



Depositary_Hunt : Prédiction du succès d'une campagne de Marketing d'une banque

Session : Janvier – Avril 2022

Réalisé par :

Nesrine Berkane

Maxime Lasou

David Martinet

SOMMAIRE

1	Introduction	4
1.1	Description du projet.....	4
1.2	Problématique et plan du projet.....	4
1.3	Inventaire des données	5
1.4	Description des différentes variables	5
2	Gestion des données manquantes.....	6
3	Analyse des variables	6
3.1	Analyse des variables catégorielles.....	6
3.1.1	Analyse de la distribution des variables catégorielles.....	7
3.1.2	Analyse des corrélations entre les variables catégorielles et la variable cible.....	11
3.2	Analyse des variables quantitatives.....	13
3.2.1	Analyse de la distribution des variables quantitatives.....	14
3.2.2	Présence de valeurs aberrantes	19
3.2.3	Corrélation et multi-colinéarité.....	21
4	Preprocessing des données	23
4.1	Features engineering	23
4.1.1	Encodage des variables.....	23
4.1.2	Normalisation des données.....	26
4.1.3	Equilibrage des modalités de la variable cible « deposit »	26
4.2	Features selection.....	27
4.2.1	Réduction de dimension par la méthode PCA	27
4.2.2	Sélection de variables selon le modèle LogisticRegression : Wrapper Methods (Recursive Feature Elimination - RFE) et méthodes traditionnelles	27
4.2.3	Sélection de variables par méthodes traditionnelles appliquées aux modèles RandomForestClassifier et GradientBoostingClassifier.....	29
4.2.4	Choix des variables et justification.....	31
5	Modélisation	33
5.1	Choix de la métrique à utiliser	33
5.2	Modèles de classification utilisés.....	34
5.2.1	Premiers modèles de classification sélectionnés.....	35
5.2.2	Modèles de classification définitifs	41
6	Interprétabilité.....	49
6.1	Interprétabilité globale.....	49

6.1.1	Interprétabilité globale du modèle LogisticRegression	49
6.1.2	Interprétabilité globale du modèle RandomForestClassifier	51
6.1.3	Interprétabilité globale du modèle GradientBoostingClassifier	51
6.1.4	Interprétabilité globale du modèle StackingClassifier.....	52
6.2	Dépendance partielle	54
6.2.1	Dépendance partielle de la variable « age » selon le modèle LogisticRegression	54
6.2.2	Dépendance partielle de la variable « age » selon le modèle RandomForestClassifier	55
6.2.3	Dépendance partielle de la variable « age » selon le modèle GradientBoostingClassifier	56
6.2.4	Dépendance partielle de la variable « age » selon le modèle StackingClassifier	56
6.3	Interprétabilité locale	57
6.3.1	Interprétabilité locale du modèle LogisticRegression	58
6.3.2	Interprétabilité locale du modèle RandomForestClassifier.....	59
6.3.3	Interprétabilité locale du modèle GradientBoostingClassifier.....	60
6.3.4	Interprétabilité locale du modèle StackingClassifier	61
7	Conclusion	63
8	Annexes.....	63

1 Introduction

1.1 Description du projet

L'analyse des données marketing appliquée aux entreprises de service est une problématique très classique des sciences de données. Ici, nous disposons d'un jeu de données comprenant des informations relatives aux clients d'une banque et ayant pour finalité la souscription à un produit que l'on appelle « dépôt à terme ». Lorsqu'un client souscrit à ce produit, il place une quantité d'argent dans un compte spécifique et ne sera pas autorisé à disposer de ces fonds avant l'expiration du délai fixé. En échange, le client reçoit des intérêts de la part de la banque à la fin de ce délai.

Les données proviennent de Kaggle et sont disponibles à l'adresse suivante :

<https://www.kaggle.com/janiobachmann/bank-marketing-dataset>

1.2 Problématique et plan du projet

L'objectif de ce projet est assez classique : il s'agit d'améliorer l'efficacité des prochaines campagnes marketing.

Pour cela, il faut augmenter le ratio entre le nombre de clients appelé et le nombre de souscription au produit « dépôt à terme ».

Dans un premier temps, nous effectuerons une analyse visuelle et statistique des facteurs permettant d'expliquer le lien entre les données personnelles du client (âge, statut marital, quantité d'argent placé en banque, nombre de fois que le client a été contacté, etc.) et la variable cible « deposit » qui répond à la question de savoir si le client a souscrit au dépôt à terme. Ensuite, nous définirons plusieurs modèles de classification permettant prédire la souscription ou non des clients. Par ailleurs, dans la mesure où l'industrie Financière est traditionnellement réfractaire aux modèles complexes, considérés comme des « boîtes noires », il sera également nécessaire de fournir une explication simple du choix de chaque client à contacter par le biais d'une analyse de l'interprétabilité du modèle. Effectivement, dans la mesure où les modèles de classification sont généralement utilisés dans le monde bancaire pour distinguer les « bons clients » des « mauvais clients » et leur octroyer ou non des prêts, les banques se servent de l'interprétabilité des modèles qu'elles utilisent habituellement pour justifier aux clients les motifs du refus de leurs emprunts.

1.3 Inventaire des données

Le dataset est constitué de 11162 observations contenant 17 variables :

- 7 variables quantitatives ;
- 10 variables catégorielles (dont la variable cible « deposit »). Il convient de préciser que toutes peuvent être transformées en variables quantitatives ou en variables binaires.

Les variables peuvent être rassemblées en 3 grands groupes :

- Les variables relatives aux clients (« age », « job », « marital », « education », « balance », « default », « housing » et « loan ») ;
- Les variables relatives à la campagne actuelle (« contact », « day », « month », « duration », « campain » et « deposit ») ;
- Les variables relatives à la campagne précédente (« pdays », « previous », « poutcome »).

1.4 Description des différentes variables

- *age* (variable quantitative) : Age du client
- *job* (variable catégorielle) : Type d'emploi du client
- *marital* (variable catégorielle) : Statut marital
- *education* (variable catégorielle) : Niveau d'étude
- *balance* (variable quantitative) : Solde du compte
- *default* (variable catégorielle) : Le client a-t-il fait défaut sur un emprunt?
- *housing* (variable catégorielle) : Emprunt Immobilier
- *loan* (variable catégorielle) : Emprunt à la consommation
- *contact* (variable catégorielle) : Moyen de communication utilisé
- *day* (variable quantitative) : Jour du dernier contact
- *month* (variable catégorielle) : Mois du dernier contact(au format texte)
- *duration* (variable quantitative) : Durée du dernier contact en secondes
- *campaign* (variable quantitative) : Nombre d'appels pendant la campagne
- *pdays* (variable quantitative) : Nombre de jour depuis la campagne précédente
- *previous* (variable quantitative) : Nombre d'appels pendant les campagnes précédentes
- *poutcome* (variable catégorielle) : Résultat de la campagne précédente

- *deposit* (variable catégorielle) : le client a-t-il souscrit ? il s'agit de la variable cible.

2 Gestion des données manquantes

Le dataset ne contient pas de données manquantes au sens informatique (NaN).

Toutefois, 4 variables qualitatives contiennent la modalité « unknown », à savoir : « job », « education », « contact » et « poutcome ». La proportion de cette modalité est faible pour « job » et « education » (<5%), modérée pour « contact » (env. 20%) et majoritaire pour « poutcome » (75%).

3 Analyse des variables

Par souci d'efficacité, nous avons choisi de scinder l'analyse des variables en deux parties :

- Les variables catégorielles d'un côté ;
- Les variables quantitatives de l'autre.

3.1 Analyse des variables catégorielles

Ici, notre objectif était de faciliter l'analyse de chaque variable catégorielle. Pour cela, nous avons élaboré une fonction nous permettant d'automatiser cette analyse (cat_scan). Ainsi, une fois élaborée, cette fonction a été appliquée à chaque variable qualitative et nous a permis de faire ressortir, pour chacune d'elle :

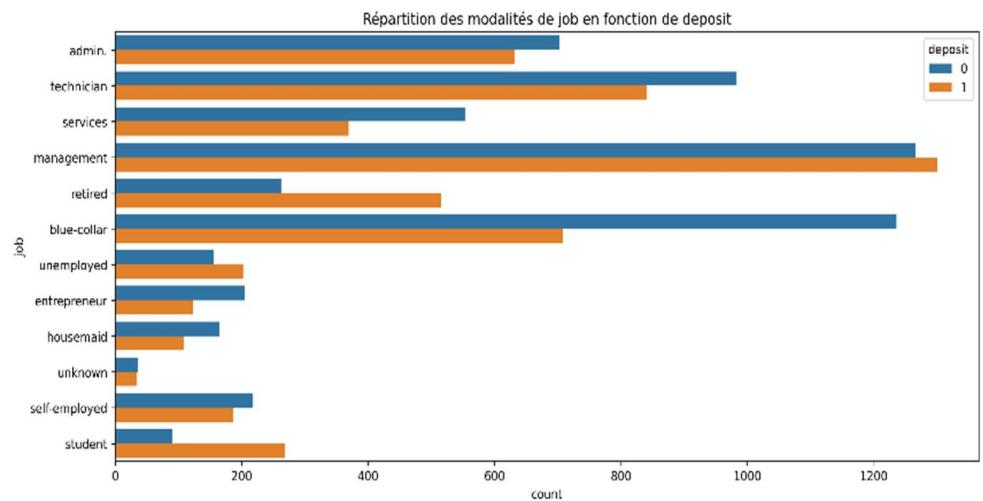
- Sa modalité la plus fréquente ;
- Un tableau constitué de 3 colonnes dans lequel chaque modalité de cette variable a été analysée selon son nombre et la part qu'elle y représente ;
- Une analyse statistique entre elle et la variable cible deposit dans laquelle on a présenté non seulement la p-value mais également le V de Cramer, et ce afin de correctement mesurer leur niveau de corrélation. Une interprétation des résultats a également été apportée ;
- Et enfin un graphique dans lequel la distribution des modalités de la variable a été représentée en fonction de la variable cible.

Nous allons représenter, ci-après, certaines des sorties obtenues pour chaque variable catégorielle après leur avoir appliqué la fonction `cat_scan`.

3.1.1 Analyse de la distribution des variables catégorielles

- Variable « job » :

Modalités	Occurrences	Proportion (%)
<i>management</i>	2566	23.0
<i>Blue-collar</i>	1944	17.4
<i>technician</i>	1823	16.3
<i>admin.</i>	1334	12.0
<i>services</i>	923	8.3
<i>retired</i>	778	7.0
<i>self-employed</i>	405	3.6
<i>student</i>	360	3.2
<i>unemployed</i>	357	3.2
<i>entrepreneur</i>	328	2.9
<i>Housemaid</i>	274	2.5
<i>Unknown</i>	70	1.0



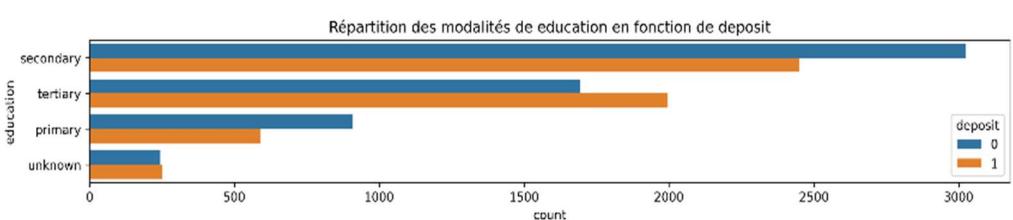
La variable semble avoir une importance sur le résultat de la campagne précédente.

En proportion, les retraités ou les étudiants semblent plus susceptibles de souscrire un dépôt à terme. A l'inverse, les ouvriers (« blue-collar ») souscrivent nettement moins. Les catégories les plus contactées sont les personnes travaillant dans l'encadrement (« management ») et les ouvriers (« blue-collar »).

Le type d'emploi est inconnu (« unknown ») pour une faible proportion des clients (1%).

- Variable « education » :

Modalités	Occurrences	Proportion (%)
<i>secondary</i>	5476	49.1
<i>tertiary</i>	3689	33.0
<i>primary</i>	1500	13.4
<i>unknown</i>	497	4.5



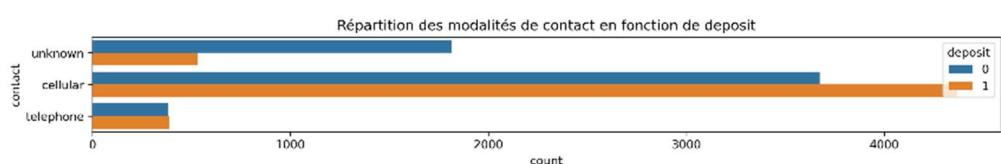
La moitié des clients contactées ont un niveau d'étude secondaire (« secondary »).

Comme on pouvait s'y attendre, les clients ayant un niveau d'étude supérieur (« tertiary ») souscrivent plus au dépôt à terme (succès supérieur à $\frac{1}{2}$) et les niveaux d'étude inférieurs y souscrivent moins en proportion.

Pour une faible proportion des clients (4,5%), le niveau d'étude est inconnu (« unknown »).

- **Variable « contact » :**

Modalités	Occurrences	Proportion (%)
cellular	8042	72.0
unknown	2346	21.0
telephone	774	6.9

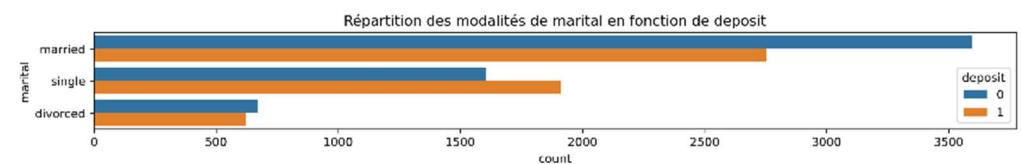


Les clients contactés par téléphone portable (« cellular ») sont très majoritaires (72%) et plus susceptibles de souscrire un dépôt à terme en proportion.

Pour une part importante des clients, le type de contact est inconnu (« unknown »). Pour ces clients, le taux de succès est très nettement inférieur. Ceci est difficilement explicable en l'état.

- **Variable « marital » :**

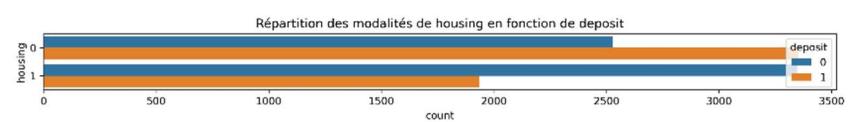
Modalités	Occurrences	Proportion (%)
married	6351	56.9
single	3518	31.5
divorced	1293	11.6



Les clients mariées (« married ») représentent la majorité d'entre eux (56,9%). Bien que leur taux de souscription au dépôt à terme soit supérieur aux autres dans les faits, il ne l'est pas en proportion. Effectivement, ce sont les clients célibataires (« single ») qui y souscrivent le plus, en proportion. Par ailleurs, toujours de manière proportionnelle, les clients divorcés (« divorced ») y souscrivent également plus que les clients mariés, mais moins que les clients célibataires.

- **Variable « housing » :**

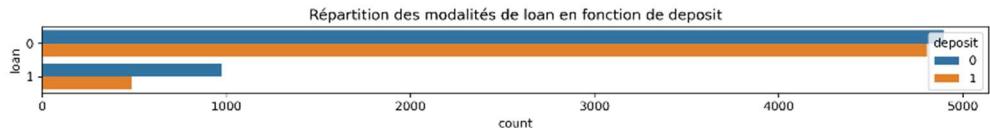
Modalités	Occurrences	Proportion (%)
0	5881	52.7
1	5281	47.3



La proportion des clients ayant ou non un crédit immobilier est équilibrée. Cependant, le taux de souscription au dépôt à terme est bien inférieur chez les détenteurs de crédits (« 1 ») : ceci est logique car l'on peut supposer que leur situation financière est moins bonne.

- **Variable « loan » :**

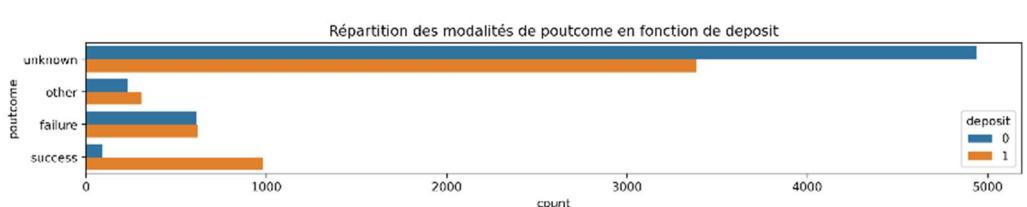
Modalités	Occurrences	Proportion (%)
0	9702	86.9
1	1460	13.1



Une grande majorité de clients n'a pas de crédit à la consommation (87%). Les clients ayant un crédit (« 1 ») souscrivent beaucoup moins au dépôt à terme : ceci est logique car, comme vu précédemment, l'on peut supposer que leur situation financière est moins bonne.

- **Variable « poutcome » :**

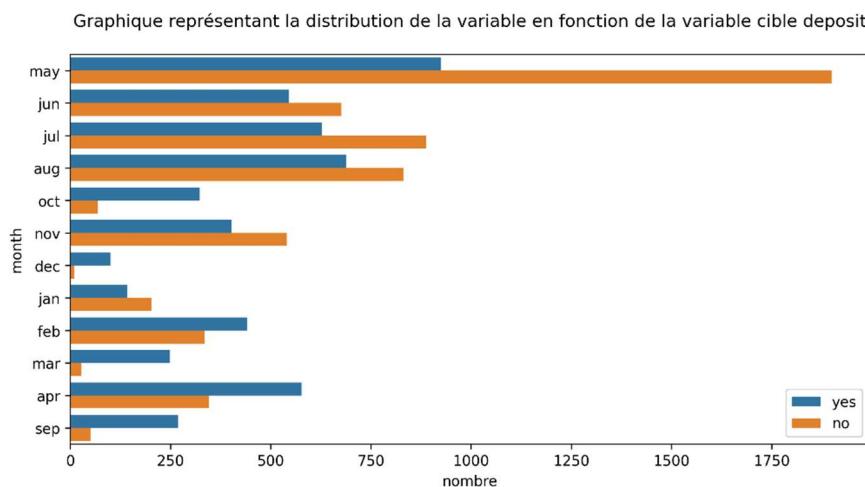
Modalité	Occurrences	Proportion (%)
unknown	8326	74.6
failure	1228	11.0
success	1071	9.6
other	537	4.8



Il s'agit d'un indicateur de succès de la campagne précédente (« poutcome » correspond à Previous Outcome). A priori, cette variable pourrait être considérée comme importante pour répondre à notre problématique dans la mesure où un client ayant déjà souscrit au dépôt à terme par le passé est probablement susceptible d'y souscrire à nouveau. Ainsi, il serait logique d'avoir deux modalités Succès/Failure pour cette variable. Or, nous constatons que la modalité « unknown » est très majoritaire (75%). A cela s'ajoute une autre modalité « other » qui n'apporte pas plus d'informations. Les clients ayant le statut « unknown » pour cette variable sont ceux qui n'avaient pas été contactés lors de la campagne précédente. Le terme « unknown » est d'ailleurs mal choisi : « none » serait plus adapté. Pour les autres clients, on constate que lorsque la campagne précédente (« poutcome ») est un succès, le taux de souscription à la campagne actuelle (« deposit ») est très élevé (95%).

- **Variable « month » :**

Modalités	Occurrences	Proportion (%)
may	2824	25.0
aug	1519	14.0
jul	1514	14.0
jun	1222	11.0
nov	943	8.0
apr	923	8.0
feb	776	7.0
oct	392	4.0
jan	344	3.0
sep	319	3.0
mar	276	2.0



Cette variable présente des variations difficilement explicables :

- Certains mois sont très sur-représentés sans raison apparente (« may »); d'autres sous-représentés (« dec », même si les fêtes de fin d'année pourraient l'expliquer) ;
- De plus, les taux de succès sont très variables (faibles en « apr », puis très forts en « may »).

On peut supposer que la nature des campagnes précédentes en est la cause :

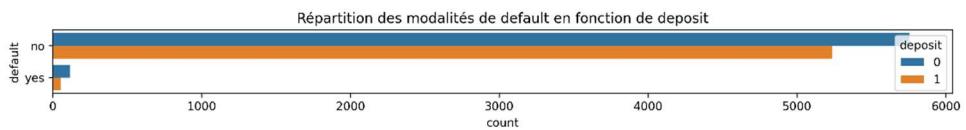
- Le dernier appel du client peut avoir différentes raisons : il peut être lié à la campagne en cours, à la précédente ou à une autre raison ;
- Ces campagnes sont probablement en nombre limité, non réparties de manière homogène sur l'année ;
- Les campagnes ont leur dynamique propre : on peut, par exemple, imaginer une campagne commençant lentement en avril et se terminant en mai avec un fort taux de souscription.

Rien ne nous permet toutefois de confirmer ou d'inflammer ces suppositions.

Par ailleurs, dans la mesure où l'on constate que la majorité des refus de dépôt ont lieu au mois de mai, celui-ci étant le plus représenté au sein de la variable « month », on peut supposer que cette dernière constituera un réel problème au niveau de la proportionnalité entre les modalités de la variable cible « deposit » lors de l'élaboration de nos modèles de Machine-Learning. Nous verrons par la suite si cela a eu un réel impact dans les parties modélisation et features engineering.

- Variable « **default** » :

Modalités	Occurrences	Proportion (%)
0	5881	52.7
1	5281	47.3



Cette variable présente une inégalité flagrante entre ses modalités : tandis que la modalité « no » (clients n'ayant jamais fait défaut à un crédit) est sur-représentée, la modalité « yes » est quant à elle quasiment inexiste (clients ayant déjà présenté un défaut de paiement). De manière équivalente, le taux de souscription au dépôt à terme est d'autant plus important en nombre chez les clients n'ayant jamais fait défaut à un crédit, même si, proportionnellement, le résultat n'est pas très éloigné entre ces deux modalités. Cela peut se justifier aisément par le fait que les clients ayant d'ores et déjà fait défaut à un crédit sont tout simplement beaucoup moins démarchés par la banque que les autres au cours de la campagne marketing et, de ce fait, ils y souscrivent beaucoup moins en nombre (même si en proportion ils y souscrivent plus que les clients n'ayant, quant à eux, jamais défaut à un crédit).

- **Variable « deposit » (cible) :**

La variable cible détermine si un client a souscrit à un dépôt à terme. Il s'agit d'une variable binaire dont la répartition est assez équilibrée, comme on peut le voir ci-dessous. Nous avons toutefois effectué un léger oversampling (nous le verrons après).

Modalités	Occurrences	Proportion (%)
0	5873	52.6
1	5289	47.4

3.1.2 Analyse des corrélations entre les variables catégorielles et la variable cible

Globalement, il est ressorti de cette analyse qu'aucune variable catégorielle n'était fortement corrélée avec la variable cible "deposit". Par souci de rigueur, nous allons tout de même insérer ici les différentes sorties qui ont été obtenues en appliquant la fonction cat_scan afin que les résultats puissent être précisément observés.

