types des variables)

#compte le nombre d'occurence de la variable Incident_r à prédire
dfp["Incident_r"].value_counts(dropna = False)

#compte le nombre d'occurence des variables qualitatives
dfp["Motif_pret"].value_counts(dropna = False)
dfp["Profession"].value_counts(dropna = False)

## Conversion de l'age_cred den annees
dfp['Age_cred'] = round(dfp['Age_cred'] /12 ,2)
dfp.describe(include="all")

# variables qualitatives
for col in dfp.select_dtypes('object'):
    print(f'{col :-<30} {dfp[col].unique()}')

for col in dfp.select_dtypes('object'):
    plt.figure()
    dfp[col].value_counts().plot.pie()

########## Semble pas utile pour le moment ###########
### Mise en forme des variables des variables avant le
### lancement de la matrice de correlation

sns.countplot(x='Montant_pret', hue='Incident_r', data=dfp)
sns.countplot(x='Ratio_dette_revenu', hue='Incident_r', data=dfp)
sns.countplot(x='Val_propriete', hue='Incident_r', data=dfp)
sns.countplot(x='Nb_report_pret', hue='Incident_r', data=dfp)
sns.countplot(x='Nb_litiges', hue='Incident_r', data=dfp)
sns.countplot(x='Age_cred', hue='Incident_r', data=dfp)
sns.countplot(x='Nb_demandes_cred', hue='Incident_r', data=dfp)
sns.countplot(x='Ratio_dette_revenu', hue='Incident_r', data=dfp)
sns.countplot(x='Montant_hypotheque', hue='Incident_r', data=dfp)

##### Recherche des valeurs manquantes #####
dfp.isna().any() # NA oui ou non
```

```
dfp.isna().sum() # comptage des NA

# Calcul en pourcentage des valeurs manquantes de chaque variable
dfp_Na=pd.DataFrame({"Pourcentage_Na" : round(dfp.isnull().sum()/(dfp.shape[0])*100,2)})
dfp_Na


####  Histogramme des valeurs manquantes ####
List_var=('Incident_r','Montant_pret','Montant_hypotheque','Val_propriete','Motif_pret',
            'Profession','Nb_annees_travail','Nb_report_pret','Nb_litiges','Age_cred',
            'Nb_demandes_cred','Ratio_dette_revenu')

List_value=[0,0,8.65,1.90,4.56,4.70,8.56,12.06,9.90,5.12,8.65,21.74]
y_pos=np.arange(len(List_var))

plt.bar(y_pos,List_value)
plt.xticks(y_pos,List_var,rotation=90)
plt.ylabel('val. manquantes (%)')
plt.subplots_adjust(bottom=0.4,top=0.99)
plt.show()

#################################
# DETECTER DES VALEURS ABERRANTES
# Histogramme des variables continues
for col in dfp.select_dtypes('float'):
   plt.figure()
   sns.distplot(dfp[col])
# variables qualitatives
for col in dfp.select_dtypes('object'):
    print(f'{col :-<30} {dfp[col].unique()}') #systeme de marge


# Diagrammes en moustache des 9 variables quantitatives
dfp.boxplot(column='Montant_pret')
dfp.boxplot(column='Ratio_dette_revenu') # <17 et >75
dfp.boxplot(column='Age_cred') # >50
dfp.boxplot(column='Nb_litiges')
dfp.boxplot(column='Nb_demandes_cred')
dfp.boxplot(column='Nb_report_pret')
dfp.boxplot(column='Nb_annees_travail')
dfp.boxplot(column='Val_propriete') # >400000
dfp.boxplot(column='Montant_pret') # >55000
dfp.boxplot(column='Montant_hypotheque') # 270000


##########################################
#TRAITEMENT : VALEURS MANQUANTES ET ABBERABTES
```

```
##Statistiques desc avant imputation des var qualitatives
cat_var_avant=dfp[['Motif_pret', 'Profession']]
print(cat_var_avant.describe())   #variables catégorielle avant imputation
## On voit clairement que nous avons des manquantes manquantes
## au niveau de ces variables, car nous avons un total de 3750 individus

##Statistiques desc avant imputation des var quantitives
print(dfp.describe())



######### Imputations #######
### VARIABLES CATEGORIELLES : imputation par le mode
cat_var = dfp[['Motif_pret', 'Profession']]
cat_var['Motif_pret'] = cat_var['Motif_pret'].fillna(cat_var['Motif_pret'].mode()[0])
cat_var['Profession'] = cat_var['Profession'].fillna(cat_var['Profession'].mode()[0])
print(cat_var.describe()) #variables catégorielle après imputation

## Autrement en application
dfp['Motif_pret']= cat_var['Motif_pret'].fillna(cat_var['Motif_pret'].mode()[0])
dfp['Profession']=cat_var['Profession'].fillna(cat_var['Profession'].mode()[0])
print(dfp['Profession'])