- **Variable « job » :**

Corrélation variables deposit - job :

Statistique : 378.0752558664989 , p-value : 2.741689587081072e-74 , degré_de_liberté : 11
V de Cramer : 0.1813532820862077
On rejette H0 car les deux variables ne sont pas indépendantes.
Cependant, leur corrélation est assez faible selon le V de Cramer

- **Variable « education » :**

Corrélation variables deposit - education :

Statistique : 122.77008967211442 , p-value : 1.9534186354212715e-26 , degré_de_liberté : 3
V de Cramer : 0.1035910291517223
On rejette H0 car les deux variables ne sont pas indépendantes.
Cependant, leur corrélation est assez faible selon le V de Cramer

- **Variable « contact » :**

Corrélation variables deposit - contact :

Statistique : 736.6866796046972 , p-value : 1.0728032438445805e-160 , degré_de_liberté : 2
V de Cramer : 0.25656622110819854
On rejette H0 car les deux variables ne sont pas indépendantes.
Cependant, leur corrélation est assez faible selon le V de Cramer

- **Variable « marital » :**

Corrélation variables deposit - marital :

Statistique : 109.58335610012479 , p-value : 1.600576988089358e-24 , degré_de_liberté : 2
V de Cramer : 0.09817945795517834
On rejette H0 car les deux variables ne sont pas indépendantes.
Cependant, leur corrélation est assez faible selon le V de Cramer

- **Variable « housing » :**

Corrélation variables deposit - housing :

Statistique : 463.1892407533161 , p-value : 9.724394114495535e-103 , degré_de_liberté : 1
V de Cramer : 0.20349714531691168
On rejette H0 car les deux variables ne sont pas indépendantes.
Cependant, leur corrélation est assez faible selon le V de Cramer

- **Variable « loan » :**

```
Corrélation variables deposit - loan :
```

```
Statistique : 135.83217051738103 , p-value : 2.171286879630289e-31 , degré_de_liberté : 1  
V de Cramer : 0.10991198880407216  
On rejette H0 car les deux variables ne sont pas indépendantes.  
Cependant, leur corrélation est assez faible selon le V de Cramer
```

- **Variable « poutcome » :**

```
Corrélation variables deposit - poutcome :
```

```
Statistique : 1004.635780185333 , p-value : 1.7761850102620281e-217 , degré_de_liberté : 3  
V de Cramer : 0.29957343855644286  
On rejette H0 car les deux variables ne sont pas indépendantes.  
Cependant, leur corrélation est assez faible selon le V de Cramer
```

- **Variable « month » :**

```
Corrélation variables deposit - month :
```

```
Statistique : 1046.7745027840656 , p-value : 1.6420829584486923e-217 , degré_de_liberté : 11  
V de Cramer : 0.304635736793811  
On rejette H0 car les deux variables ne sont pas indépendantes.  
Cependant, leur corrélation est assez faible selon le V de Cramer
```

- **Variable « default » :**

```
Corrélation variables deposit - default :
```

```
Statistique : 17.808573693473093 , p-value : 2.4428001791928345e-05 , degré_de_liberté : 1  
V de Cramer : 0.038807236869287685  
On rejette H0 car les deux variables ne sont pas indépendantes.  
Cependant, leur corrélation est assez faible selon le V de Cramer
```

3.2 Analyse des variables quantitatives

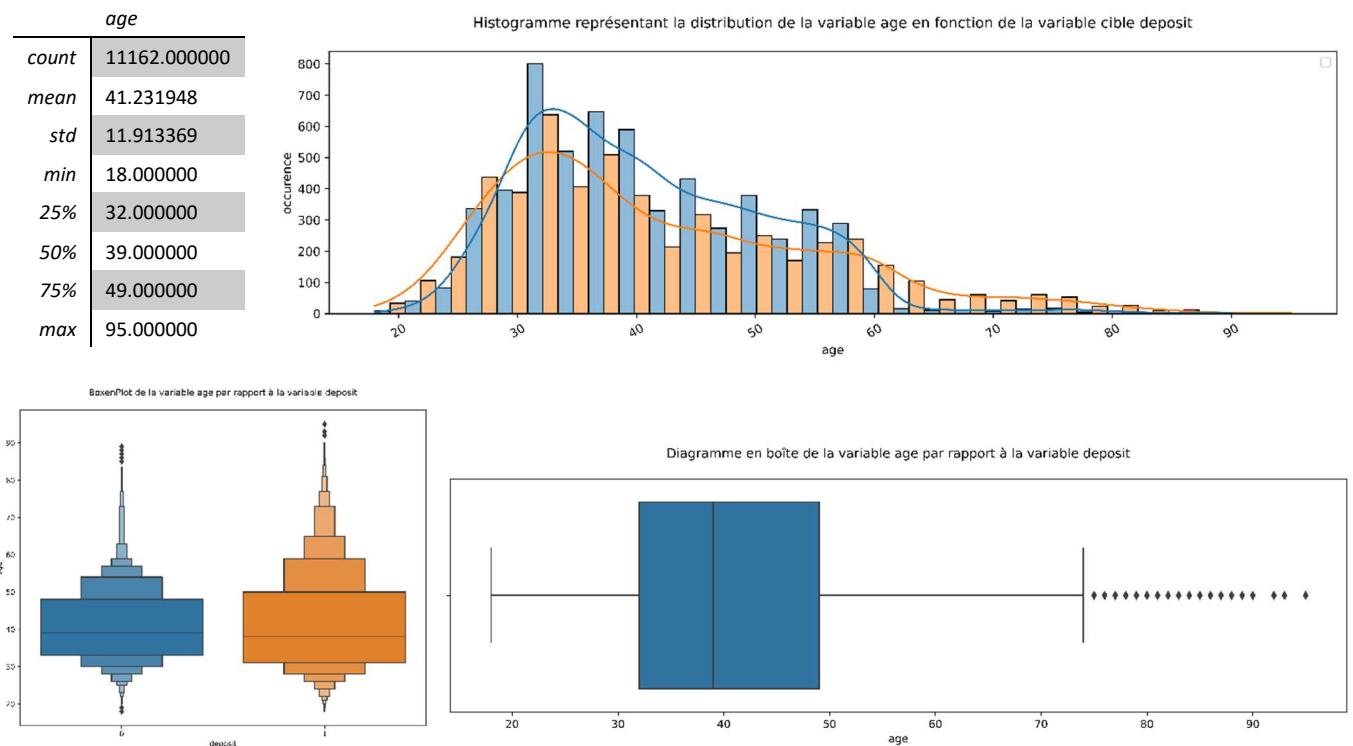
Comme nous l'avions effectué avec les variables qualitatives, nous avons également élaboré une fonction afin d'analyser automatiquement la distribution ainsi que la présence de valeurs aberrantes dans toutes les variables quantitatives de notre jeu de données : il s'agit de la fonction num_scan. A la différence de la précédente, elle avait pour objectif de représenter graphiquement les valeurs aberrantes de chacune des variables quantitatives et également de permettre l'analyse de leur distribution. Elle se constitue :

- d'un tableau présentant les principales statistiques descriptives de chaque variable ;
- d'un boxplot représentant la variable concernée en fonction de la variable cible deposit ;
- d'un boxenplot de ladite variable, encore en fonction de notre variable cible ;
- Et enfin, d'un histogramme permettant de représenter la distribution de la variable en question, toujours en fonction de notre variable cible deposit.

Nous allons représenter, ci-après, les sorties qui ont été obtenues pour chaque variable quantitative après leur avoir appliqué la fonction num_scan.

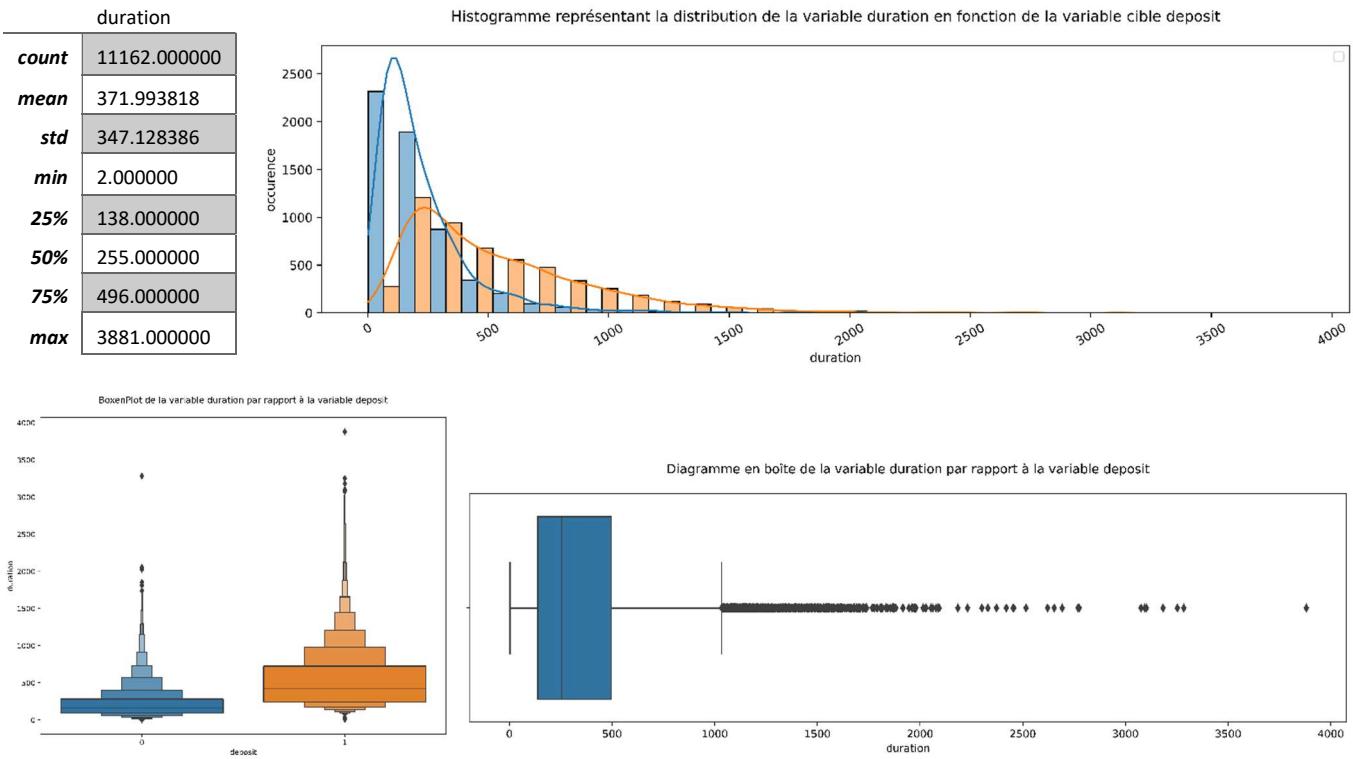
3.2.1 Analyse de la distribution des variables quantitatives

- Variable « age » :



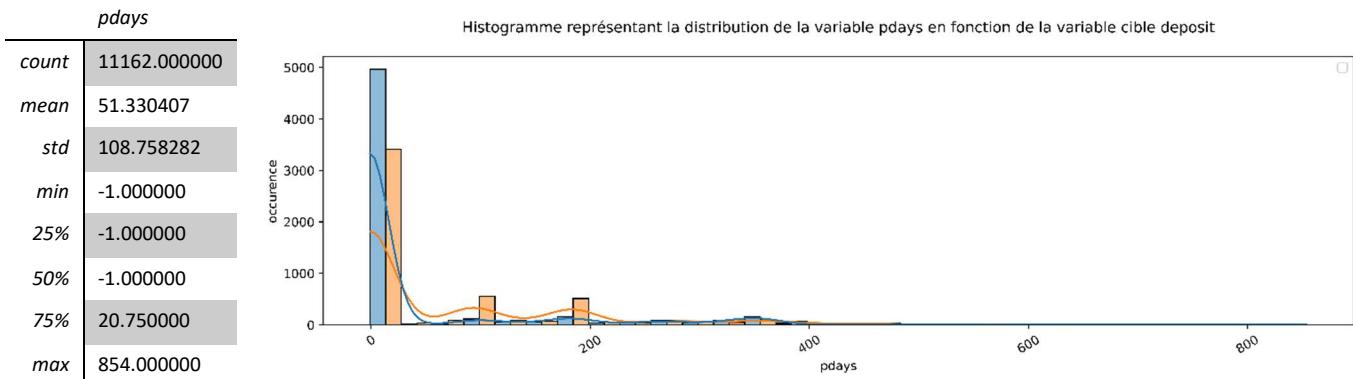
Les clients contactés ont entre 18 et 95 ans. La majorité d'entre eux a entre 30 et 50 ans (moyenne et médiane vers 40 ans). Les clients ayant un âge avancé (au-delà de 75 ans) sont sous-représentés (ce qui est logique). Malgré tout, on constate que les clients les plus âgés et les plus jeunes (avant 30 ans) ont, en proportion, un taux de souscription au dépôt à terme supérieur.

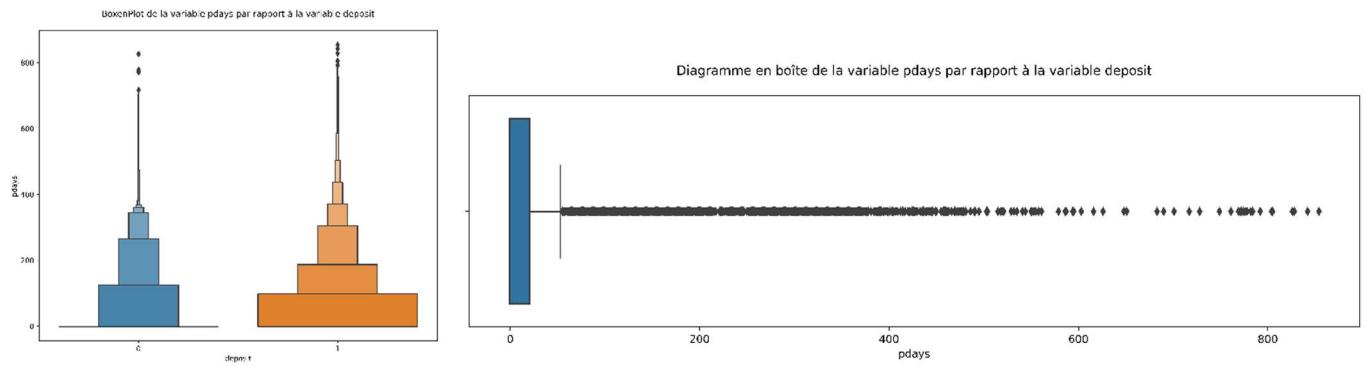
- Variable « duration » :



Il s'agit de la durée du dernier appel. Les appels durent entre 0 et 3900 secondes (1h 5min), et ceux supérieurs à 10 minutes sont sous-représentés. On constate un fort décalage entre les durées des appels ayant débouché sur un succès et les autres (ceci est logique : plus l'appel est long, plus cela démontre l'intérêt du client). Par conséquent, cette variable semble fortement corrélée au succès, nous le verrons par la suite.

- **Variable « pdays » :**



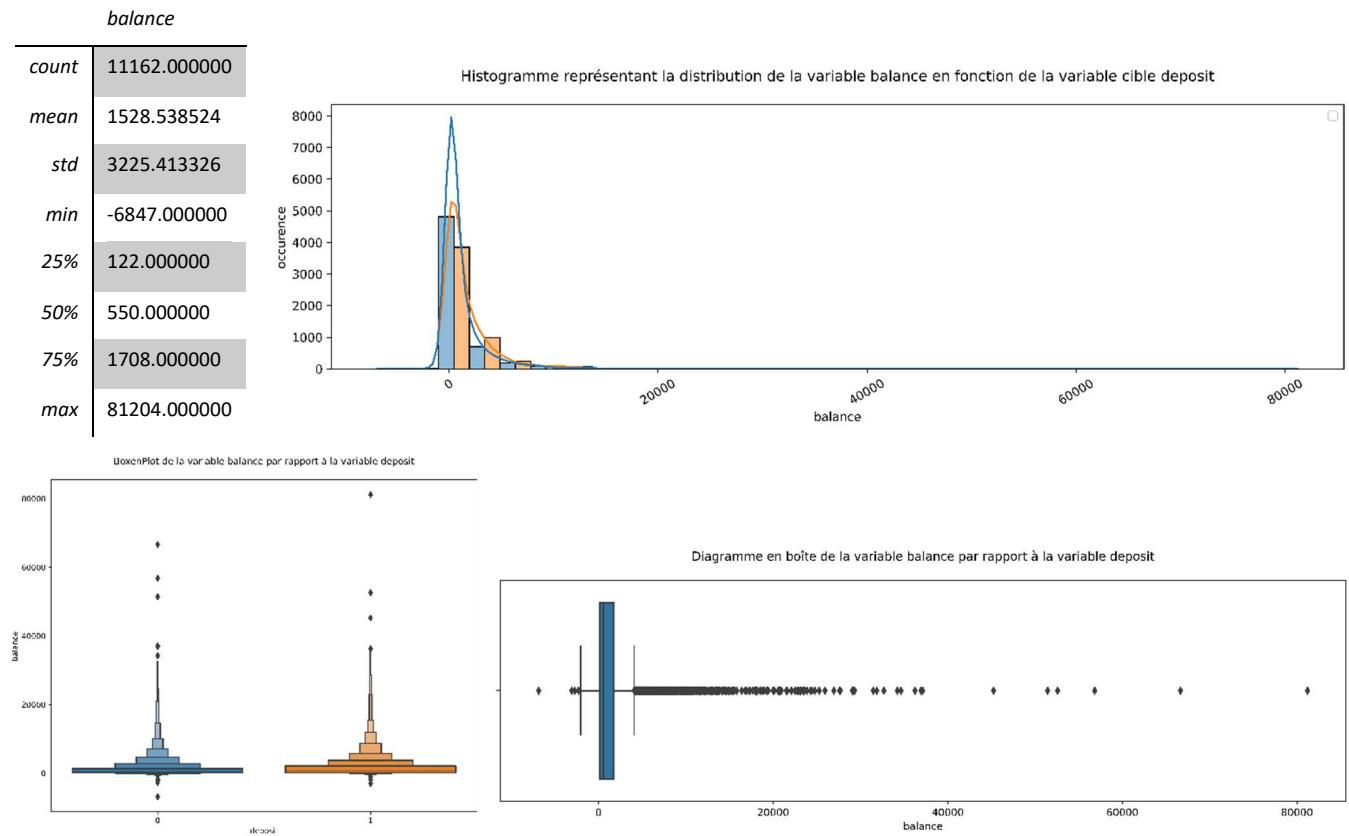


Pour 75 % des client, cette variable est -1. Il ne s'agit pas d'une valeur aberrante mais plus probablement d'un indicateur pour les clients qui n'ont pas été appelés lors de la campagne précédente.

Pour le reste, La plupart des clients ont été contactés depuis moins d'un an.

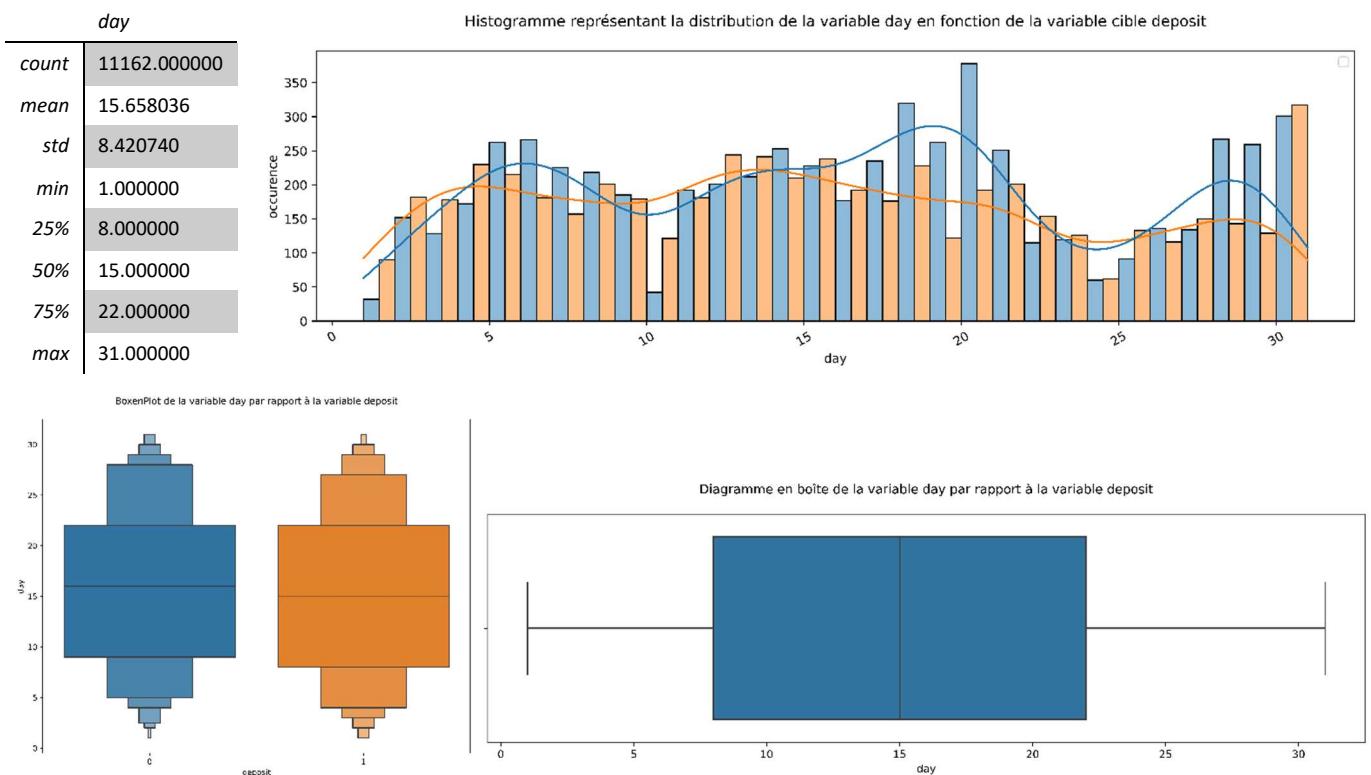
On constate une forte proportion de succès pour deux pics (vers 85 et 180 jours). On peut supposer qu'il y a eu deux campagnes précédentes : les personnes appelées lors de la campagne précédente ont donc fortement souscrit à la campagne actuelle.

- Variable « balance » :**



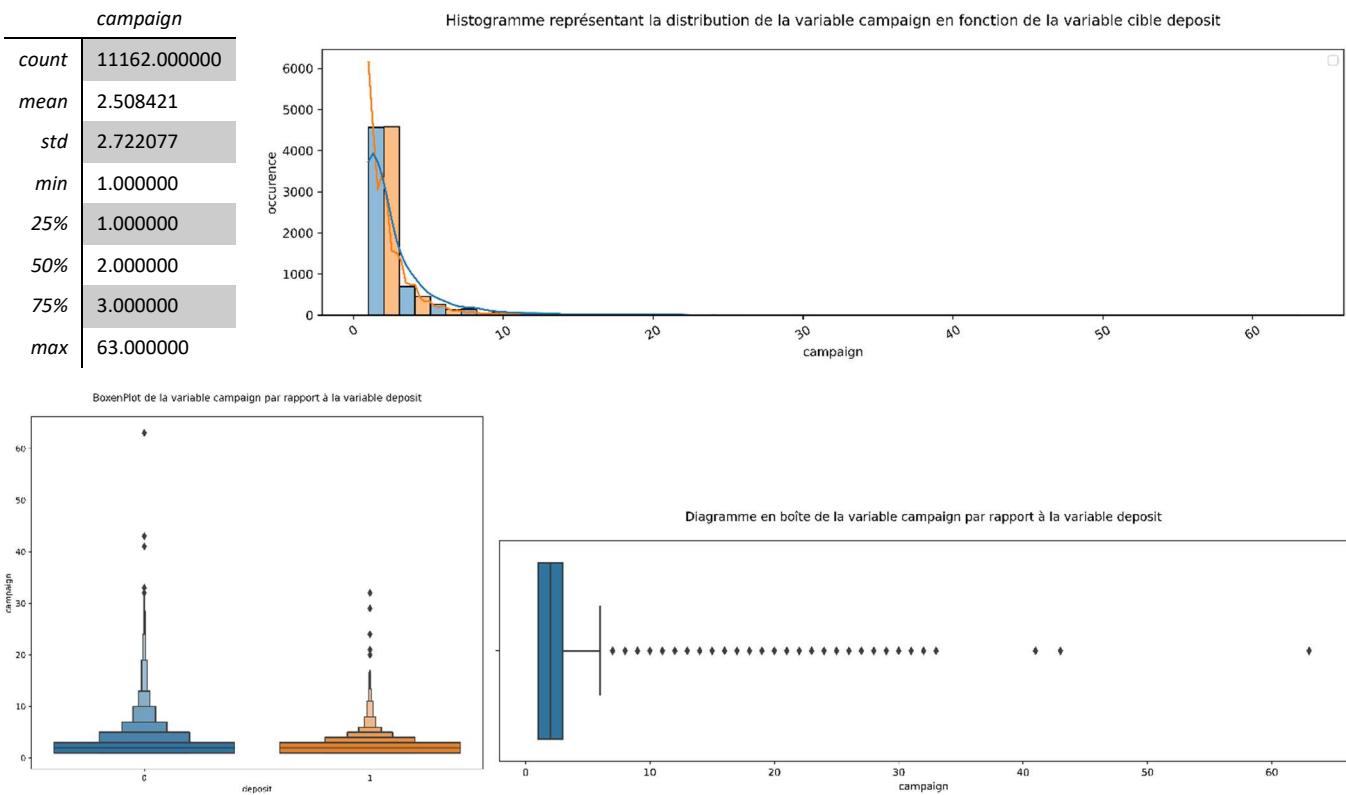
Aucune valeur ne peut être considérée comme aberrante même si certaines sont hors-norme (une valeur vers -6800 et plusieurs supérieurs à 40000, ce qui conduit à un écart significatif entre la moyenne (1530) et la médiane (550)). En effet, si l'on exclut les valeurs extrêmes de l'affichage (coupure à -1000 et +6000) on voit que 75% des clients sont en dessous de 1700 (voir tableau des statistiques descriptives ci-dessus). Par ailleurs, on constate que la part de souscription au dépôt à terme augmente proportionnellement à l'augmentation du solde du compte. C'est ce qui peut être observé dans l'histogramme de la distribution ci-dessus : plus le solde du compte est important, plus la proportion de clients qui souscrivent au contrat à terme est importante.

- **Variable « day » :**



Les jours d'appels sont équitablement répartis au fil des mois. Toutefois, deux baisses peuvent être observées en début et en fin de mois, à savoir vers le 10^{ème} et le 25^{ème} jour. Ces baisses sont difficilement explicables en l'état.

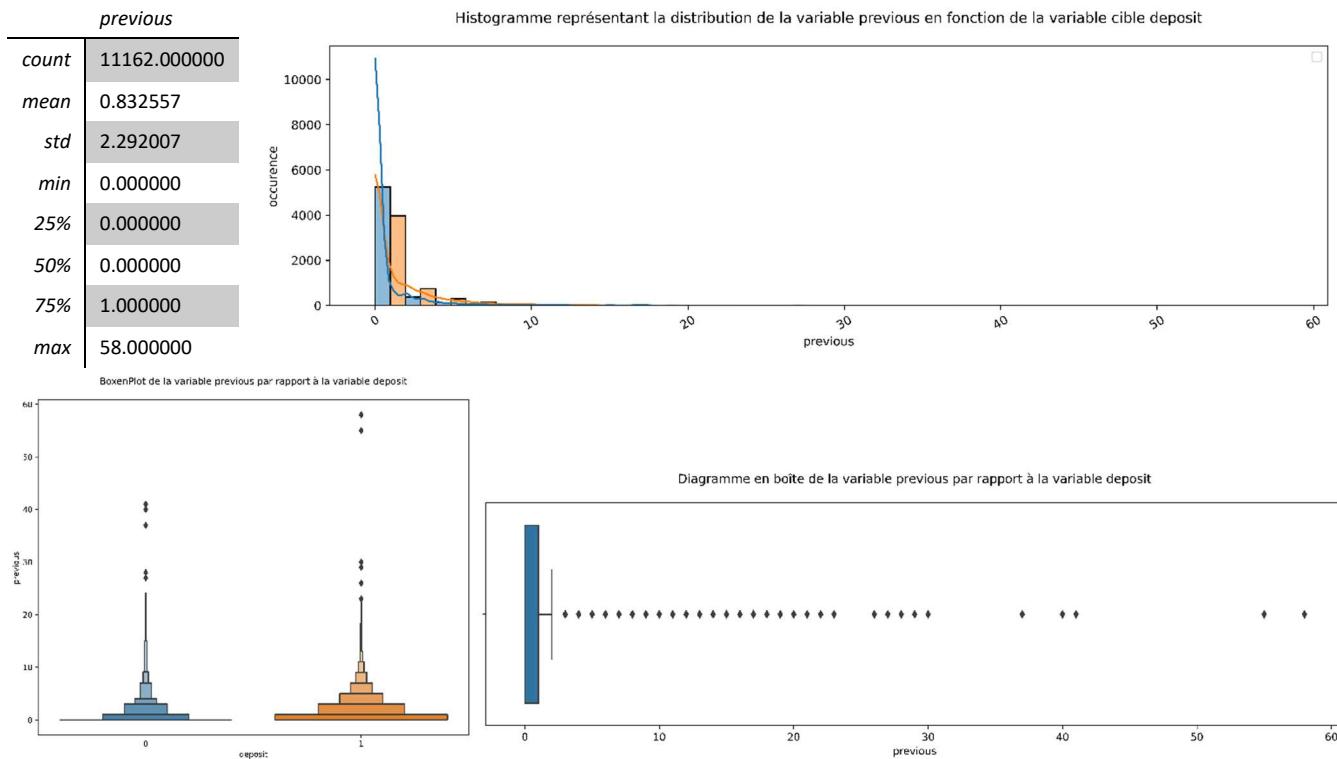
- **Variable « campaign » :**



Il s'agit du nombre d'appels émis vers les clients au cours de la campagne. On constate que certains d'entre eux ont été contactés plusieurs dizaines de fois, mais que la plupart ont été appelés moins de 10 fois (75% d'entre eux ont été appelés 3 fois seulement selon le tableau des statistiques descriptives). Ainsi, le nombre d'appels « hors norme » est très faible et ne semble pas réellement affecter la moyenne qui reste proche de la médiane.

D'autre part, en termes de nombre, on constate que les clients contactés deux fois semblent davantage souscrire au dépôt à terme que les autres clients. La raison en est très simple : la moitié des clients ne sont contactés que deux fois. Il est donc normal de constater que leur taux de souscription au contrat est supérieur aux autres en termes de nombre. Cependant, proportionnellement, on constate l'effet inverse : les clients contactés entre 2 et 10 fois souscrivent beaucoup plus au dépôt à terme en termes de proportion. Cela signifie que les clients souscriraient certainement davantage s'ils étaient contactés plus de deux fois.

- **Variable « previous » :**



Il s'agit du nombre d'appels émis au cours de la campagne précédente. Comme pour *Campaign*, on constate que certains clients ont été contactés plusieurs dizaines de fois, mais que la plupart ont été appelés moins de 10 fois (75% d'entre eux n'ont été appelés qu'une seule fois).

On constate également une forte proportion de « 0 » : il s'agit de clients qui n'avaient pas été contactés lors de cette campagne mais qui ont été contactés lors de la suivante (« campaign »). Ces clients sont majoritaires et leur taux de souscription est inférieur à ceux qui avaient été contactés lors de la campagne précédente (>0 sur cet histogramme).

Par ailleurs, le même constat que précédemment peut être effectué : en termes de proportion, les clients contactés plus de 2 fois lors de la campagne précédente souscrivent davantage au contrat à terme actuel. Cela signifie que les clients souscriraient certainement davantage s'ils étaient contactés plus de deux fois.

3.2.2 Présence de valeurs aberrantes

Après avoir brièvement analysé la distribution de chacune des 7 variables quantitatives, nous nous concentrerons désormais sur la présence de valeurs aberrantes. Pour cela, nous allons présenter, dans le tableau ci-dessous, une analyse globale de la présence de valeurs aberrantes au sein des variables quantitatives par le biais de l'écart entre :

- La moyenne et la médiane ;
- Les quantiles ;
- Le min et le max.

	moyenne	médiane	diff_médiane_moyenne	écart-type	quantile_1	quantile_2	quantile_3	min	max	Ecart max-min
age	41.23	39.0	2.23	11.913369	32.0	39.0	49.00	18	95	77
balance	1528.54	550.0	978.54	3225.413326	122.0	550.0	1708.00	-6847	81204	88051
day	15.66	15.0	0.66	8.420740	8.0	15.0	22.00	1	31	30
duration	371.99	255.0	116.99	347.128386	138.0	255.0	496.00	2	3881	3879
campaign	2.51	2.0	0.51	2.722077	1.0	2.0	3.00	1	63	62
pdays	51.33	-1.0	52.33	108.758282	-1.0	-1.0	20.75	-1	854	855
previous	0.83	0.0	0.83	2.292007	0.0	0.0	1.00	0	58	58

De prime abord, on constate la présence de valeurs aberrantes particulièrement pour 3 variables quantitatives, à savoir « balance », « duration », et « pdays ». On peut observer cela dans les colonnes « diff_médiane_moyenne » et « Ecart max-min » notamment, ainsi que dans l'écart entre les différents quantiles pour ces mêmes variables. Cependant, comme on l'a brièvement abordé dans la partie précédente pour certaines d'entre elles, cela reste à nuancer. Effectivement, il y a peu de points isolés dans les différents graphiques de visualisation. Les diagrammes en boîte ainsi que les histogrammes présentent souvent des alignements de points qui ressemblent plus à des distributions exponentielles qu'à de vrais outliers, d'où l'intérêt de ne pas se contenter uniquement de valeurs brutes dans notre analyse mais d'observer également la forme de la distribution de chaque variable.

Malgré tout, après de nombreux essais, nous avons réussi à identifier quelques points à extraire de notre jeu de données :

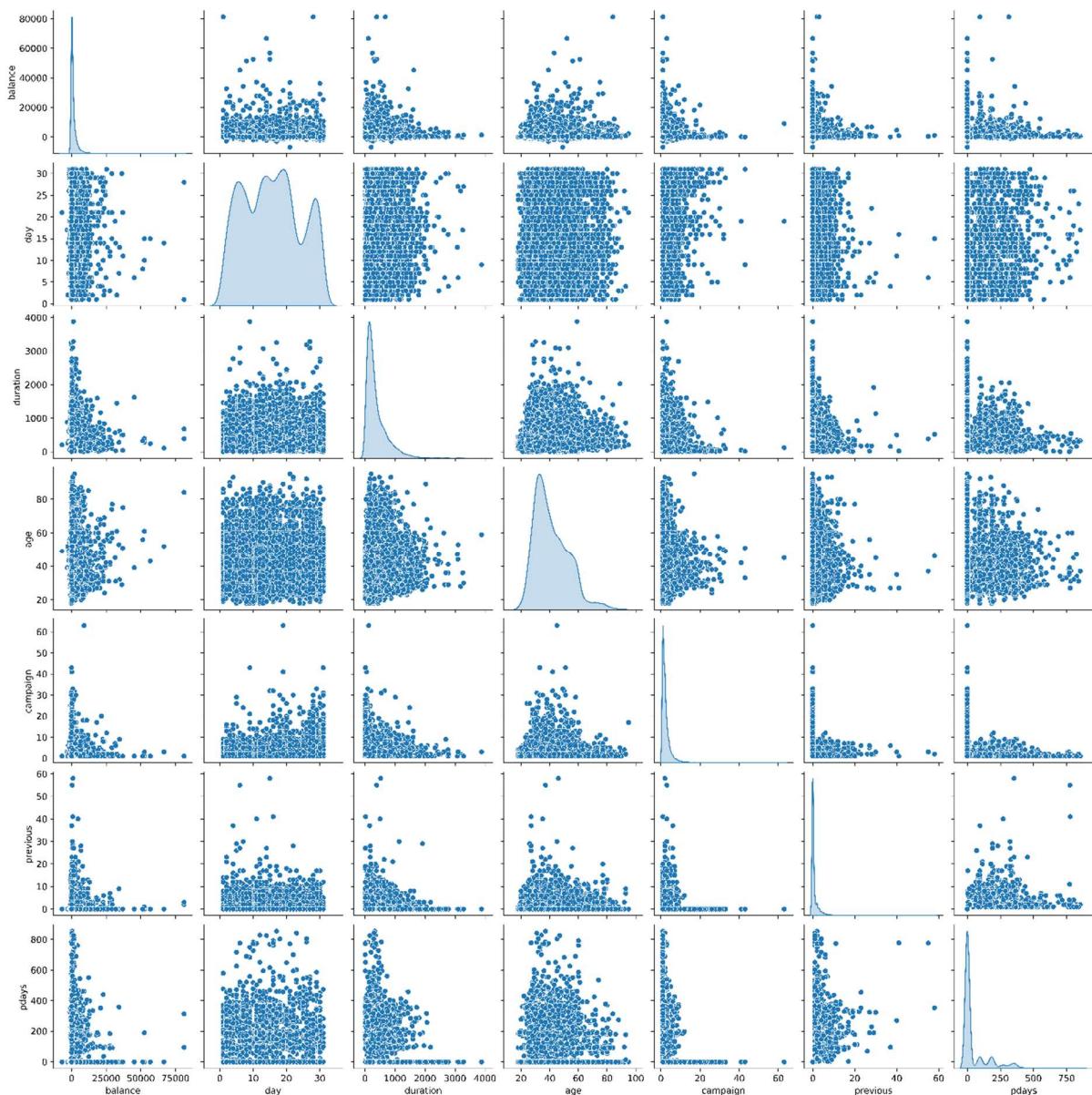
- Les clients dont la variable « balance » est supérieure à 60000 ;
- Les clients qui ont été appelé plus de 35 fois non seulement pour la campagne en cours (« campaign ») mais aussi pour la précédente (« previous ») ;
- Les clients pour lesquels la variable « pdays » est supérieure à 750. En effet, nous nous sommes interrogés sur les valeurs extrême de « pdays » et nous avons tout d'abord estimé que si la durée depuis la campagne précédente était supérieure à 1 an (365) les points pouvaient être considérés comme des outliers. Etonnamment, ceci a eu un fort impact négatif sur les résultats des modèles de classification utilisés. C'est la raison pour laquelle nous nous sommes restreints au nombre 750.

Finalement, nous n'avons supprimé que 60 observations dans notre jeu de données. Nous avons considéré qu'il était inutile de tenter une substitution des valeurs et avons décidé de simplement supprimer les observations.

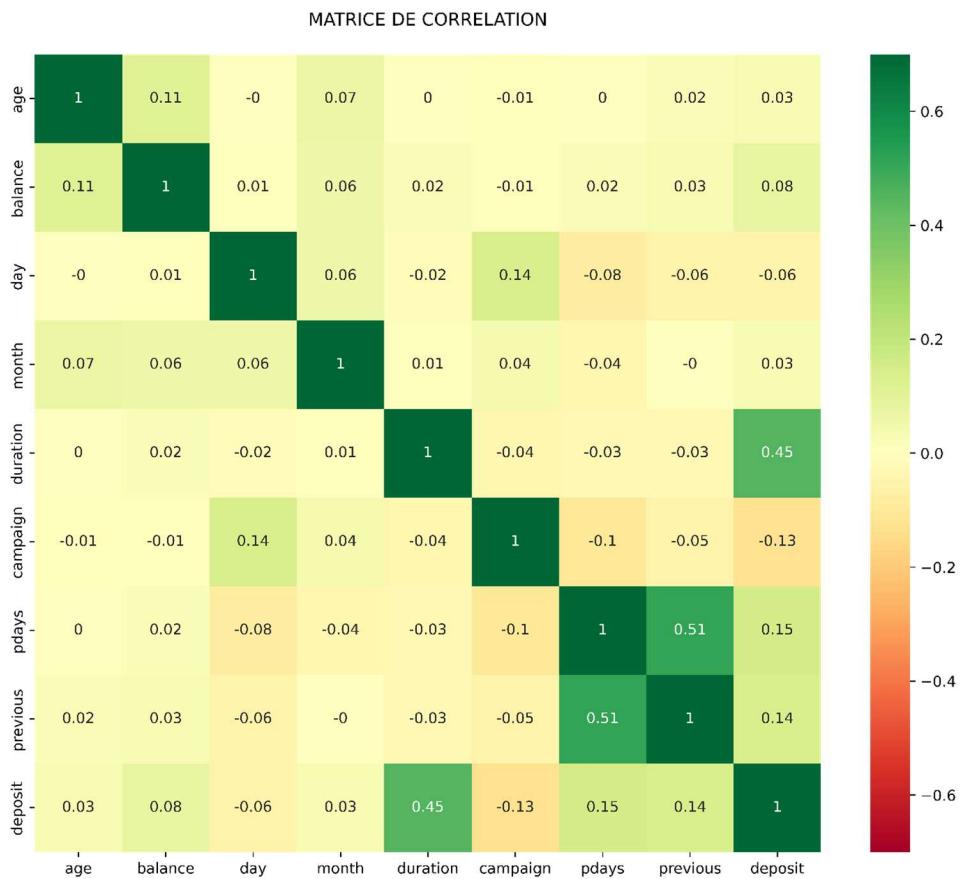
Note : la suppression de 60 observations sur 11000 semble marginale. De ce fait, cette étape de suppression des outliers était peut-être superflue.

3.2.3 Corrélation et multi-colinéarité

Une fois l'identification des outliers achevée, nous nous sommes intéressés à l'analyse de la corrélation entre les variables quantitatives. Pour cela, nous avons privilégié, dans un premier temps, une analyse visuelle par l'élaboration d'un Pairplot représentant les nuages de points entre les différentes variables quantitatives que voici :



On peut déjà observer par la forme des différents nuages de points que les corrélations entre les variables quantitatives sont quasiment inexistantes, sauf pour une exception, à savoir « pdays » et « previous ». La matrice de corrélation nous a apporté davantage de détails, notamment sur la corrélation entre la variable cible « deposit » et une autre variable quantitative :



L'analyse de la matrice de corrélation nous indique que les corrélations entre les variables quantitatives sont dans l'ensemble assez faibles (« month » a été labellisée pour être transformée en variable numérique ici). Les seules exceptions constituent les couples de variables « previous » / « pdays » et « deposit » / « duration » :

- Pour le premier couple, c'est assez logique car nous avons vu que « previous » constitue le nombre d'appels émis lors de la campagne précédente et que « pdays » constitue, quant à lui, le nombre de jours depuis cette dernière campagne. Or nous avons également constaté que 75 % des clients n'avaient pas été appelés lors de la campagne précédente : ils ont donc tous pour valeurs « previous » = -1 et « pdays » = « unknown ». Malgré cela la corrélation totale est seulement de 0.51, ce qui signifie que les 25% restant ne présentent pas une colinéarité marquée.

- Pour le second couple, c'est également tout à fait logique : nous avons vu précédemment que plus la durée d'un appel est longue et plus la proportion de clients qui souscrivaient à l'offre devenait importante. On en déduit que plus l'appel est long, plus l'intérêt du client est important. Cette variable est donc fortement corrélée à notre variable cible « deposit ».

Pour confirmer ce résultat, nous avons effectué un test de Pearson entre la variable cible et « duration ». Celui-ci a confirmé notre constat concernant la corrélation entre les deux variables ci-dessus :

```
Test de corrélation de Pearson variables deposit-duration
```

```
resultat_test
pearson_coeff      0.45
p-value            0.00
```

Le coefficient de Pearson n'est pas négligeable, comme on a déjà pu le observer dans la matrice de corrélation. La p-value étant strictement inférieure à 0.5, on rejette H_0 : les deux variables testées ne sont pas indépendantes. De plus, la corrélation est assez significative selon le coefficient de Pearson.

Nous avons désormais élaboré une analyse suffisamment détaillée des différentes variables qui composent notre jeu de données. Nous allons maintenant détailler les transformations qui ont été opérées dans ces différentes variables afin d'être en mesure d'élaborer nos modèles de classification.

4 Preprocessing des données

L'objectif était de préparer le jeu de données à l'élaboration des différents modèles de classification sélectionnés. Pour ce faire, nous avons cherché, en premier lieu, à transformer nos différentes variables (features engineering) afin de sélectionner celles qui nous paraissaient les plus significatives selon les différents modèles de classification que nous avions sélectionnés (features selection).