# reVérifier s'il y a des
cat_var.isna().any()
## Nous voyons clairement que toutes les variables ont été imputé avec succès


### VARIABLES CONTINUES : imputation par le kNN
quant_var = dfp[['Incident_r', 'Montant_pret', 'Montant_hypotheque', 'Val_propriete',
                 'Nb_annees_travail', 'Nb_report_pret', 'Nb_litiges', 'Age_cred',
                 'Nb_demandes_cred', 'Ratio_dette_revenu']]


# Transformation des valeurs abberantes en manquantes

u=quant_var.Ratio_dette_revenu
for i in range(len(u)):
 if u[i] > 75 or u[i] < 17 : u[i]= 'NaN'
print(u)

#variable Val_propriete
u=quant_var.Val_propriete
for i in range(len(u)):
 if u[i] > 400000 : u[i]= 'NaN'
```

```
#variable Age_cred
u=quant_var.Age_cred
for i in range(len(u)):
 if u[i] > 50 : u[i]= 'NaN'

#variable Montant_pret
u=quant_var.Montant_pret
for i in range(len(u)):
 if u[i] > 55000 : u[i]= 'NaN'

 #variable Montant_hypotheque
u=quant_var.Montant_hypotheque
for i in range(len(u)):
 if u[i] > 270000 : u[i]= 'NaN'


# IDéfinition d'une fonction pour choisir le k-optimal

rmse = lambda y, yhat: np.sqrt(mean_squared_error(y, yhat))

def optimize_k (data, target):
    errors = [] # liste vide qui va contenir les erreurs calculées
    for k in range(1, 20, 1):
        imputer = KNNImputer(n_neighbors=k)
        imputed = imputer.fit_transform(data)
        quant_var_imputed = pd.DataFrame(imputed, columns=quant_var.columns)

        X = quant_var_imputed.drop(target, axis=1)
        y = quant_var_imputed[target]
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
        random_state=42)

        model = RandomForestRegressor()
        model.fit(X_train, y_train)
        preds = model.predict(X_test)
        error = rmse(y_test, preds)
        errors.append({'K': k, 'RMSE': error})
    return errors

# le k-optimal est égal est à 16
k_errors = optimize_k(data=quant_var, target='Incident_r')
k_errors

# Imputation proprement dite
imputer = KNNImputer(n_neighbors=16)
quant_var = pd.DataFrame(imputer.fit_transform(quant_var),columns = quant_var.columns)
```

```
# reVérifier s'il y a des
quant_var.isna().any()


# Base de données totalement imputée
dfp = pd.concat([quant_var, cat_var], axis=1)
dfp.isna().any()


# Transformation des variables catégorielles en dummies
cat_variables = dfp[['Motif_pret', 'Profession']]
cat_dummies = pd.get_dummies(cat_variables, drop_first=True)


# Ajout des dummies à la base initiale
dfp2 = dfp
dfp2 = dfp.drop(['Motif_pret', 'Profession'], axis=1)
dfp2 = pd.concat([dfp2, cat_dummies], axis=1)


# ############# Matrice de correlation ###########
# Matrice de corrélation
corr = dfp2.corr()
corr.style.background_gradient(cmap='coolwarm')
corr.style.background_gradient(cmap='coolwarm').set_precision(2)

# ### Commentaire :
# # La matrice de coorelation montre une forte correlation entre
# # la valeur_propriété et  montant_hypothèque s 87,92%.
# # Donc nous allons supprimer le montant de l'hypothèque
##########################################################


# Supression de la colonne du Montant_hypotheque
dfp2.drop(['Montant_hypotheque'], axis='columns', inplace=True)
dfp2.head()


#################################################################
# ------------------------ MODELISATION  -----------------------
#################################################################

# SEPARATION DE la variable DEPENDANTE (Y) DES EXPLICATIVES (X)
dfp2=dfp

# Matrice des explicatives (X)
X = dfp2.iloc[:,1:14]
```

```
print(X)

# Matrice de la dépendante (Incident_r)
Y = dfp2.iloc[:, 0]
print(Y)


# CONSTRUCTION DES DEUX ECHANTILLONS (Apprentissage et test)
from sklearn import model_selection
X_app, X_test, Y_app, Y_test = model_selection.train_test_split(X, Y,
test_size = 0.2, random_state=10)
print(X_app.shape, X_test.shape, Y_app.shape, Y_test.shape)
## Echantillon Aprentissage: 80 %
## Echantillon Test: 20 %


##########################################
# MODELE DE REGRESSION LOGISTIQUE
##########################################
# on importe LogisticRegression
from sklearn.linear_model import LogisticRegression
logit = LogisticRegression()
modele = logit.fit(X_app,Y_app)
# construction du modele sur l'´echantillon d'apprentissage
print(modele.coef_,modele.intercept_) # param`etres du mod`ele

# prediction sur l'échantilllon test
Y_pred = modele.predict(X_test) # prediction sur l'´echantillon test

# taux de bonne prédiction
from sklearn import metrics
succes = metrics.accuracy_score(Y_test,Y_pred)
print(succes)

# taux d'erreur
err = 1.0 - succes
print(err)
## taux_erre=0.206


##########################################
#           MODELE DE k-NN
##########################################
# Importation du package
import sklearn
from sklearn import neighbors, metrics
```

```
# Fixer les valeurs des hyperparamètres à tester
param_grid = {'n_neighbors':list(range(1,16))}

# Choisir un score à optimiser, ici l'accuracy
# (proportion de prédictions correctes)
score = 'accuracy'

# Créer un classifieur kNN avec recherche d'hyperparamètre par
#validation croisée
knn = model_selection.GridSearchCV(
    neighbors.KNeighborsClassifier(), # un classifieur kNN
    param_grid,     # hyperparamètres à tester
    cv=5,           # nombre de folds de validation croisée
    scoring=score   # score à optimiser
    )

# Optimiser le classifieur sur le jeu d'entraînement
digit_knn=knn.fit(X_app, Y_app)

# # Afficher le(s) hyperparamètre(s) optimaux
digit_knn.best_params_["n_neighbors"]
##le paramètre vaut 11

#Ensuite on procède à l'estimation du modèle avec la valeur
#"optimale" de notre paramètre qui vaut 6
knn = sklearn.neighbors.KNeighborsClassifier(
n_neighbors=digit_knn.best_params_["n_neighbors"])
digit_knn=knn.fit(X_app, Y_app)

# Estimation de l’erreur de prévision
1-digit_knn.score(X_test,Y_test)

# Prévision
Y_chap = digit_knn.predict(X_test)

# Matrice de confusion
mat_conf=pd.crosstab(Y_test, Y_chap)
print(mat_conf)
plt.matshow(mat_conf)
## taux_erre=0.206


##########################################
# MODELE D'ARBRE DE DECISION
##########################################

from sklearn.tree import DecisionTreeClassifier
```

```
dtree = DecisionTreeClassifier(min_samples_split=5)

# définition du modèle de l'arbre de décision
tit_tree = dtree.fit(X_app, Y_app)

# Estimation de l'erreur de prévision
1-tit_tree.score(X_test, Y_test)
## taux_erre=0.166


#Visualisation de l’arbre.
import os
os.environ["PATH"] += os.pathsep + 'C:\Program Files\Graphviz\bin'

#installer (vian Anaconda Prompt) la librairie pydotplus : pip install pydotplus
import pydotplus
from sklearn.tree import export_graphviz

# Représentation de l'arbre
dot_data = export_graphviz(tit_tree, out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf("Incident.pdf")
graph.write_png("Incident.png")
#L’arbre est généré dans un fichier image ainsi qu'un pdf
# à visualiser pour se rende compte



###########################################
# RANDOME FOREST
###########################################

from sklearn.ensemble import RandomForestClassifier

#définition des paramètres
forest = RandomForestClassifier(n_estimators=500, min_samples_split=5,
oob_score=True)

# apprentissage
forest = forest.fit(X_app, Y_app)
print(1-forest.oob_score_) # =0.1252

# erreur de prévision sur le test
1-forest.score(X_test,Y_test)
## taux_erre=0.116

# prévision
```

```
Y_chap = forest.predict(X_test)

# Matrice de confusion
mat_conf=pd.crosstab(Y_test, Y_chap)
print(mat_conf)



##########################################
# SCORING
##########################################

# obtenir les scores
modele = logit.fit(X_app,Y_app)

probas = logit.predict_proba(X_test)
#calcul des probabilités d'affectation sur l'´echantillon test

#score de "Incident"
score = probas[:,1]
print(score)


score = pd.DataFrame(data=score, columns = ['Score'])

X_test1 = pd.DataFrame(data = X_test, columns = (['Montant_pret',
'Val_propriete', 'Nb_annees_travail', 'Nb_report_pret', 'Nb_litiges',
'Age_cred', 'Nb_demandes_cred','Ratio_dette_revenu', 'Motif_habitat',
'Prof_office', 'Prof_other','Prof_Profex', 'Prof_sales']))
Table_score = pd.concat([X_test1, score],axis=1) # Concaténation

Table_score['Score'] = Table_score['Score'].apply(lambda x: np.log(x/(1-x)))
Table_score.sort_values(by=['Score'],inplace=True)


# Construire la grille sur 100
Table_score['Grille100'] = Table_score['Score'].apply(lambda x: 100*(x+max(Table_score['Sc
max(Table_score)


# Segmenter la population en 4 classes
Table_score['Classe_Score']=pd.qcut(Table_score.Grille100, 4,
labels=["Aucun risque", "Très peu risqué", "Risqué", "Très risqué"])
```