4.1 Features engineering

4.1.1 Encodage des variables

On a cherché ici à transformer les variables catégorielles en variables numériques afin d'éviter les erreurs qui pourraient survenir dans nos différents modèles de classification, ces derniers se révélant incompatibles avec ce type de variables. Ainsi :

- Pour les variables comprenant uniquement les deux types de modalités « yes » et « no », le « One Hot Encoding » nous paraissait le plus adapté afin de les convertir en variables indicatrices : il était effectivement assez aisément de remplacer « no » par « 0 » et « yes » par « 1 ». Les variables concernées étaient « deposit », « default », « loan » et « housing ».
- Pour les variables présentant un nombre de modalités plus important, nous avons privilégié la dichotomisation, ce qui revient à créer de nouvelles variables auxquelles on aurait appliqué un One Hot Encoding. En effet, la dichotomisation revient à convertir chaque modalité de la variable dichotomisée en nouvelle variable indicatrice. Les variables concernées étaient « job », « marital », « education », « contact » et « poutcome ». Le choix de cet encodage dans le cas présent a induit la création d'une quinzaine de variables supplémentaires. Pour ces variables, il était également possible d'appliquer un Ordinal Encoding (discrétisation) dans la mesure où leurs modalités présentaient une hiérarchie entre elles. A titre d'exemple, « education » présentait en son sein une hiérarchie assez claire avec ses modalités « primary », « secondary » et « tertiary ». Cependant, la présence de la modalité « unknown » pour la plupart d'entre elles (« job », « education », « contact » et « poutcome ») qui n'entrait pas dans cette hiérarchie et qu'il n'était pas possible de substituer nous a convaincu d'appliquer la dichotomisation pour chacune d'elles : notre objectif était de supprimer les modalités « unknown » sans que cela ne crée de Nans dans le jeu de données, ce qui aurait été le cas si l'on avait choisi un encodage différent. Ainsi, avec la dichotomisation, nous obtenions une variable pour cette modalité, le problème étant que ces variables ne correspondaient pas à des segmentations réelles de clients. Par exemple : nous avons vu que la variable « poutcome » prenait la valeur « unknown » lorsque les clients n'avaient pas été contactés lors de la campagne précédente (ils étaient 75 % dans ce cas). Toutefois, ils ne constituaient pas une population à part entière, il s'agissait de clients qui auraient souscrit (success) ou non (failure) si on les avait contactés.
- La variable « month » quant à elle a été renommée en appliquant à chaque mois le numéro qui lui correspondait de la manière suivante : « may » : 5, « jun » : 6, « jul » : 7, « aug » : 8, « oct » : 10, « nov » : 11, « dec » : 12, « jan » : 1, « feb » : 2, « mar » : 3, « apr » : 4, « sep » : 9.

Après leur encodage, le nombre de variables du jeu de données est passé à 38 :

- « age » ;
- « default » ;
- « balance » ;
- « housing » ;
- « loan » ;

- « day » ;
- « month » ;
- « duration » ;
- « campaign » ;
- « pdays » ;
- « previous » ;
- « deposit » ;
- « job_admin » ;
- « job_blue-collar » ;
- « job_entrepreneur » ;
- « job_housemaid » ;
- « job_management » ;
- « job_retired » ;
- « job_self-employed » ;
- « job_services » ;
- « job_student » ;
- « job_technician » ;
- « job_unemployed » ;
- « job_unknown » ;
- « marital_divorced » ;
- « marital_married » ;
- « marital_single » ;
- « education_primary » ;
- « education_secondary » ;
- « education_tertiary » ;
- « education_unknown » ;
- « contact_cellular » ;
- « contact_telephone » ;
- « contact_unknown » ;
- « poutcome_failure » ;
- « poutcome_other » ;
- « poutcome_success » ;
- « poutcome_unknown » .

4.1.2 Normalisation des données

Après avoir encodé les données, il a été constaté que celles-ci n'étaient pas à la même échelle :

	age	default	balance	housing	loan	day	month	duration	campaign	pdays	...
0	59	0	2343	1	0	5	5	1042	1	-1	...
1	56	0	45	0	0	5	5	1467	1	-1	...
2	41	0	1270	1	0	5	5	1389	1	-1	...

Par conséquent, nous avons appliqué une normalisation afin de les rendre homogènes et comparables :

	age	default	balance	housing	loan	day	month	duration	campaign	pdays	...
0	1.492832	-0.123763	0.279751	1.055983	-0.387831	-1.265324	-0.461443	1.928097	-0.578461	-0.487403	...
1	1.240846	-0.123763	-0.490675	-0.946985	-0.387831	-1.265324	-0.461443	3.151626	-0.578461	-0.487403	...
2	-0.019083	-0.123763	-0.079982	1.055983	-0.387831	-1.265324	-0.461443	2.927072	-0.578461	-0.487403	...

4.1.3 Equilibrage des modalités de la variable cible « deposit »

Après avoir normalisé les données, une dernière étape importante du processus de Features Engineering était de vérifier que la variable cible « deposit » était bien équilibrée afin d'éviter le surentraînement des modèles de Machine-Learning :

```
distribution de la variable cible target
 0      0.526311
 1      0.473689
Name: deposit, dtype: float64
```

D'après les résultats, on a constaté qu'elle était à peu près équilibrée. Malgré tout, on a tout de même appliqué un Oversampling par le biais de SMOTE et de RandomOverSampler afin qu'elles deviennent complètement équilibrées :

```
Classes échantillon oversampled :
{0: 0.5, 1: 0.5}

Classes échantillon SMOTE :
{0: 0.5, 1: 0.5}
```

4.2 Features selection

4.2.1 Réduction de dimension par la méthode PCA

Une tentative de réduction de dimension a été élaborée par la méthode PCA (Analyse en composantes principales) :

```
Nombre de composantes retenues avec PCA en conservant une variance expliquée à 95%: 27  
array([0.10015863, 0.1733475 , 0.24267111, 0.29222008, 0.33811738,  
       0.37817058, 0.41660938, 0.45022485, 0.48349621, 0.5161002 ,  
       0.54792391, 0.5787445 , 0.60895949, 0.63859564, 0.66767017,  
       0.69629589, 0.72471338, 0.75238697, 0.77867151, 0.80464949,  
       0.82962018, 0.8534874 , 0.8765447 , 0.89862172, 0.92063827,  
       0.94080454, 0.95941837])
```

Cependant, au vu de la faible corrélation des variables, cette méthode n'était pas très utile dans le cas présent puisque l'intérêt de la PCA est justement d'éliminer les corrélations entre les variables par le biais de la combinaison linéaire, tout en conservant l'ensemble de l'information (avec une variance expliquée à 95%). C'est la raison pour laquelle nous avons estimé qu'il n'y avait aucun intérêt à conserver les résultats de la PCA dans le cas présent.

4.2.2 Sélection de variables selon le modèle LogisticRegression : Wrapper Methods (Recursive Feature Elimination - RFE) et méthodes traditionnelles

Nous avons choisi de ne pas mettre en application les Embedded Methods pour la sélection des variables car elles utilisent le modèle Support Vector Machine : dans le cas de la classification, ce n'est pas optimal en termes de temps d'exécution. Les Wrapper Methods, quant à elles, et plus précisément la RFE (Recursive Feature Elimination), peuvent utiliser un modèle de régression logistique. C'est la méthode que nous avons privilégiée ici car c'est un modèle plus stable que les arbres de décision (DecisionTreeClassifier) que nous avions testés. Nous avons fait tourner la fonction plusieurs fois afin de tester la stabilité du modèle avec une validation croisée à 5 n_split, et une dizaine de variables a toujours été sélectionnée. Il s'agissait généralement des variables :

Nombre de features retenus : 17

Les variables qui ont été sélectionnées par wrapper method sont :

```
['balance', 'housing', 'loan', 'day', 'campaign', 'job_retired', 'job_student', 'marital_married', 'marital_single', 'education_primary', 'education_tertiary', 'contact_cellular', 'contact_telephone', 'contact_unknown', 'poutcome_failure', 'poutcome_success', 'poutcome_unknown']
```

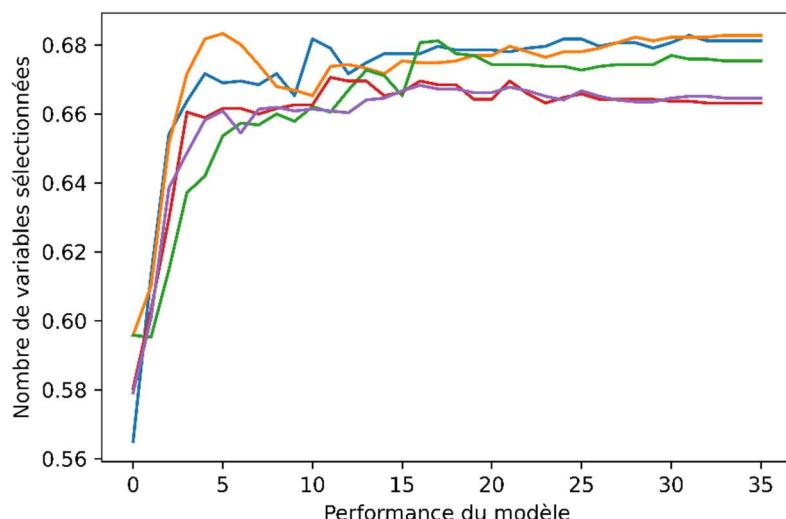
Le nombre de variables supprimées par le modèle est de : 19

Les variables qui ont été supprimées par wrapper method sont :

```
['age', 'default', 'month', 'pdays', 'previous', 'job_admin.', 'job_blue-collar', 'job_entrepreneur', 'job_housemaid', 'job_mangement', 'job_self-employed', 'job_services', 'job_technician', 'job_unemployed', 'job_unknown', 'marital_divorced', 'education_secondary', 'education_unknown', 'poutcome_other']
```

Cela était cohérent avec le graphique que nous avions élaboré nous permettant d'évaluer la performance du modèle (la courbe des scores) en fonction du nombre de variables sélectionnées :

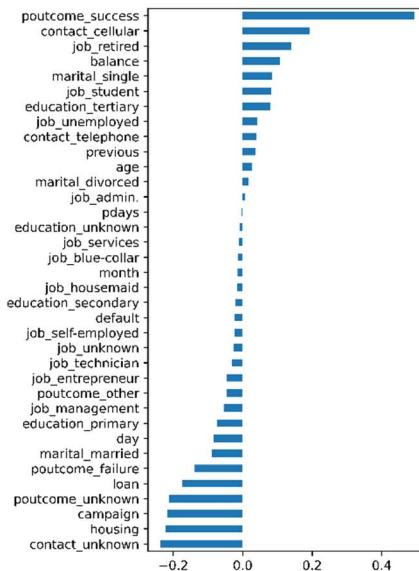
Performance du modèle en fonction du nombre de variables sélectionnées par validation croisée



Comme on peut l'observer, le score du modèle se stabilise et n'évolue pas entre 15 et 20 variables. Cela signifie que, selon le modèle de régression logistique appliqué à la RFE, il n'est pas nécessaire de conserver plus de 15-20 variables pour améliorer la performance du modèle. En appliquant les méthodes traditionnelles au modèle de régression logistique (méthode `coef_` appliquée à la régression logistique), on a également obtenu un graphique représentant les variables de notre jeu de données classées selon leur niveau de significativité :

- Les variables classées en hauteur ont une forte influence positive sur le modèle ;
- Les variables classées au milieu ont un degré de significativité faible ;
- Les variables classées en bas ont une forte influence négative sur le modèle.

Les variables les plus importantes selon le modèle de régression logistique :



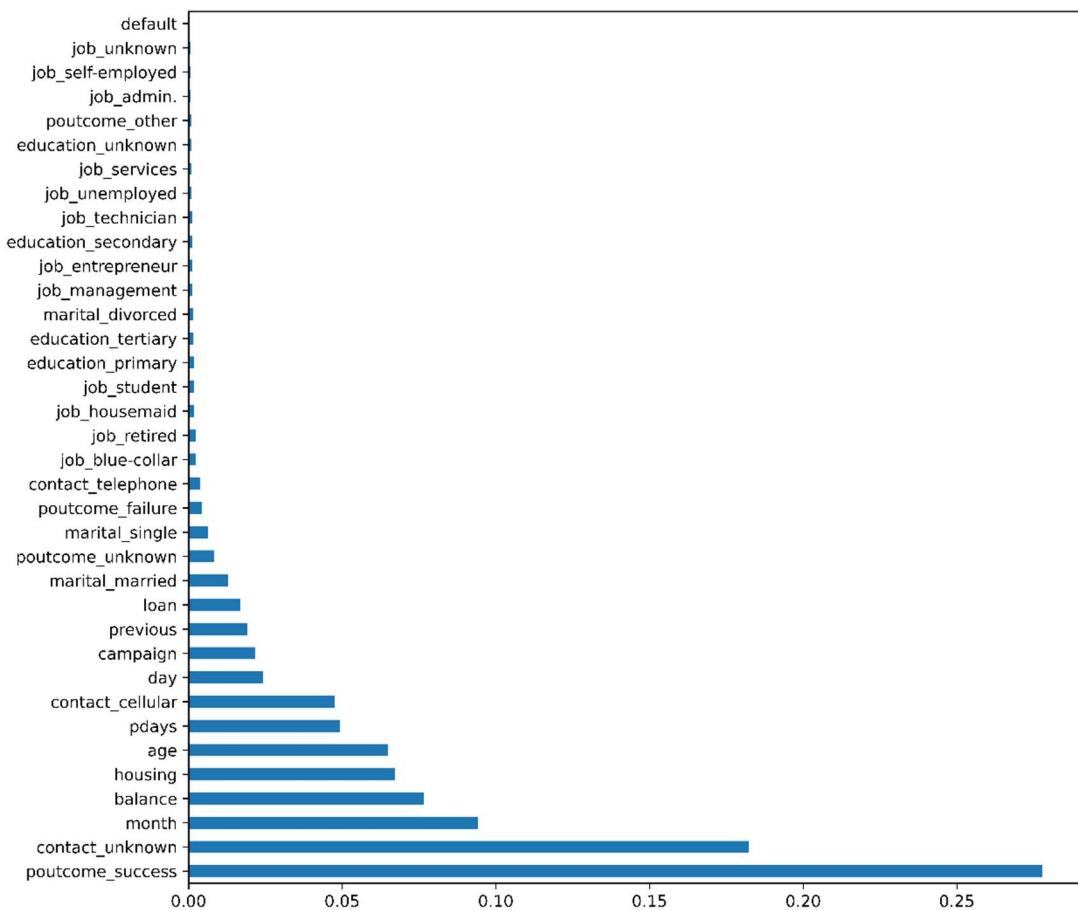
On retrouvait bien les mêmes résultats que précédemment : les variables supprimées par le biais de la RFE sont bien présentes au milieu du graphique tandis que celles qui ont été conservées par la RFE sont présentes aux deux extrémités du graphique. Pour conclure, si l'on décidait de suivre le modèle de régression logistique, nous ne devions conserver que les 15-20 variables citées précédemment.

4.2.3 Sélection de variables par méthodes traditionnelles appliquées aux modèles RandomForestClassifier et GradientBoostingClassifier

Nous avons ensuite appliqué les méthodes traditionnelles de sélection de variables à la fois au **RandomForestClassifier** et au **GradientBoostingClassifier** :

- En appliquant les méthodes traditionnelles au modèle de forêts aléatoires (méthode `feature_importances_` appliquée au modèle `RandomForestClassifier`), on a obtenu un graphique représentant les variables de notre jeu de données classées selon leur niveau de significativité :

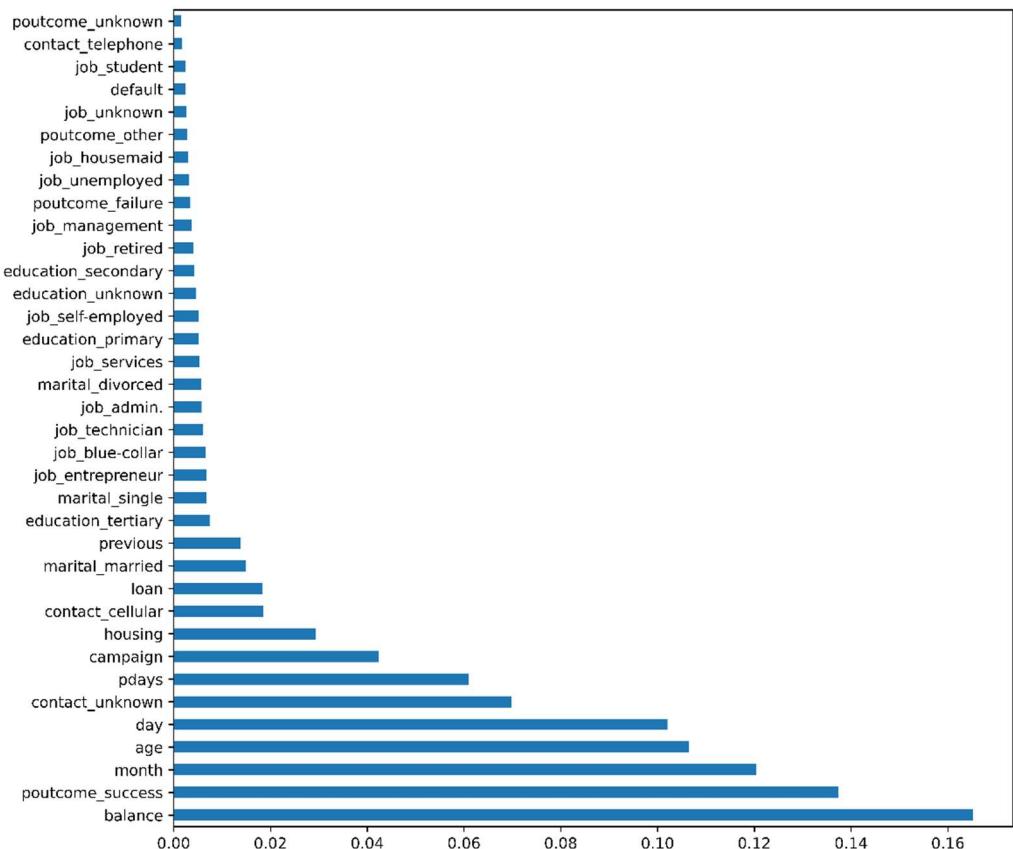
Les variables les plus importantes selon RandomForestClassifier :



On a ainsi constaté qu'à l'inverse de la régression logistique, le modèle de forêts aléatoires considère les variables « month », « pdays », « previous » et « age » comme significatives.

- En appliquant les méthodes traditionnelles cette fois-ci au modèle de GradientBoosting (méthode `feature_importances_` appliquée au modèle `GradientBoostingClassifier`), sans surprise, on a obtenu le même graphique représentant les variables de notre jeu de données classées selon leur niveau de signification :

Les variables les plus importantes selon GradientBoostingClassifier :



Comme on peut l'observer, la conclusion était la même que pour le modèle de forêts aléatoires : les variables considérées comme peu significatives au sein de la régression logistique sont au contraire cruciales au sein du GradientBoostingClassifier (« month », « pdays », « previous » et « age »). Il n'est pas étonnant de constater les mêmes résultats que pour le RandomForestClassifier puisque tous deux sont basés sur la méthode des arbres de décision. Leur principale différence réside dans la manière dont les arbres se construisent : alors que les arbres se construisent indépendamment avec le RandomForestClassifier, ils se construisent un à un avec le GradientBoostingClassifier. Effectivement, le principe pour ce dernier est le même que celui du StackingClassifier : il introduit un algorithme apprenant faible pour améliorer leur efficacité au fur et à mesure que les arbres se construisent et il combine donc les résultats en cours de route et non à la fin comme le RandomForestClassifier. Il n'est donc pas étonnant de constater que les résultats ne sont pas complètement similaires dans les deux graphiques, notamment concernant l'ordre des variables même si, globalement, les mêmes variables sont considérées comme significatives.

4.2.4 Choix des variables et justification

La problématique de notre étude était de construire un modèle prédictif applicable aux campagnes à venir. Nous utilisions pour cela les données de la dernière campagne en date. Notre modèle devait donc s'appuyer sur des variables qui étaient disponibles avant la campagne afin d'effectuer un ciblage des clients.

Les variables relatives aux clients (« age », « job », « marital », « education », « balance », « default », « housing » et « loan ») devaient être à priori disponibles. Les variables relatives à la campagne précédente (« pdays », « previous » et « poutcome ») devaient également être disponibles à l'avance. En revanche, certaines variables relatives à la campagne (« contact », « day », « month » et « duration ») posaient problème :

- Variable « contact » : le type de contact pouvaient être déduits des contacts passés ;
- Variables « day » et « month » : les jours et mois de l'appel étaient en revanche plus problématiques. Une nouvelle campagne de placement de compte à terme aurait forcément une durée limitée dans le temps et il était difficile de définir à l'avance le mois et le jour propice pour contacter les clients. Nous ne pouvions, tout au mieux, qu'estimer le jour de la semaine le plus favorable ;
- Variable « duration » : de même, par définition, la durée des appels n'était pas disponible avant la campagne, ceci sans compter le fait que cette variable était trop fortement corrélée avec la variable cible « deposit ». Elle ne pouvait donc pas être utilisée dans la modélisation.

En raison de tous ces éléments, et après une multitude de tests de suppression de variables qui se sont avérés infructueux (que nous ne détaillerons pas ici), nous avons considéré qu'il était préférable d'éliminer les variables « duration », « day », « month », « pdays », « contact_unknown », « poutcome_other », « poutcome_unknown », « job_unknown » et « education_unknown ». Cependant, notre choix s'axait principalement sur les résultats des modèles de classification que nous avions expérimentés au départ, à savoir les K plus proches voisins (KNeighborsClassifier), les arbres de décision (DecisionTreeClassifier) et la régression logistique. Bien que ce dernier modèle ait été conservé, nous le verrons par la suite, il nous a finalement paru judicieux de sélectionner les variables à supprimer en fonction de celles considérées comme peu significatives par les modèles basés sur les arbres de décision. Effectivement, nous le verrons dans la partie suivante, les résultats les plus intéressants que l'on ait obtenus ont été produits par des modèles de classification reposant sur les arbres de décision. De ce fait, pour améliorer davantage nos résultats, il était indispensable de prendre en considération les deux graphiques précédents dans lesquels les variables sont classées par ordre de significativité selon les modèles RandomForestClassifier et GradientBoostingClassifier. Pour tester cette théorie, nous avons appliqué nos modèles de classification avec et sans sélection de variables. Dans le cas où nous supprimions les variables « duration », « day », « month », « pdays », « contact_unknown », « poutcome_other », « poutcome_unknown », « job_unknown » et « education_unknown », les résultats étaient bien moins satisfaisants que lorsque nous ne les supprimions pas. C'est la raison pour laquelle les variables « day », « month » et « pdays » ont été conservées, car

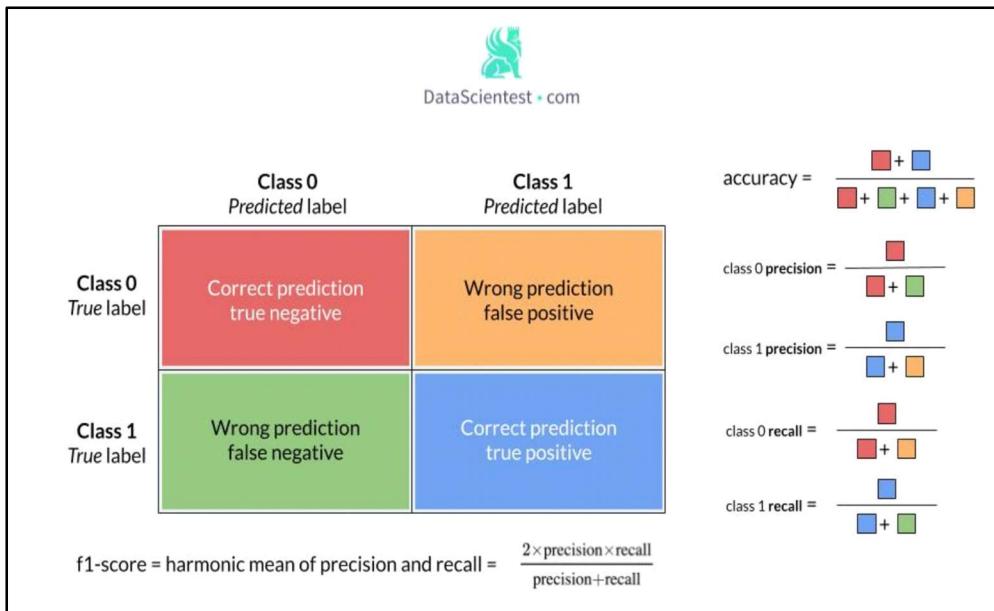
leur suppression nuisait fortement aux scores de nos modèles de classification puisqu'elles étaient considérées comme significatives au sein des modèles RandomForestClassifier et GradientBoostingClassifier. En revanche, les résultats demeuraient similaires malgré la suppression des autres variables « contact_unknown », « poutcome_other », « poutcome_unknown », « job_unknown » et « education_unknown » (hormis la variable « duration » fortement corrélée à la variable cible « deposit » et dont la suppression détériorait les résultats). De ce fait, nous avons estimé qu'il n'était pas judicieux de supprimer d'autres variables que celles-ci.

Ainsi, les variables que nous avons choisi de supprimer de notre jeu de données sont « duration », « contact_unknown », « poutcome_other », « poutcome_unknown », « job_unknown » et « education_unknown ». Autrement dit, nous avons choisi de supprimer la seule variable fortement corrélée avec la variable cible (« duration ») et toutes les variables douteuses ayant pour suffixe « unknown » qui ont été créées à la suite de la dichotomisation appliquée aux différentes variables citées précédemment (« job », « poutcome », « education » et « contact »). Par ailleurs, « poutcome_other » a également été supprimée car, comme nous l'avons vu dans la partie relative à l'analyse des variables catégorielles, elle est totalement incompréhensible. En effet, il existe uniquement deux possibilités : soit les campagnes précédentes sont un succès soit elles sont un échec, aucune autre réponse n'est cohérente (d'autant plus qu'elle s'ajoute à une autre modalité « unknown »). Malgré tout, il convient de préciser que cette sélection de variables à supprimer a été effectuée arbitrairement : elle n'a pas permis d'améliorer les résultats mais a simplement permis de se débarrasser de variables considérées comme inexplicables et gênantes dans notre modèle.

5 Modélisation

5.1 Choix de la métrique à utiliser

Afin de déterminer la bonne métrique à considérer en fonction de notre problématique, nous nous sommes appuyés sur le schémas suivant :



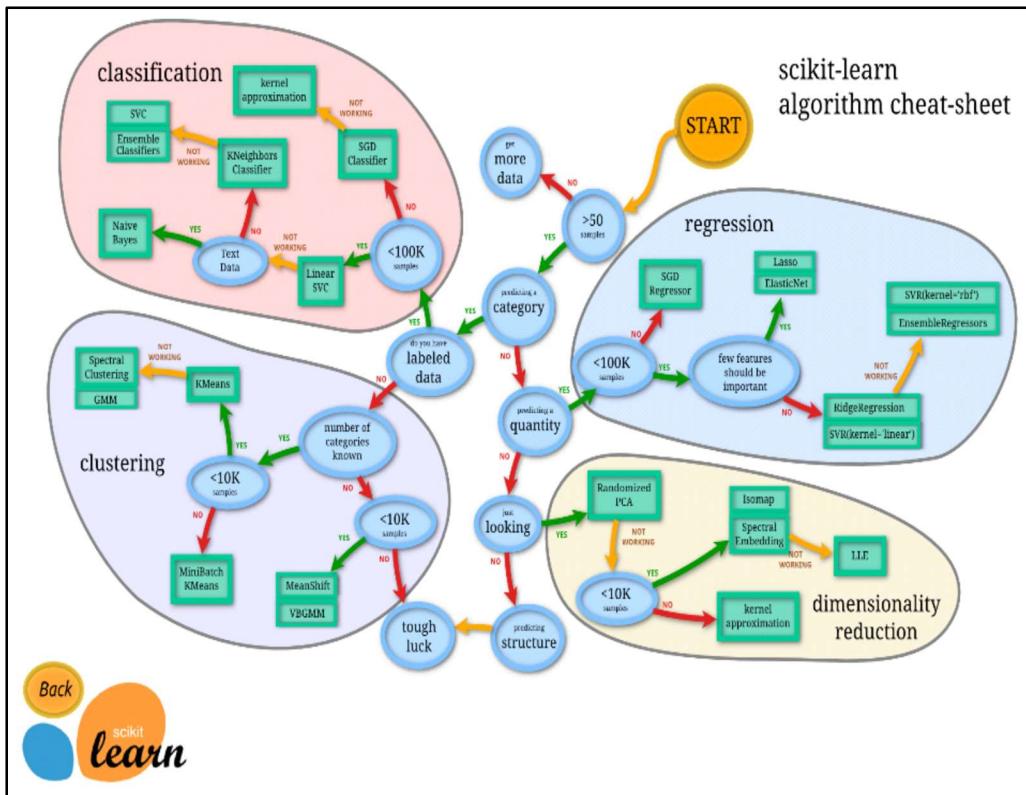
Le choix de la métrique était essentiel pour l'évaluation de nos modèles. Deux métriques simples nous ont semblé tout à fait adaptées dans le cas présent :

- Le rappel de la modalité 1 : il s'agit de la proportion des clients intéressés que notre modèle identifie (vrais positifs / vrais positifs + faux négatifs). C'est l'indicateur à choisir dans le cas où la banque souhaite maximiser le nombre de souscriptions (maximiser le revenu).
- La précision de la modalité 1 : il s'agit de la proportion des prédictions positives de notre modèle qui s'avèrent exactes (vrais positifs / vrais positifs + faux positifs). C'est l'indicateur à choisir si la banque souhaite minimiser le nombre d'appels émis vers des clients non intéressés (limiter les couts).

Dans notre cas, nous n'avions pas d'informations sur la stratégie commerciale de la banque. Nous avons donc choisi d'utiliser une autre métrique : le f1-Score des modalités 0 et 1 de la variable cible. Il s'agit de la moyenne harmonique des deux indicateurs précédents. Cela permet d'avoir un indicateur adapté quelle que soit la stratégie de la Banque.

5.2 Modèles de classification utilisés

Pour sélectionner nos différents modèles de classification, nous nous sommes appuyés sur le schéma ci-dessous (partie rouge) :



5.2.1 Premiers modèles de classification sélectionnés

Comme évoqué plus haut, nous avons, dans un premier temps, testé différents modèles de classification assez abordables tels que les K plus proches voisins (KNeighborsClassifier), les arbres de décision (DecisionTreeClassifier), la régression logistique et les machines à vecteur de support (SVC). Par ailleurs, nous souhaitions également combiner les résultats de ces modèles afin de créer un modèle plus efficace, qui améliore les performances des précédents. C'est la raison pour laquelle nous avions également sélectionné le StackingClassifier en plus des 4 précédents, après avoir essayé le VotingClassifier. Ainsi, les 4 modèles KNeighborsClassifier, DecisionTreeClassifier, LogisticRegression et SVC ont été testés individuellement afin que l'on soit en mesure de désigner le plus performant d'entre eux selon le f1-score : une fois le plus performant sélectionné, il était inséré comme estimateur final au sein du StackingClassifier (méthode final_estimator). Nous avions sélectionné la régression logistique comme estimateur final car c'est le modèle qui présentait les résultats les plus stables. En revanche, les modèles KNeighborsClassifier et DecisionTreeClassifier se sont révélés très instables : les résultats variaient beaucoup dès lors que la fonction était exécutée une nouvelle fois. C'est d'ailleurs la raison pour laquelle nous obtenions certainement des résultats similaires entre les modèles StackingClassifier et LogisticRegression dans la mesure où les modèles KNeighborsClassifier et DecisionTreeClassifier ne présentaient pas de résultats suffisamment stables pour permettre d'améliorer efficacement les résultats de la régression logistique au sein du StackingClassifier. De plus, en raison de la lenteur de son exécution, le modèle de machines à vecteur de support a été éliminé car il ne présentait pas des résultats significativement meilleurs que ceux des autres modèles. Au final, les résultats de ces

modèles se sont avérés quelque peu décevants : on constatait un important décalage entre le f1-score observé pour la modalité « 0 » et celui observé pour la modalité « 1 », en sachant que ces scores avoisinaient les 0.70 mais n'allait pas au-delà. Un RandomizedSearchCV a pourtant été appliqué sur un large éventail de paramètres pour chaque modèle, et ce afin de sélectionner automatiquement la meilleure combinaison de paramètres pour chacun d'entre eux. Malgré tout, les résultats demeuraient tout de même faibles et assez instables pour les deux modèles cités précédemment (KNeighborsClassifier et le DecisionTreeClassifier).

Voici, ci-après, une ébauche des résultats obtenus avec les différents modèles retenus au départ. Nous avons exécuté le code du projet 2 fois afin de vous présenter les résultats de chacun de ces 2 essais pour chaque modèle (hormis le VotingClassifier et le SVC) en fonction de chaque méthode choisie (RandomizedSearchCV et cross_validate).

Note : il convient de préciser que nous présenterons uniquement les résultats obtenus sans sélection de variables et non ceux obtenus avec sélection de variables car cela ne présente aucun intérêt pour ces modèles (la sélection n'étant plus la même pour les nouveaux modèles).

- **Modèle DecisionTreeClassifier essai 1 :**

- **Avec RandomizedSearchCV :**

```
RESULTAT POUR LE MODELE DecisionTreeClassifier() :
```

	precision	recall	f1-score	support
0	0.72	0.63	0.67	1202
1	0.62	0.71	0.66	1024
accuracy			0.67	2226
macro avg	0.67	0.67	0.67	2226
weighted avg	0.67	0.67	0.67	2226

```
Meilleurs paramètres retenus : {'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': 7, 'criterion': 'entropy'}
```

- **Avec cross_validate :**

```
DECISION TREE CLASSIFIER :
```

```
Accuracy moyenne : 0.6570793553472891
```

```
Accuracy Ecart-type : 0.014340661227891174
```

```
F1 moyenne : 0.6632158457945223
```

```
F1 Ecart-type : 0.018559360176335304
```

- Modèle DecisionTreeClassifier essai 2 :

- Avec RandomizedSearchCV :

```
RESULTAT POUR LE MODELE DecisionTreeClassifier() :
```

	precision	recall	f1-score	support
0	0.67	0.81	0.73	1181
1	0.72	0.55	0.62	1045
accuracy			0.69	2226
macro avg	0.70	0.68	0.68	2226
weighted avg	0.69	0.69	0.68	2226

```
Meilleurs paramètres retenus : {'min_samples_split': 6, 'min_samples_leaf': 4, 'max_depth': 9, 'criterion': 'gini'}
```

- Avec cross_validate :

```
DECISION TREE CLASSIFIER :
```

```
Accuracy moyenne : 0.6769994058229353
Accuracy Ecart-type : 0.016900147174063483
F1 moyenne : 0.6814900517732373
F1 Ecart-type : 0.01594418071021155
```

- Modèle KNeighborsClassifier essai 1 :

- Avec RandomizedSearchCV :

```
RESULTAT POUR LE MODELE KNeighborsClassifier() :
```

	precision	recall	f1-score	support
0	0.68	0.71	0.70	1202
1	0.64	0.62	0.63	1024
accuracy			0.67	2226
macro avg	0.66	0.66	0.66	2226
weighted avg	0.67	0.67	0.67	2226

```
Meilleurs paramètres retenus : {'n_neighbors': 9, 'metric': 'manhattan'}
```

- Avec cross_validate :

```
KNN :  
Accuracy moyenne : 0.6505325622455894  
Accuracy Ecart-type : 0.01689392189274582  
F1 moyenne : 0.63925473333843  
F1 Ecart-type : 0.019468520966006993
```

- Modèle KNeighborsClassifier essai 2 :

- Avec RandomizedSearchCV :

```
RESULTAT POUR LE MODELE KNeighborsClassifier() :
```

	precision	recall	f1-score	support
0	0.66	0.72	0.69	1134
1	0.68	0.63	0.65	1094
accuracy			0.67	2228
macro avg	0.67	0.67	0.67	2228
weighted avg	0.67	0.67	0.67	2228

```
Meilleurs paramètres retenus : {'n_neighbors': 9, 'metric': 'manhattan'}
```

- Avec cross_validate :

```
KNN :  
Accuracy moyenne : 0.6603550454713247  
Accuracy Ecart-type : 0.014738978233629338  
F1 moyenne : 0.6516223826762371  
F1 Ecart-type : 0.021564904826307244
```

- Modèle LogisticRegression essai 1 :

- Avec RandomizedSearchCV :

```
RESULTAT POUR LE MODELE LogisticRegression() :
```

	precision	recall	f1-score	support
0	0.72	0.72	0.72	1202
1	0.67	0.67	0.67	1024
accuracy			0.70	2226
macro avg	0.70	0.70	0.70	2226
weighted avg	0.70	0.70	0.70	2226

```
Meilleurs paramètres retenus : {'C': 1}
```

- **Avec cross_validate :**

```
LOGISTIC REGRESSION :
```

```
Accuracy moyenne : 0.6709296616771849
Accuracy Ecart-type : 0.013056543565101856
F1 moyenne : 0.6625420009988091
F1 Ecart-type : 0.011952700208987749
```

- **Modèle LogisticRegression essai 2 :**

- **Avec RandomizedSearchCV :**

```
RESULTAT POUR LE MODELE LogisticRegression() :
```

	precision	recall	f1-score	support
0	0.71	0.70	0.70	1181
1	0.66	0.67	0.67	1045
accuracy			0.69	2226
macro avg	0.69	0.69	0.69	2226
weighted avg	0.69	0.69	0.69	2226

```
Meilleurs paramètres retenus : {'C': 3}
```

- **Avec cross_validate :**

```
LOGISTIC REGRESSION :  
Accuracy moyenne : 0.6758213355272179  
Accuracy Ecart-type : 0.010390714941399395  
F1 moyenne : 0.6671867551360652  
F1 Ecart-type : 0.010602733768185655
```

- **Modèle StackingClassifier essai 1 :**

- **Avec RandomizedSearchCV :**

```
RESULTAT POUR LE MODELE StackingClassifier avec RandomizedSearchCV (sans sélection variables) :
```

	precision	recall	f1-score	support
0	0.70	0.75	0.72	1202
1	0.68	0.63	0.65	1024
accuracy			0.69	2226
macro avg	0.69	0.69	0.69	2226
weighted avg	0.69	0.69	0.69	2226

Meilleurs paramètres retenus :

```
{'log_C': 5, 'knn_n_neighbors': 1, 'knn_metric': 'manhattan', 'arbre_min_samples_split': 9, 'arbre_min_samples_leaf': 1, 'arbre_max_depth': 7, 'arbre_criterion': 'entropy'}
```

- **Avec cross_validate :**

```
Accuracy moyenne : 0.6950893865565201  
Accuracy Ecart-type : 0.014193918836751118  
F1 moyenne : 0.6880719469818073  
F1 Ecart-type : 0.016806148805721257
```

- **Modèle StackingClassifier essai 2 :**

- **Avec RandomizedSearchCV :**

```

RESULTAT POUR LE MODELE StackingClassifier avec RandomizedSearchCV (sans sélection variables) :

      precision    recall   f1-score   support

          0       0.70      0.76      0.73     1181
          1       0.70      0.64      0.67     1045

   accuracy                           0.70      2226
macro avg       0.70      0.70      0.70      2226
weighted avg    0.70      0.70      0.70      2226

Meilleurs paramètres retenus :

{'log_C': 1, 'knn_n_neighbors': 6, 'knn_metric': 'chebyshev', 'arbre_min_samples_split': 7, 'arbre_min_samples_leaf': 2, 'arbre_max_depth': 6, 'arbre_criterion': 'entropy'}

```

- **Avec cross_validate :**

```

Accuracy moyenne : 0.7005141916906623
Accuracy Ecart-type : 0.016397109800411955
F1 moyenne : 0.6925110439382353
F1 Ecart-type : 0.014400485065585073

```

5.2.2 Modèles de classification définitifs

Finalement, nous avons décidé de nous tourner vers des modèles d'ensemble afin de maximiser nos résultats, tout en conservant la régression logistique pour la notoriété dont elle jouit au sein du milieu bancaire. Enfin, dans l'objectif de maximiser encore davantage les résultats de nos modèles, et particulièrement le f1_score, nous avons également décidé de conserver le StackingClassifier qui, comme on l'a vu, permet de combiner les différents modèles et d'améliorer au fur et à mesure les performances du modèle suivant. Ainsi, les modèles que nous avons sélectionnés sont :

- LogisticRegression ;
- RandomForestClassifier ;
- GradientBoostingClassifier ;
- StackingClassifier.

Nous allons ci-après présenter les résultats obtenus en utilisant RandomizedSearchCV pour chacun d'eux, avec et sans sélection de variables, afin de démontrer le faible impact de la suppression des variables « duration », « poutcome_other », ainsi que celles ayant pour suffixe « _unknown ». Pour le StackingClassifier, il est important de préciser que nous allons exceptionnellement présenter les résultats avec et sans RandomizedSearchCV : dans le cas où RandomizedSearchCV n'a pas été utilisé, nous avons instancié les différents classificateurs avec les meilleurs paramètres choisis pour chaque modèle (lorsque ces derniers ont été testés individuellement).

Note : nous avons supprimé la validation croisée avec la fonction `cross_validate` pour tous les modèles hormis le `StackingClassifier` car elle présentait des résultats proches de ceux obtenus avec `RandomizedSearchCV` dans lequel la validation croisée est également appliquée. Nous avons utilisé les deux méthodes afin de mesurer leur écart et/ou leur ressemblance.

- **Modèle LogisticRegression :**

- **Sans sélection de variables :**

```
RESULTATS POUR LE MODELE LogisticRegression() sans sélection de variables :
```

	precision	recall	f1-score	support
0	0.72	0.70	0.71	1155
1	0.68	0.70	0.69	1073
accuracy			0.70	2228
macro avg	0.70	0.70	0.70	2228
weighted avg	0.70	0.70	0.70	2228

```
Meilleurs paramètres retenus : {'C': 0.9}
```

- **Avec sélection de variables :**

```
RESULTATS POUR LE MODELE LogisticRegression() avec sélection de variables :
```

	precision	recall	f1-score	support
0	0.70	0.69	0.69	1185
1	0.65	0.66	0.66	1043
accuracy			0.68	2228
macro avg	0.67	0.67	0.67	2228
weighted avg	0.68	0.68	0.68	2228

```
Meilleurs paramètres retenus : {'C': 0.1}
```

Comme attendu, les résultats sont légèrement moins bons après avoir retiré les variables mentionnées ci-dessus de notre jeu de données. La précision, le rappel et, par conséquent, le f1_score ont été diminués aussi bien pour la modalité « 0 » que pour la modalité « 1 » de la variable cible. Cependant, cette suppression a malgré tout eu un

faible impact sur les résultats qui n'ont été diminués que de 0.04 points tout au plus. Ainsi, on peut affirmer que la stratégie de suppression des variables douteuses de notre jeu de données a malgré tout été efficace dans ce modèle.

De manière plus précise, on obtient de meilleurs scores pour la modalité « 0 » avec les 3 métriques mentionnées plus haut :

- Pour la précision, cela signifie que le modèle identifie mieux les clients intérressés par le contrat de dépôt à terme (modalité « 0 ») que ceux qui le sont (modalité « 1 ») : effectivement, la précision constitue le taux de vrais positifs sur le total des positifs (vrais et faux positifs confondus) dans le cas de la modalité « 1 » et le taux de vrais négatifs sur le total des négatifs (vrais et faux négatifs confondus) dans le cas de la modalité « 0 ». Ainsi, dans le cas où ce modèle est appliqué à notre jeu de données, la banque minimiserait davantage le nombre d'appels émis vers des clients intérressés qu'elle ne maximiserait le nombre d'appels émis vers des clients intéressés.
- Concernant le rappel, ce dernier est encore plus précis que la précision puisqu'il prend en compte les cas d'erreur de prédiction. En effet, le rappel de la modalité « 0 » constitue le taux de vrais négatifs sur le total des véritables négatifs (vrais négatifs et faux positifs) tandis que le rappel de la modalité « 1 » constitue le taux de vrais positifs sur le total des véritables positifs (vrais positifs et faux négatifs). Il permet de nous assurer que les observations ont été correctement classifiées par le modèle. Dans le cas présent, il est supérieur pour la modalité « 0 » : cela signifie que le modèle détecte mieux les observations réellement négatives que celles qui sont réellement positives. Ainsi, le taux d'erreur de prédiction du modèle est plus élevé pour les clients ayant prétendument souscrit au dépôt à terme que pour les autres.
- Evidemment, le f1_score ne pouvait être qu'inférieur pour la modalité « 1 » puisque ce dernier constitue une moyenne harmonique des 2 métriques précédentes.

- **Modèle RandomForestClassifier :**

- **Sans sélection de variables :**

```
RESULTATS POUR LE MODELE RandomForestClassifier() sans sélection de variables :
```

	precision	recall	f1-score	support
0	0.71	0.77	0.74	1155
1	0.73	0.67	0.70	1073
accuracy			0.72	2228
macro avg	0.72	0.72	0.72	2228
weighted avg	0.72	0.72	0.72	2228

```
Meilleurs paramètres retenus : {'n_estimators': 900, 'max_features': 0.5, 'max_depth': 6, 'bootstrap': True}
```

➤ Avec sélection de variables :

```
RESULTATS POUR LE MODELE RandomForestClassifier() avec sélection de variables :
```

	precision	recall	f1-score	support
0	0.72	0.81	0.76	1185
1	0.75	0.64	0.69	1043
accuracy			0.73	2228
macro avg	0.73	0.72	0.72	2228
weighted avg	0.73	0.73	0.73	2228

```
Meilleurs paramètres retenus : {'n_estimators': 800, 'max_features': 0.5, 'max_depth': 6, 'bootstrap': False}
```

Dans ce modèle, les résultats sont plus nuancés que précédemment : seul le rappel et le f1_score ont été diminués pour la modalité « 1 » uniquement, le reste des résultats ayant augmenté avec la suppression des variables susmentionnées. Ainsi, la précision est supérieure à celle du modèle précédent : elle est de 0.72 pour la modalité « 0 » et de 0.75 pour la modalité « 1 ». Cela signifie que le modèle identifie mieux les clients réellement intéressés par le contrat de dépôt à terme (modalité « 1 ») que ceux qui ne le sont pas (modalité « 0 »). Ainsi, dans le cas où ce modèle est appliqué à notre jeu de données, la banque maximiserait davantage le nombre d'appels émis vers des clients intéressés qu'elle ne minimiseraient ceux émis vers des clients inintéressés. De plus, le rappel est lui aussi supérieur à celui obtenu dans le modèle précédent pour la modalité « 0 » : il est de 0.81 pour celle-ci et de 0.64 pour la modalité « 1 ». Autrement dit, il a été augmenté de 0.04 points dans le cas de la modalité « 0 » et a été diminué de 0.03 points dans le cas de la modalité « 1 ». Cela signifie que le taux d'erreur de prédiction est plus élevé pour les clients ayant prétendument souscrit au dépôt à terme que pour les autres. Enfin, le f1_score, quant à lui, a été augmenté de 0.02 points pour la modalité « 0 » et a été diminué de 0.01 points dans le cas de la modalité « 1 » : on constate des résultats approximativement similaires à ceux du modèle précédent antérieur à la sélection de variables avec une légère amélioration des résultats pour la modalité « 0 » (+ 0.02 points) et une légère détérioration des résultats pour la modalité « 1 » (- 0.01 points). Notre conclusion est donc la même que précédemment : malgré quelques changements, cette suppression des variables susmentionnées n'a eu qu'un

faible impact sur les résultats. Ainsi, on peut affirmer que la stratégie de suppression des variables douteuses de notre jeu de données a malgré tout été efficace.

- **Modèle GradientBoostingClassifier :**

- **Sans sélection de variables :**

```
RESULTATS POUR LE MODELE GradientBoostingClassifier() sans sélection de variables :
```

	precision	recall	f1-score	support
0	0.71	0.74	0.73	1155
1	0.71	0.67	0.69	1073
accuracy			0.71	2228
macro avg	0.71	0.71	0.71	2228
weighted avg	0.71	0.71	0.71	2228

```
Meilleurs paramètres retenus : {'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.5}
```

- **Avec sélection de variables :**

```
RESULTATS POUR LE MODELE GradientBoostingClassifier() avec sélection de variables :
```

	precision	recall	f1-score	support
0	0.72	0.73	0.72	1185
1	0.69	0.67	0.68	1043
accuracy			0.70	2228
macro avg	0.70	0.70	0.70	2228
weighted avg	0.70	0.70	0.70	2228

```
Meilleurs paramètres retenus : {'n_estimators': 900, 'max_depth': 6, 'learning_rate': 0.1}
```

Ici aussi les résultats sont plus nuancés que pour la LogisticRegression mais les conclusions sont approximativement similaires à celle-ci :

- Concernant la précision, on constate une très légère amélioration des résultats pour la modalité « 0 » (+0.01 points) combinée à une très légère baisse de ces derniers pour la modalité « 1 » (-0.02 points) qui le rend inférieur au précédent : cela signifie que le modèle identifie mieux les clients intéressés par le contrat de dépôt à terme (modalité « 0 ») que ceux qui ne le sont pas (modalité « 1 »). Ainsi, dans le cas où ce modèle

est appliqué à notre jeu de données, la banque minimisera davantage le nombre d'appels émis vers des clients inintéressés qu'elle ne maximisera le nombre d'appels émis vers des clients intéressés.

- Concernant le rappel, celui de la modalité « 0 » est supérieur à celui de la modalité « 1 » bien qu'il ait été stabilisé pour la modalité « 1 » et diminué de 0.01 points pour la modalité « 0 » : cela signifie que le modèle détecte mieux les observations réellement négatives que celles qui sont réellement positives. Ainsi, le taux d'erreur de prédiction du modèle est plus élevé pour les clients ayant prétendument souscrit au dépôt à terme que pour les autres.
- Au final, comme on pouvait s'y attendre, le f1_score a été très légèrement diminué pour les deux modalités « 0 » et « 1 » (- 0.01 points).

Par conséquent, notre conclusion est à nouveau la même concernant le faible impact de la suppression des variables : cette nécessaire suppression a été efficace dans ce modèle également puisqu'elle n'a eu qu'une faible incidence sur les résultats de ce modèle.

- **Modèle StackingClassifier :**

- **Sans sélection de variables :**

- **Sans RandomizedSearchCV :**

RESULTATS POUR LE MODELE StackingClassifier sans RandomizedSearchCV (sans sélection de variables) :

	precision	recall	f1-score	support
0	0.72	0.78	0.75	1155
1	0.74	0.67	0.70	1073
accuracy			0.73	2228
macro avg	0.73	0.73	0.73	2228
weighted avg	0.73	0.73	0.73	2228

- **Avec RandomizedSearchCV :**

RESULTATS POUR LE MODELE StackingClassifier avec RandomizedSearchCV (sans sélection de variables) :

	precision	recall	f1-score	support
0	0.71	0.85	0.77	1155
1	0.79	0.62	0.70	1073
accuracy			0.74	2228
macro avg	0.75	0.74	0.73	2228
weighted avg	0.75	0.74	0.74	2228

Meilleurs paramètres retenus :

```
{'RANDOM_FOREST__n_estimators': 600, 'RANDOM_FOREST__max_features': 0.5, 'RANDOM_FOREST__max_depth': 6, 'RANDOM_FOREST__bootstrap': False, 'LOGISTIC__C': 0.6, 'GRADIENT_BOOSTING__n_estimators': 600, 'GRADIENT_BOOSTING__max_depth': 5, 'GRADIENT_BOOSTING__learning_rate': 0.7}
```

➤ Avec sélection de variables :

- Sans RandomizedSearchCV :

RESULTATS POUR LE MODELE StackingClassifier sans RandomizedSearchCV (avec sélection de variables) :

	precision	recall	f1-score	support
0	0.70	0.74	0.72	1185
1	0.68	0.65	0.67	1043
accuracy			0.70	2228
macro avg	0.69	0.69	0.69	2228
weighted avg	0.69	0.70	0.69	2228

- Avec RandomizedSearchCV :

RESULTATS POUR LE MODELE StackingClassifier avec RandomizedSearchCV (avec sélection de variables) :

	precision	recall	f1-score	support
0	0.72	0.78	0.75	1185
1	0.73	0.65	0.69	1043
accuracy			0.72	2228
macro avg	0.72	0.72	0.72	2228
weighted avg	0.72	0.72	0.72	2228

Meilleurs paramètres retenus :

```
{'RANDOM_FOREST__n_estimators': 1000, 'RANDOM_FOREST__max_features': 0.7, 'RANDOM_FOREST__max_depth': 6, 'RANDOM_FOREST__bootstrap': False, 'LOGISTIC__C': 0.6, 'GRADIENT_BOOSTING__n_estimators': 100, 'GRADIENT_BOOSTING__max_depth': 6, 'GRADIENT_BOOSTING__learning_rate': 0.1}
```

Ici, globalement, on observe une amélioration des résultats lorsque RandomizedSearchCV est appliqué à notre modèle, et ce que l'on supprime ou non les variables de ce dernier. Les seules exceptions résident dans le modèle antérieur à la suppression des variables : la précision de la modalité « 0 » a été très légèrement diminuée (- 0.01 points) tout comme le rappel de la modalité « 1 » a été également diminué (- 0.05 points) avec

RandomizedSearchCV. Cependant, cela ne change rien aux conclusions : les proportions entre les modalités restent globalement les mêmes sauf pour la précision du modèle après suppression des variables puisque cette dernière devient supérieure pour la modalité « 1 » alors qu'elle était inférieure à la modalité « 0 » avant suppression des variables. Nous allons donc comparer les résultats du modèle avant et après suppression des variables en se concentrant uniquement sur le cas où RandomizedSearchCV est utilisé. Effectivement, dans ce cas, les proportions entre les modalités « 0 » et « 1 » sont les mêmes avant et après la suppression des variables :

- Malgré une assez forte diminution de la modalité « 1 » (- 0.06 points) et une amélioration de la modalité « 0 » (+ 0.01 points), la précision de la modalité « 0 » est inférieure à celle de la modalité « 1 » : cela signifie que le modèle identifie mieux les clients réellement intéressés par le contrat de dépôt à terme (modalité « 1 ») que ceux qui ne le sont pas (modalité « 0 »). Ainsi, dans le cas où ce modèle est appliqué à notre jeu de données, la banque minimiserait moins le nombre d'appels émis vers des clients inintéressés qu'elle ne maximiserait le nombre d'appels émis vers des clients intéressés.
- Malgré une assez forte baisse pour la modalité « 0 » (- 0.07 points) et une amélioration pour la modalité « 1 » (+ 0.03 points), le rappel de la modalité « 0 » demeure supérieur à celui de la modalité « 1 » : cela signifie que le modèle détecte mieux les observations réellement négatives que celles qui sont réellement positives. Ainsi, le taux d'erreur de prédiction du modèle est plus élevé pour les clients ayant prétendument souscrit au dépôt à terme que pour les autres.
- Sans surprise, le f1_score est également similaire en proportion malgré une légère baisse simultanée pour les 2 modalités « 0 » et « 1 » respectivement de 0.02 et 0.01 points.

Ainsi, de même que pour nos 3 modèles précédents, la suppression des variables a eu une faible incidence négative sur les résultats globaux du modèle (f1_score), raison pour laquelle on peut considérer qu'elle a été efficace.

En conclusion, les résultats des modèles définitifs sont nettement supérieurs à ceux des modèles sélectionnés précédemment (KNeighborsClassifier, DecisionTreeClassifier et StackingClassifier). En effet, les modèles définitifs sont plus performants et ont donné de meilleurs résultats que ceux sélectionnés précédemment lorsque l'on compare ces derniers avant la sélection de variables qui a été opérée : dans la mesure où celle-ci est différente entre les deux, ils sont donc incomparables après cette sélection (raison pour laquelle nous n'avons pas présenté les résultats après sélection de variables pour nos anciens modèles). De plus, en termes de choix de variables, nous avons également constaté que cette sélection n'engendrait qu'un faible impact négatif sur les résultats de nos modèles définitifs et qu'elle permettait en contrepartie de se débarrasser des variables douteuses de notre

jeu de données : nous considérons donc que cette sélection a été efficiente. Ainsi, globalement, notre démarche s'est avérée efficace pour améliorer la qualité de notre jeu de données et les résultats de nos modèles.

Dans la partie suivante, nous nous intéresserons à l'interprétabilité globale et locale de nos différents modèles afin de mieux comprendre leurs résultats.

6 Interprétabilité

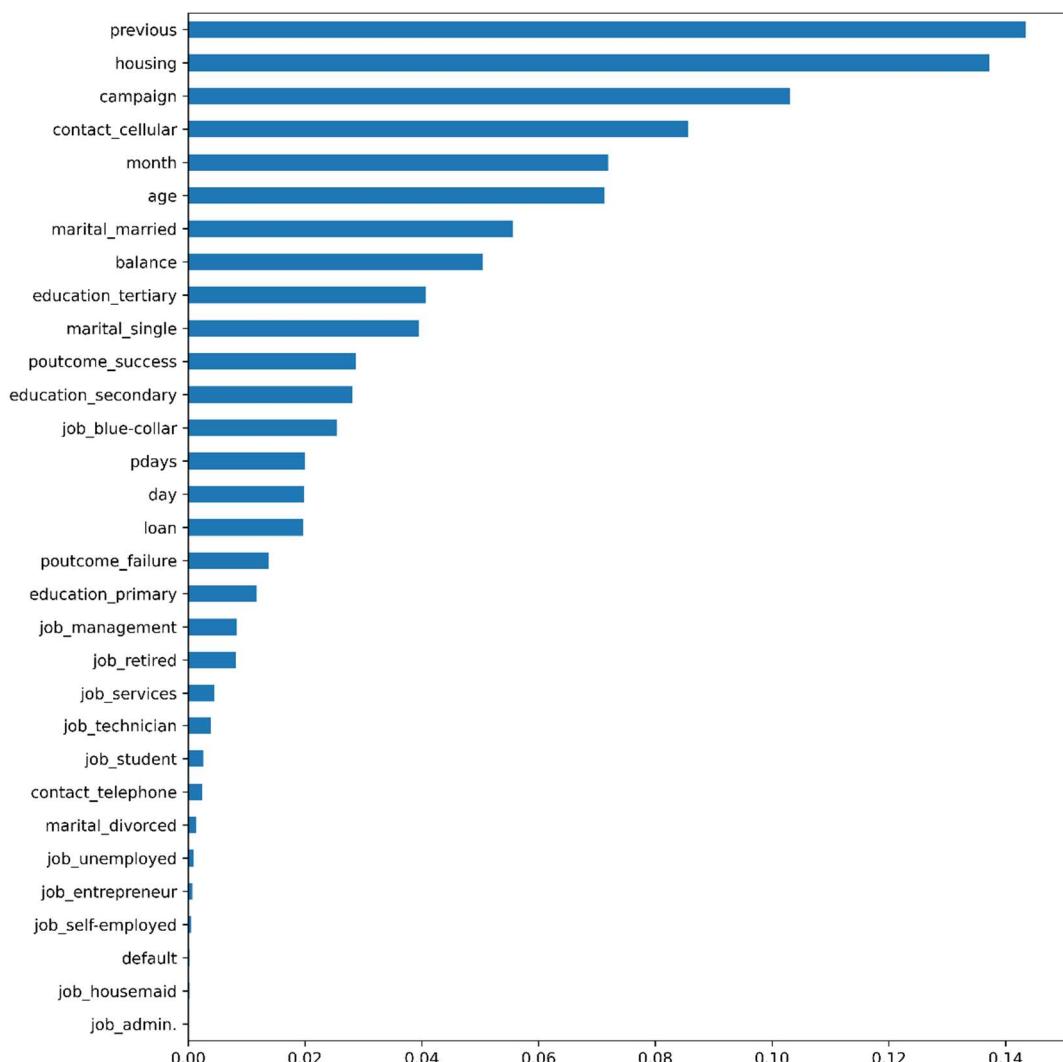
Nous ne nous attarderons pas beaucoup sur l'analyse de l'interprétabilité de nos différents modèles : cette partie sera davantage analysée et exploitée lors de la présentation du projet (Streamlit). Nous nous contenterons ici de présenter et de comparer brièvement l'interprétation globale et locale des modèles que nous avons sélectionnés, ainsi que la dépendance partielle de la variable « age » pour chacun d'eux.

6.1 Interprétabilité globale

Il est important de préciser que nous ne pourrons pas comparer l'interprétabilité globale de chacun de nos modèles avec les graphiques élaborés précédemment relatifs aux méthodes traditionnelles de sélection des variables : en effet, les méthodes traditionnelles faisaient ressortir des graphiques élaborés sans que les variables « contact_unknown », « poutcome_other », « poutcome_unknown », « job_unknown » et « education_unknown » n'aient été supprimées.

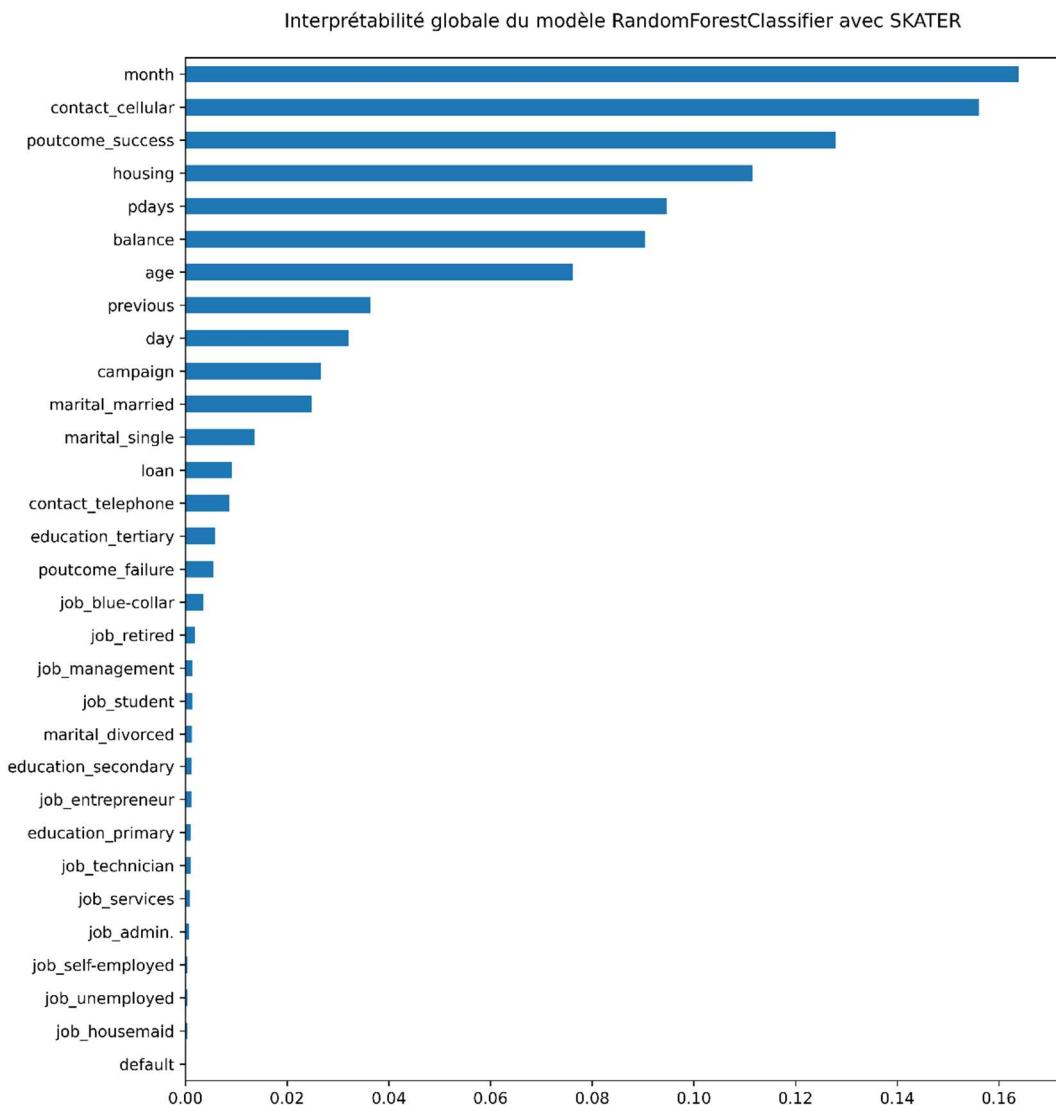
6.1.1 Interprétabilité globale du modèle LogisticRegression

Interprétabilité globale du modèle LogisticRegression avec SKATER



D'après le graphique, on constate que seulement 20 variables sur 31 sont considérées comme significatives par le modèle de régression logistique. Par ordre décroissant d'importance, il s'agit des variables : « previous », « housing », « campaign », « contact_cellular », « month », « age », « marital_married », « balance », « education_tertiary », « marital_single », « poutcome_success », « education_secondary », « job_blue-collar », « pdays », « day », « loan », « poutcome_failure », « education_primary », « job_management » et « job_retired ». Cela est cohérent avec le graphique que nous avions élaboré (dans la partie de sélection des variables) nous permettant d'évaluer la performance du modèle (la courbe des scores) en fonction du nombre de variables sélectionnées selon le modèle de régression logistique appliqué à la RFE. Le score du modèle se stabilisait et n'évoluait pas entre 15 et 20 variables. Cela signifie que, selon le modèle de régression logistique appliqué à la RFE, il n'est pas nécessaire de conserver plus de 15-20 variables pour améliorer la performance du modèle.

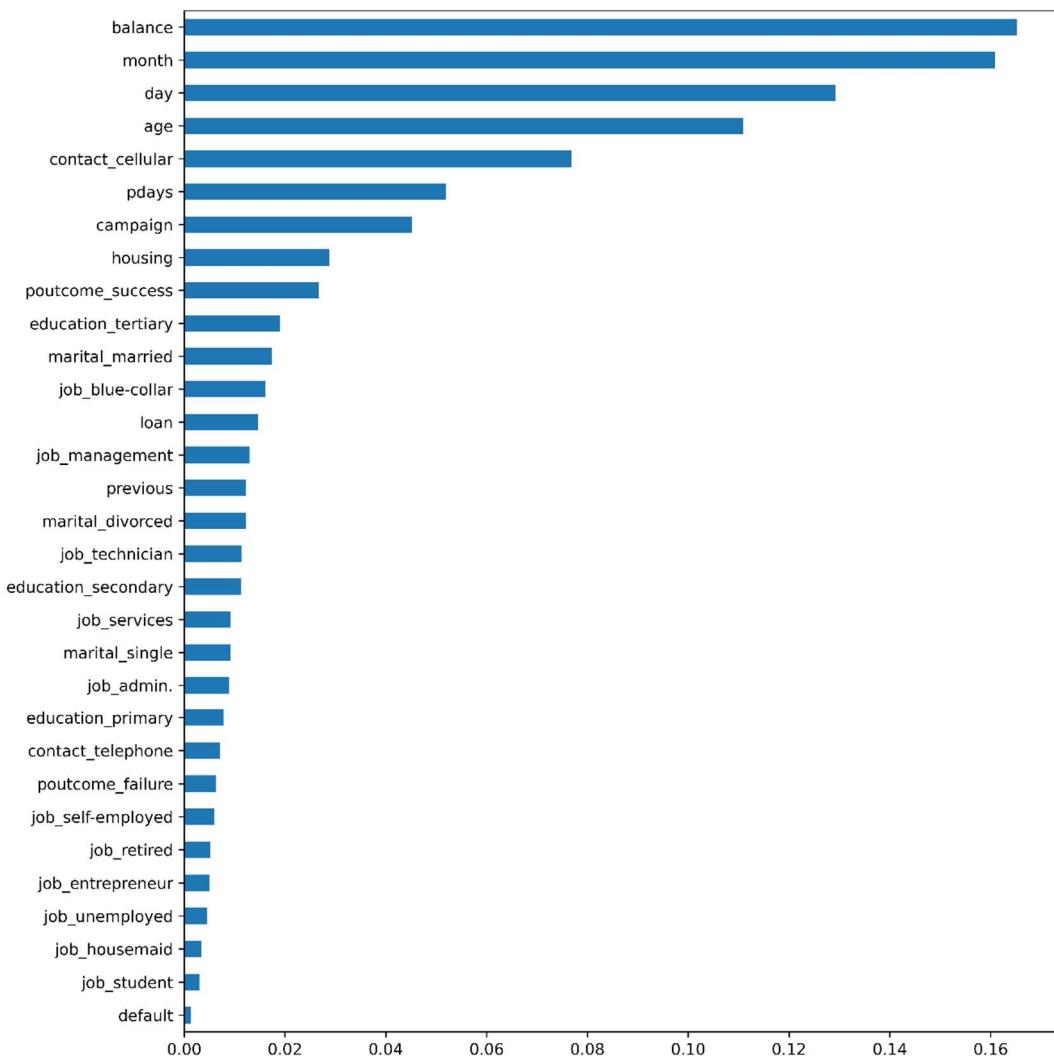
6.1.2 Interprétabilité globale du modèle RandomForestClassifier



En observant le graphique d'interprétabilité globale du modèle de forêts aléatoires, on constate que seules 17 variables sont considérées comme significatives par le modèle. Il s'agit principalement des mêmes variables que celles citées précédemment dans le modèle de régression logistique, bien que l'ordre d'importance ne soit pas le même : « month », « contact_cellular », « poutcome_success », « housing », « pdays », « balance », « age », « previous », « day », « campaign », « marital_married », « marital_single », « loan », « contact_telephone », « education_tertiary », « poutcome_failure » et « job_blue-collar ».

6.1.3 Interprétabilité globale du modèle GradientBoostingClassifier

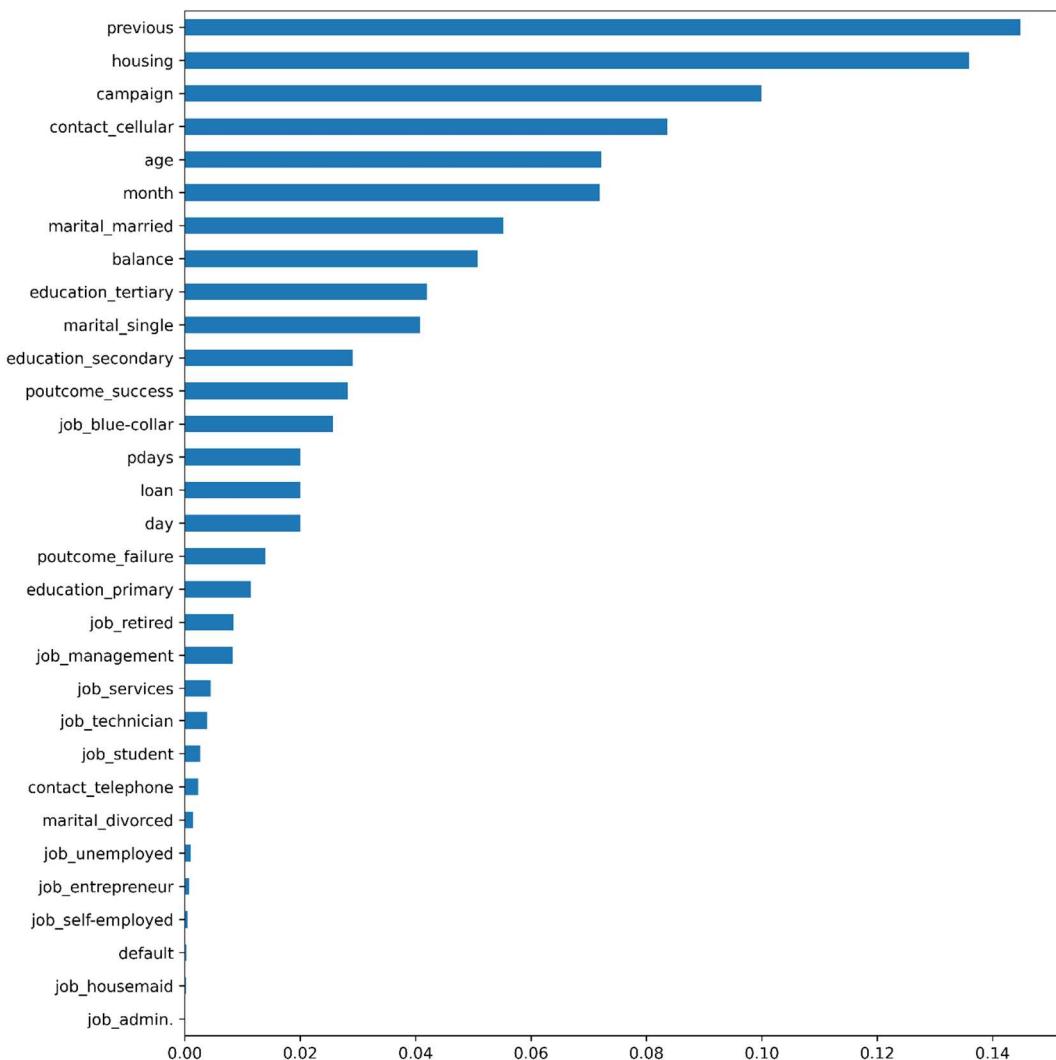
Interprétabilité globale du modèle GradientBoostingClassifier avec SKATER



Selon le modèle GradientBoostingClassifier, la majorité des variables sont considérées comme significatives d'après le graphique, les plus importantes d'entre elles étant les mêmes que pour les modèles précédents dans un ordre de significativité différent : « balance », « month », « day », « age », « contact_cellular », « pdays », « campaign », « housing », « poutcome_success », « education_tertiary », « marital_married », « job_blue-collar », « loan », « previous », « education_secondary » et « marital_single ». A ces variables, on peut ajouter celles n'apparaissant pas comme significatives dans les modèles précédents : « job_management », « marital_divorced », « job_technician », « job_services » et « job_admin ».

6.1.4 Interprétabilité globale du modèle StackingClassifier

Interprétabilité globale du modèle StackingClassifier avec SKATER



Enfin, selon le modèle StackingClassifier, les mêmes variables que celles évoquées précédemment sont considérées comme importantes : « previous », « housing », « campaign », « contact_cellular », « age », « month », « marital_married », « balance », « education_tertiary », « marital_single », « education_secondary », « poutcome_success », « job_blue-collar », « pdays », « loan », « day », « education_primary » et « job_management ». A ces variables, s'ajoutent les suivantes, non présentes dans les modèles précédents : « poutcome_failure » et « job_retired ».

En conclusion, globalement, nous pouvons classer nos 31 variables dans les 3 catégories suivantes selon nos 4 modèles ci-dessus :

- Les variables considérées comme peu importantes : « job_admin », « job_housemaid », « job_self-employed », « job_entrepreneur », « job_unemployed », « job_student », « job_technician », « job_services » et « default ».

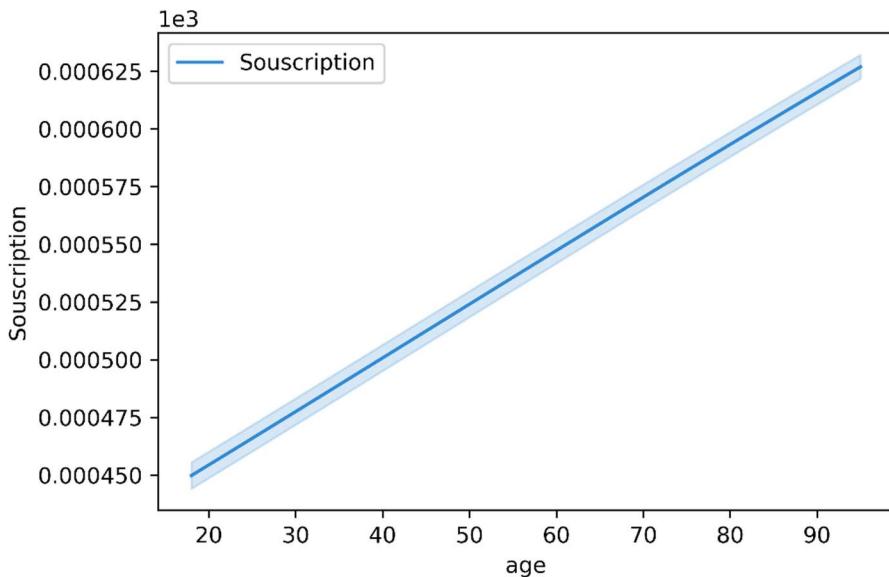
- Celles considérées comme peu importantes par certains modèles et relativement importantes pour d'autres : « education_primary », « education_secondary », « job_management », « marital_divorced », « job_retired », « poutcome_failure », « contact_telephone »,
- Celles considérées comme importantes : « previous », « housing », « campaign », « contact_cellular », « age », « month », « balance », « marital_married », « marital_single », « poutcome_success », « pdays », « loan », « day », « job_blue-collar » et « education_tertiary ».

6.2 Dépendance partielle

La dépendance partielle permet de représenter la manière dont évolue le résultat du modèle en fonction de l'évolution d'une variable donnée, tout en gardant les autres variables du jeu de données constantes. Nous nous contenterons ici de fournir un exemple de dépendance partielle avec la variable « age ».

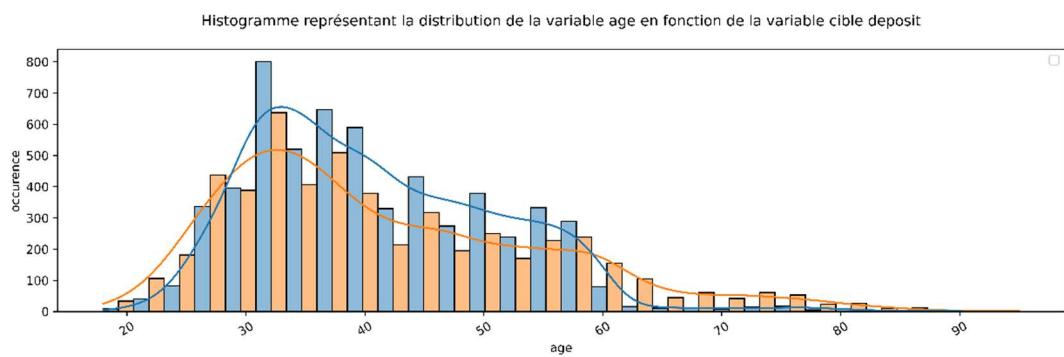
6.2.1 Dépendance partielle de la variable « age » selon le modèle LogisticRegression

Dépendance partielle de la variable age selon le modèle LogisticRegression avec SKATER



Selon le modèle de régression logistique, on constate que le taux de souscription évolue proportionnellement avec l'âge, ce qui, au premier abord, ne paraît pas cohérent avec l'analyse de la distribution que nous avions élaborée : effectivement, nous avions constaté que les clients les plus âgés et les plus jeunes ont, en proportion, un taux de souscription au dépôt à terme supérieur. Or, ici on observe que plus l'âge augmente, plus le taux de souscription

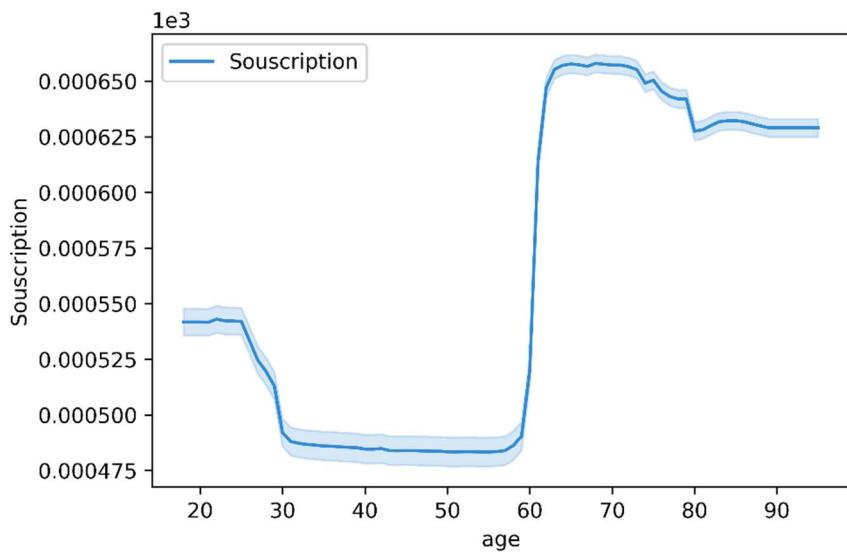
augmente. En réalité, il ne s'agit pas d'une incohérence, c'est simplement que le graphique manque de précision. En effet, lorsque l'on observe l'histogramme ci-dessous, on constate que la courbe orange (qui correspond à la distribution de la variable cible « deposit » en fonction de la variable « age ») est au-dessus de la courbe bleu (correspondant quant à elle à la distribution de la variable « age ») juste avant l'âge de 30 ans, puis se retrouve en dessous de celle-ci entre 30 et 60 ans, puis à nouveau au-dessus d'elle au-delà de 60 ans, mais de manière plus importante qu'elle ne l'était avant 30 ans. En d'autres termes, globalement, le taux de souscription évolue bien de manière croissante avec l'âge, mais le graphique de dépendance partielle ci-dessus ne représente pas précisément ces différentes évolutions.



6.2.2 Dépendance partielle de la variable « age » selon le modèle

RandomForestClassifier

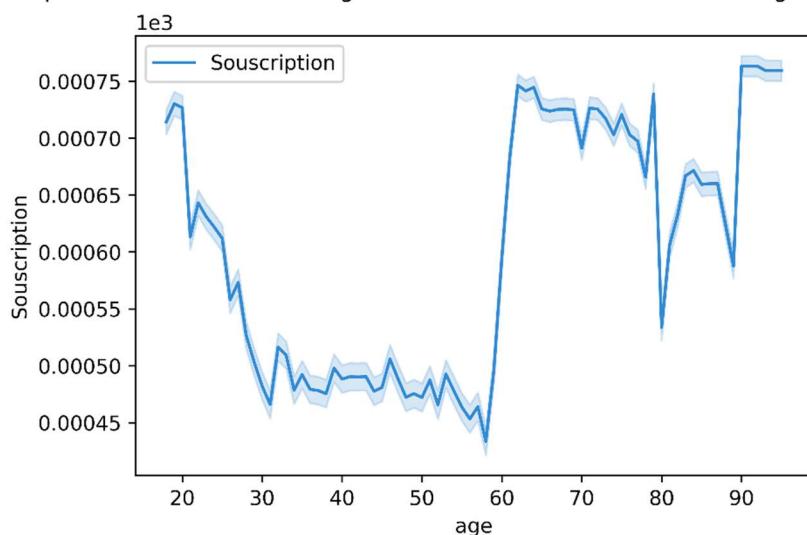
Dépendance partielle de la variable age selon le modèle RandomForestClassifier avec SKATER



Ici, cependant, les différentes évolutions de la variable « age » sont représentées de manière beaucoup plus précise qu'elles ne l'étaient dans le graphique précédent : on voit bien que le taux de souscription est plus élevé avant 30 ans (avec une nette décroissance entre 26 et 30 ans) puis entre 60 et 90 ans (avec une légère décroissance à partir de 75 ans) en comprenant une certaine stabilisation entre 30 et 60 ans. Ainsi, on peut affirmer que le graphique de dépendance partielle appliqué au modèle RandomForestClassifier est beaucoup plus précis que celui appliqué au modèle LogisticRegression.

6.2.3 Dépendance partielle de la variable « age » selon le modèle GradientBoostingClassifier

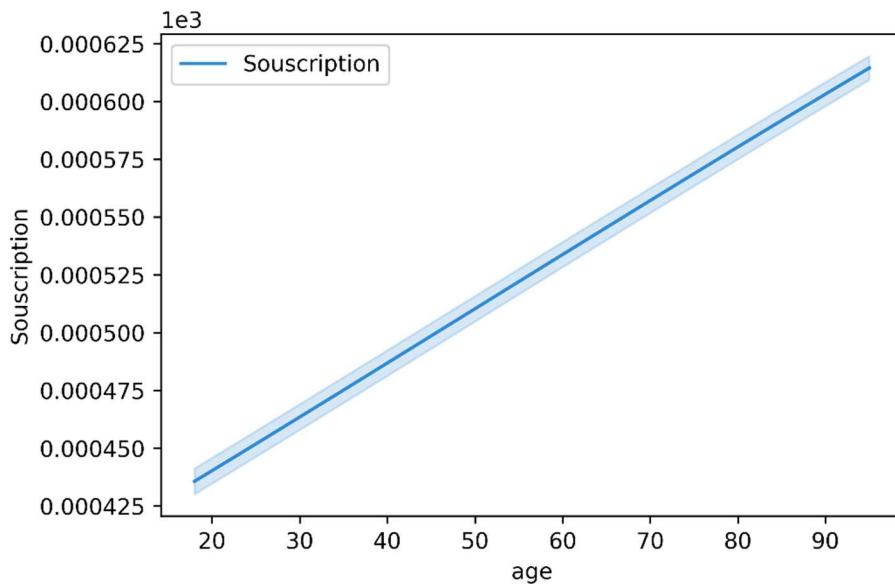
Dépendance partielle de la variable age selon le modèle GradientBoostingClassifier avec SKATER



Le graphique de dépendance partielle du modèle GradientBoostingClassifier est encore plus précis que celui appliqué au RandomForestClassifier : il identifie les différentes tendances internes à l'évolution de la courbe représentant l'évolution du taux de souscription en fonction de la variable « age ». La forme de la courbe reste malgré tout globalement la même que précédemment : on observe que le taux de souscription est élevé avant l'âge de 30 ans et après celui de 60 ans, avec une stabilisation entre les deux.

6.2.4 Dépendance partielle de la variable « age » selon le modèle StackingClassifier

Dépendance partielle de la variable age selon le modèle StackingClassifier avec SKATER



Le graphique de dépendance partielle du modèle StackingClassifier est, quant à lui, similaire à celui appliqué au modèle de LogisticRegression : il nous informe du fait que, globalement, la variable cible « deposit » évolue de manière croissante à la variable « age » sans aucune autre précision.

6.3 Interprétabilité locale

L’interprétabilité locale avec LIME est une méthode indépendante du modèle auquel elle s’applique et qui permet de justifier les décisions du modèle en question : elle ne peut être comparée à l’interprétation globale de celui-ci. De manière plus précise, il s’agit de prendre une ligne au hasard dans notre jeu de données ou de constituer un ID fictif en sélectionnant à notre guise une valeur pour les différentes variables qui composent notre jeu de données, puis de lui appliquer la méthode LIME afin d’expliquer, localement, pourquoi le modèle a décidé de classer cet ID dans telle ou telle autre modalité de la variable cible. Ici, nous allons montrer un exemple d’interprétation locale pour chacun de nos 4 modèles en sélectionnant pour eux le même ID client parmi tous ceux présent dans notre jeu de données. Cela nous permettra de comparer nos modèles entre eux.

Note : l’ID client sélectionné est la ligne 701 du tableau initial, ce qui correspond à l’ID 2050 de notre jeu de test.

Le client choisi présente les informations suivantes :

ID	701
age	30
default	0
balance	5389

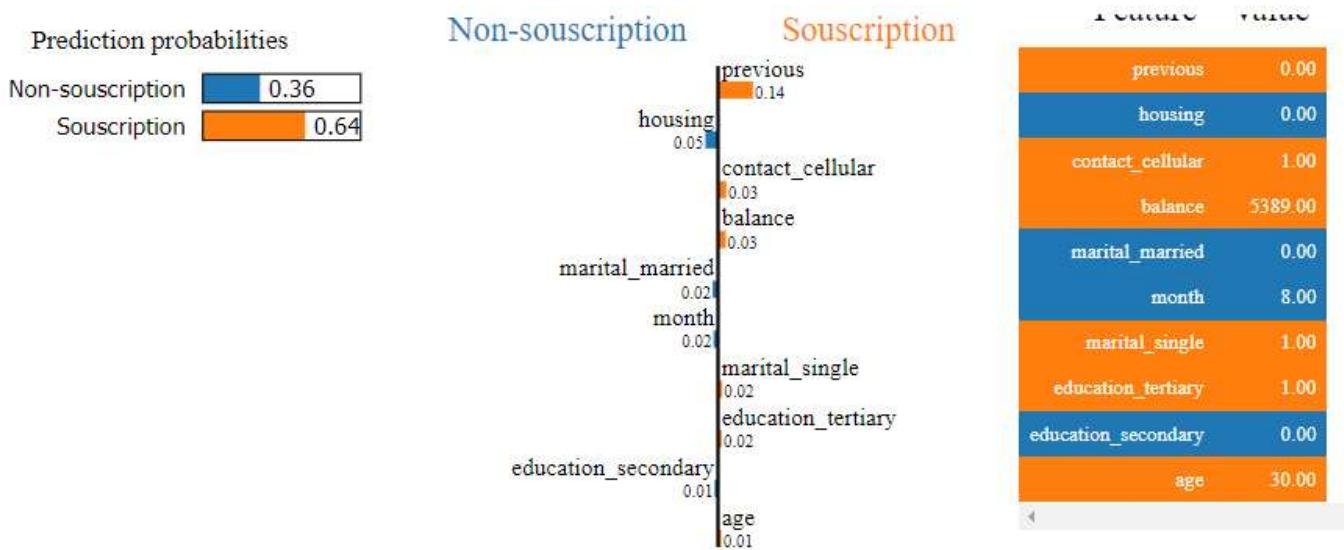
housing	0
loan	0
day	6
month	8
campaign	3
pdays	-1
previous	0
job_admin.	0
job_blue-collar	0
job_entrepreneur	0
job_housemaid	0
job_management	1
job_retired	0
job_self-employed	0
job_services	0
job_student	0
job_technician	0
job_unemployed	0
marital_divorced	0
marital_married	0
marital_single	1
education_primary	0
education_secondary	0
education_tertiary	1
contact_cellular	1
contact_telephone	0
poutcome_failure	0
poutcome_success	0

Dans un premier temps, on s'aperçoit qu'il s'agit d'un client n'ayant pas participé à la campagne marketing précédente non seulement parce que « pdays » a pour modalité « -1 », mais aussi parce que « poutcome_success » et « poutcome_failure » ont la même modalité « 0 » (dans le cas contraire, au moins un des 2 devrait avoir la modalité « 1 »). Néanmoins, il a reçu 3 appels sur son téléphone mobile au cours de cette campagne, le dernier datant du 6 août. De plus, il a 30 ans, un niveau d'études plutôt élevé et travaille dans le « management » (« job_management » = « 1 »). Par ailleurs, il est célibataire (« marital_single » = « 1 »), n'a pas effectué de crédit à la consommation ni immobilier et n'a jamais fait défaut à un crédit. Enfin, il possède 5389 euros sur son compte bancaire.

6.3.1 Interprétabilité locale du modèle LogisticRegression

Interprétation locale du modèle LogisticRegression selon l index : 2050

Libellé réel : 1
Prédit : 1

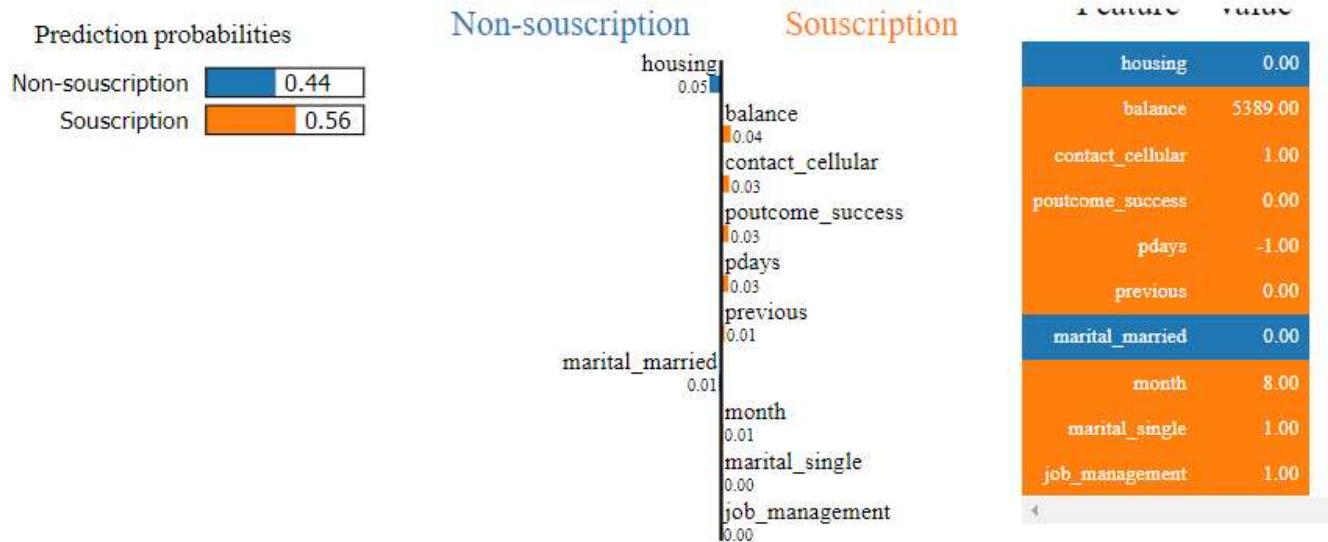


On constate que le modèle de régression logistique est plutôt confiant sur le fait de savoir si cet ID souscrira à la prochaine campagne marketing de la banque : il établit la probabilité de souscription à 64%. Par ailleurs, on observe qu'un nombre de variables approximativement égal agissent à la fois positivement et négativement sur le résultat du modèle local : en effet, 4 variables ayant un taux de significativité élevé agissent négativement sur le modèle local (« housing », « month », « marital_married » et « education_secondary ») tandis que 6 variables ayant également un taux de significativité élevé agissent positivement sur le modèle local (« previous », « contact_cellular », « age », « education_tertiary », « marital_single » et « balance »). Au final, le modèle s'est avéré efficace puisque la classe prédite était bien la bonne (classe « 1 » = souscription au dépôt à terme) : au vu des caractéristiques du client qui présente davantage de caractéristiques agissant positivement sur le modèle, il est normal que la classe prédite soit la classe « 1 ». A titre d'exemple, le client n'a pas de crédit immobilier, n'est pas marié et n'a pas arrêté ses études au niveau secondaire, 3 des 4 caractéristiques les plus importantes agissant négativement sur le modèle local. En revanche, il est célibataire, a effectué des études supérieures, a été contacté sur son téléphone mobile et présente un solde de compte bancaire assez intéressant, soit 4 des 6 caractéristiques les plus importantes agissant positivement sur le modèle local.

6.3.2 Interprétabilité locale du modèle RandomForestClassifier

```
Interprétation locale du modèle RandomForestClassifier selon l index : 2050
```

Libellé réel : 1
Prédit : 1

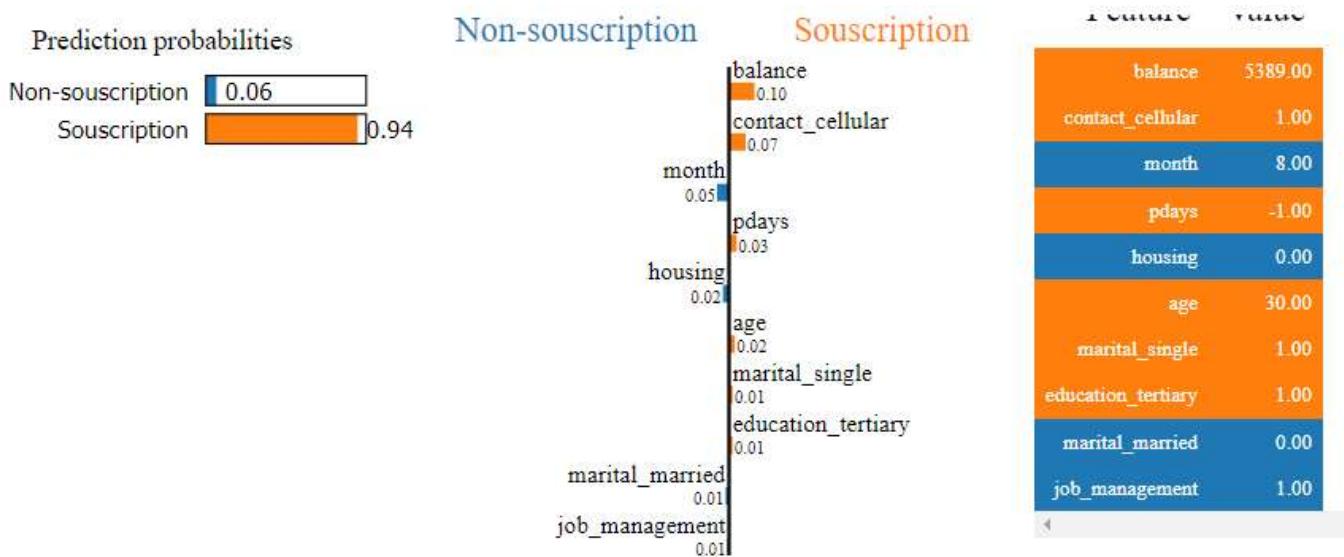


Le modèle RandomForestClassifier, quant à lui, est bien moins confiant que ne l'est le précédent : il évalue la prédiction de souscription à seulement 56%, un taux assez proche de celui d'une prédiction aléatoire. Cela peut s'expliquer par le fait que le client ne présente que peu de caractéristiques les plus importantes agissant positivement sur le modèle malgré la conséquente liste de caractéristiques significativement positives que l'on peut observer dans le tableau de droite (variables de couleur orange). Effectivement, il n'a pas été contacté lors de la campagne précédente (`pdays = « -1 »` et `« previous » = « 0 »`), le résultat de la campagne précédente ne se soldant donc pas par un succès (`« poutcome_success » = « 0 »`) : il lui manque 3 caractéristiques positives les plus significatives sur 8. Malgré tout, la classe prédictive par le modèle est bien la bonne.

6.3.3 Interprétabilité locale du modèle GradientBoostingClassifier

```
Interprétation locale du modèle GradientBoostingClassifier selon l index : 2050
```

Libellé réel : 1
Prédit : 1

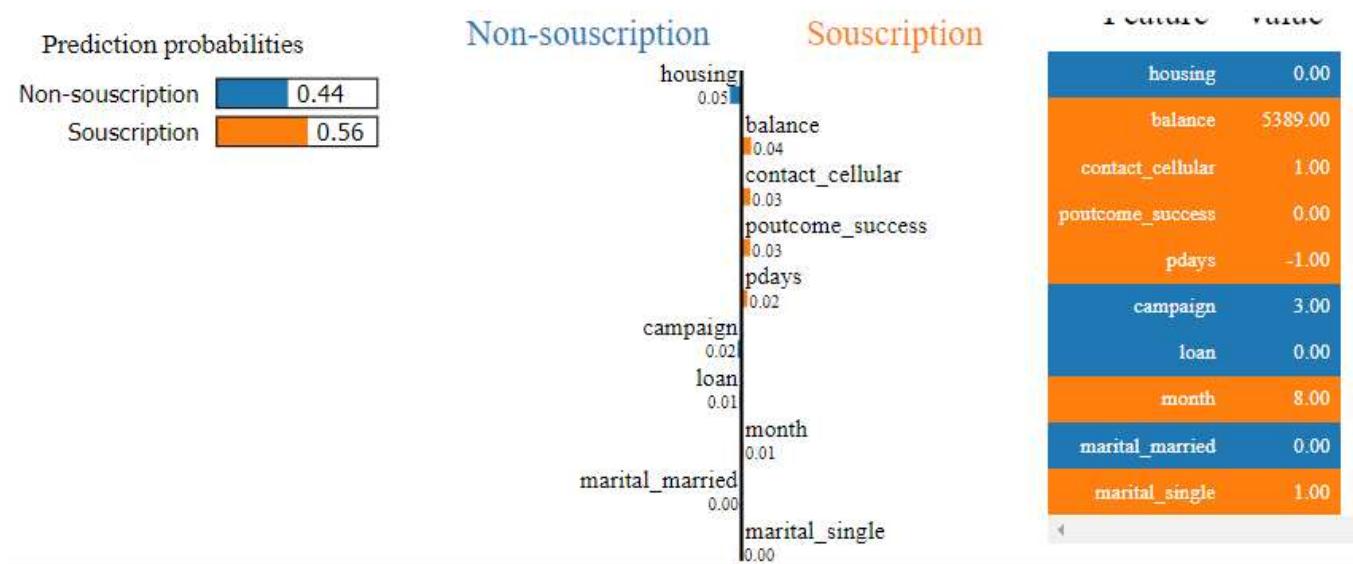


En revanche, le GradientBoostingClassifier est, quant à lui, extrêmement confiant quant à la classification du client concerné : en effet, son taux de prédiction pour la classe positive est de 94%. Autrement dit, le modèle est à 94% certain que le client souscrira au dépôt à terme. Lorsque l'on observe les variables les plus significatives du modèle local, l'explication nous paraît évidente : une seule caractéristique de celles les plus significativement positives manque à l'appel (« pdays » = « -1 »). Par ailleurs, seule une caractéristique de celles les plus significativement négatives est présente (« job_management » = « 1 »). Sans surprise, la prédiction du modèle s'est bien avérée exacte.

6.3.4 Interprétabilité locale du modèle StackingClassifier

Interprétation locale du modèle stackingClassifier selon l index : 2050

Libellé réel : 1
Prédit : 1



Tout comme le RandomForestClassifier, le StackingClassifier n'est lui non plus pas très éloigné des résultats d'une prédiction aléatoire puisqu'il évalue la prédiction du client pour la classe positive à seulement 56%.

Ainsi, on constate que les modèles présentent souvent des prédictions très différentes bien qu'ils considèrent toujours les mêmes variables comme étant significatives. Effectivement, on s'aperçoit que les variables identifiées comme étant importantes dans nos 4 modèles d'interprétabilité globale (« previous », « housing », « campaign », « contact_cellular », « age », « month », « balance », « marital_married », « marital_single », « poutcome_success », « pdays », « loan », « day », « job_blue-collar » et « education_tertiary ») sont les mêmes que celles qui reviennent dans nos 4 modèles d'interprétation locale (en sachant que l'on a limité le nombre de variables à 10 pour ces derniers). Dans la mesure où il s'agit du même client, le taux de prédiction local de chaque modèle devrait être assez proche de celui des autres, et pourtant il ne l'est pas. La raison en est toute simple : chaque variable présente un taux de significativité différent en fonction du modèle dans lequel elle est appliquée. Ainsi, les différents écarts entre les résultats des classifieurs sont liés aux différentes valeurs des paramètres de leurs variables explicatives les plus importantes. D'autre part, après avoir effectué différents tests que nous n'insérerons pas ici, notamment avec l'ID client 3104 (1990 pour X_test), nous avons également constaté que les modèles RandomForestClassifier et StackingClassifier présentaient des résultats toujours très proches l'un de l'autre en termes d'interprétabilité locale, et ce quel que soit l'ID client testé. On a également remarqué que le GradientBoostingClassifier constituait toujours le classifieur qui présentait les résultats les plus exacts, et que le modèle de régression logistique, au contraire, était celui qui présentait les résultats les moins fiables par rapport aux 3 précédents.

7 Conclusion

Pour conclure, bien qu'il ne fût pas aisé de réussir à optimiser les résultats de nos modèles de classification pour ce projet, particulièrement pour la modalité « 1 » de la variable cible, nous sommes finalement parvenus à obtenir des résultats assez convenables aussi bien pour la modalité « 1 » que « 0 », tout en supprimant les variables qui ne pouvaient être expliquées. Nous considérons qu'il n'est pas possible d'améliorer davantage les résultats pour ce projet, sauf en améliorant la qualité du jeu de données, et notamment en apportant davantage d'informations pertinentes sur les clients, sur l'année des campagnes précédentes et en précisant les modalités non expliquées pour certaines variables (« other » pour « poutcome » et « unknown » pour « job », « education », « contact » et « poutcome »). Ainsi, nous estimons que ce projet peut être pris en exemple pour justifier le fait qu'il ne soit pas possible, dans certains cas, d'obtenir de meilleurs résultats sauf en améliorant la qualité des informations fournies sur lesquelles reposent les modèles de classification. C'est la réponse principale que nous avons décidé d'apporter à la question de départ qui est de savoir comment améliorer l'efficacité des prochaines campagnes marketing. En l'état actuel de nos données, et d'après notre analyse précédente, il semblerait qu'il faille augmenter le nombre d'appels émis par téléphone mobile (« contact_cellular », « campaign » et « previous ») vers des clients présentant les caractéristiques considérées comme essentielles dans l'interprétation globale de nos 4 modèles de classification précédents, notamment les clients célibataires, ayant effectué des études supérieures, avec un solde de compte positif, ne présentant pas de crédit immobilier ni de consommation et ayant déjà été contactés lors des campagnes précédentes. De plus, parmi ces derniers (ceux ayant déjà été contactés lors des campagnes précédentes), il faut davantage multiplier le nombre d'appels vers ceux ayant déjà souscrit au contrat de dépôt à terme car les chances qu'ils y souscrivent à nouveau sont assez élevées. Enfin, il est également nécessaire de multiplier les appels vers des clients compris dans la tranche d'âge suivante : <30 ans et >60 ans.

8 Annexes

Il convient de préciser que les images ont été correctement enregistrées (par le biais de la fonction « savefig ») bien qu'elles ne soient pas de bonne qualité dans certains cas. Pour de plus amples détails, voici ci-dessous le code du projet correctement achevé.

ANALYSE_PREPROCESSING_MACHINE_LEARNING_bank

March 24, 2022

1 INTRODUCTION

Ce fichier est une association du travail conjoint réalisé au préalable sur Google Colab par Nesrine, David et Maxime. Il a pour objectif de réaliser, dans un premier temps, l'analyse des variables catégorielles et quantitatives du dataset ‘bank.csv’, puis ensuite d'effectuer un preprocessing des données en vue de sélectionner les variables nous permettant d’élaborer notre modèle de prédiction. Il s’agit d’une problématique de classification.

2 IMPORT DES PRINCIPALES FONCTIONS

```
[ ]: import pandas as pd
      import numpy as np
      from scipy.stats import pearsonr
      from scipy.stats import chi2_contingency
      import seaborn as sns
      import matplotlib.pyplot as plt
```

3 CHARGEMENT DU DATASET ET VISION GENERALE DES DONNEES

```
[ ]: df = pd.read_csv('bank.csv')

# Visualisation des premières valeurs
display(df.head())

# info sur toutes les variables
print(df.info())
```

4 INVENTAIRE DES DONNEES

Le dataset est constitué de 11162 observations contenant 17 variables :

- 7 variables quantitatives
- 10 variables catégorielles (dont deposit pour l'évaluation) mais certaines devront être transformées en variables quantitatives ou en variables binaires.

Il n'y a pas de valeurs manquantes.

Ci-dessous une liste exhaustive des différentes variables :

- age (variable quantitative) : Age du client
- job (variable categorielle) : Type d'emploi du client
- marital (variable categorielle) : Status marital
- education (variable categorielle) : Niveau d'étude
- balance (variable quantitative) : Solde du compte
- default (variable categorielle) : Le client a-t-il fait defaut sur un emprunt?
- housing (variable categorielle) : Emprunt Immobilier
- loan (variable categorielle) : Emprunt à la consommation
- contact (variable categorielle) : Moyen de communication utilisé
- day (variable quantitative): Jour du dernier contact
- month (variable categorielle) : Mois du dernier contact(au format texte)
- duration (variable quantitative): Duree du dernier contact en secondes
- campaign (variable quantitative): Nombre d'appels pendant ma campagne
- pdays(variable quantitative): Nombre de jour depuis la campagne precedente
- previous(variable quantitative): Nombre d'appels pendant les campagnes precedentes
- poutcome (variable categorielle) : Résultat de la campagne precedente
- Deposit(variable categorielle): le client a-t-il souscrit? il s'agit de la variable cible

5 ANALYSE DES VARIABLES CATEGORIELLES

La fonction cat-scan ci-dessous permet d'analyser intégralement chaque variable catégorielle du dataset (df) en spécifiant pour chacune d'elles :

- Sa modalité la plus fréquente ;
- Un tableau constitué de 3 colonnes dans lesquelles chaque modalité de cette variable est analysée selon son nombre et la part qu'elle y représente ;
- Une analyse statistique entre elle et la variable cible deposit dans laquelle on présente non seulement la p-value mais également le V de Cramer afin de correctement mesurer leur niveau de corrélation. Une interprétation des résultats est également apportée ;
- Et enfin un graphique dans lequel la distribution des modalités de la variable est représentée en fonction de la variable cible.

```
[ ]: def cat_scan(var):  
    print('\nLa modalité la plus présente pour la variable', var, 'est : ',  
        df[var].mode()[0])
```

```

scan = pd.DataFrame(columns=['variable', 'count', 'pourcent'])
scan['variable'] = df[var].value_counts().index.tolist()
scan['count'] = df[var].value_counts().values.tolist()
scan['pourcent'] = df[var].value_counts(normalize = True).round(2).values.
˓→tolist()
display(scan)

table = pd.crosstab(df[var], df.deposit)
resultats_test = chi2_contingency(table)
# la statistique du test du khi2 avec la table de contingence est le
˓→premier indice
statistique = resultats_test[0]
# la p-value du test du khi2 avec la table de contingence est le deuxième
˓→indice
p_valeur = resultats_test[1]
# les degrés de liberté du test du khi2 avec la table de contingence est le
˓→troisième indice
degre_liberte = resultats_test[2]
# print(resultats_test)
print('\nCorrélation variables deposit -', var, ': \n\nStatistique :
˓→',statistique,',', 'p-value :', p_valeur,',', 'degré_de_liberté :',_
˓→degre_liberte,)

# Recherche du V de Cramer :
stat_chi2 = chi2_contingency(table)[0]
# k = nombre de lignes
k = table.shape[0]
# r = nombre de colonnes
r = table.shape[1]
# N = nombre d'observations du jeu de données (nombre de lignes de df)
N = df.shape[0]
# On applique ensuite la formule de phi pour trouver le V de cramer
phi = max(0,(stat_chi2/N)-((k-1)*(r-1)/(N-1)))
k_corr = k - (np.square(k-1)/(N-1))
r_corr = r - (np.square(r-1)/(N-1))
v_cramer = np.sqrt(phi/min(k_corr - 1,r_corr - 1))
# On a tous les éléments pour calculer le v de cramer
print('V de Cramer :',v_cramer,)
if resultats_test[1] < 0.5 and v_cramer < 0.5 :
    print('On rejette H0 car les deux variables ne sont pas indépendantes.
˓→\nCependant, leur corrélation est assez faible selon le V de Cramer\n')
elif resultats_test[1] < 0.5 and v_cramer > 0.5 :
    print('On rejette H0 car les deux variables ne sont pas indépendantes.
˓→\nDe plus, leur corrélation est assez faible selon le V de Cramer\n')
elif resultats_test[1] > 0.5 and v_cramer < 0.5 :

```

```

        print('On accepte H0 et les deux variables ne sont pas\u
↳ significativement dépendantes.\nDe plus, leur corrélation est assez faible\u
↳ selon le V de Cramer\n')
    else :
        print('On accepte H0 et les deux variables ne sont pas\u
↳ significativement dépendantes.\nMalgré tout, leur corrélation est assez\u
↳ forte selon le V de Cramer\n')

# Code pour que les graphiques aient la même hauteur
plt.figure(figsize = (14, df[var].unique().shape[0]/2))

plt.title('Graphique représentant la distribution de la variable ' + var +\u
↳ ' en fonction de la variable cible deposit\n')
sns.countplot(y = var, hue = 'deposit', data = df);
plt.xlabel('occurrence')
plt.legend()
# Enregistrement des images dans le répertoire courant
filename = 'Histogramme_'+var+'.png'
plt.savefig(filename, dpi = 600, bbox_inches = "tight")
plt.show()

```

Application de la fonction cat-scan à chacune des variables catégorielles du dataset df

[]: cat_scan('job')

[]: cat_scan('education')

[]: cat_scan('contact')

[]: cat_scan('marital')

[]: cat_scan('housing')

[]: cat_scan('loan')

[]: cat_scan('month')

[]: cat_scan('poutcome')

[]: cat_scan('default')

6 ANALYSE DES VARIABLES QUANTITATIVES

6.1 Présence de valeurs aberrantes

Nous allons présenter, dans le tableau ci-dessous, une analyse globale de la présence de valeurs aberrantes au sein des variables quantitatives par le biais :

- de l'écart entre la moyenne et la médiane
- l'écart entre les quantiles
- l'écart entre le min et le max

```
[ ]: # Pour commencer, on transforme la variable cible en variable indicatrice.
from sklearn.preprocessing import LabelEncoder
L_E = LabelEncoder()
df.deposit = L_E.fit_transform(df.deposit)

# On regroupe les variables quantitatives dans un dataframe pour une analyse
# plus simple
data_quant = df.select_dtypes(include=['int64','float64'])

# On analyse la présence de valeurs aberrantes par l'écart entre la moyenne et
# la médiane
stats = pd.DataFrame(data_quant.mean().round(2), columns = ['moyenne'])
stats['médiane'] = data_quant.median().round(2)
stats['diff_médiane_moyenne'] = abs((data_quant.mean() - data_quant.median())).round(2)
ecart_type = data_quant.std()
stats['écart-type'] = ecart_type

# On analyse la présence de valeurs aberrantes par l'écart entre les quantiles
stats[['quantile_1', 'quantile_2', 'quantile_3']] = data_quant.quantile(q=[0.
    ↪25, 0.5, 0.75]).transpose()

# Enfin, on analyse la présence de valeurs aberrantes avec l'écart entre le min
# et le max
stats['min'] = data_quant.min()
stats['max'] = data_quant.max()
stats['Ecart_max-min'] = data_quant.max() - data_quant.min()

# On affiche le tableau
stats
```

On constate la présence de valeurs aberrantes pour 3 variables quantitatives, à savoir ‘balance’, ‘duration’, ‘pdays’.

Comme nous l'avons effectué avec les variables qualitatives, nous avons également décidé d'élaborer une fonction afin d'analyser automatiquement la distribution ainsi que la présence de valeurs aberrantes dans toutes les variables quantitatives de notre jeu de données : il s'agit de la fonction num_scan. A la différence de la précédente,

elle a pour objectif de représenter graphiquement les valeurs aberrantes de chacune des variables quantitatives et également de permettre l'analyse de leur distribution. Elle se constitue :

- d'un tableau présentant les principales statistiques descriptives de chaque variable ;
- d'un boxplot représentant la variable concernée en fonction de la variable cible deposit ;
- d'un boxenplot de ladite variable, encore en fonction de notre variable cible ;
- Et enfin, d'un histogramme permettant de représenter la distribution de la variable en question, toujours en fonction de notre variable cible deposit.

```
[ ]: def num_scan(var):  
    display(pd.DataFrame(df[var].describe()))  
  
    fig = plt.figure(figsize = (10, 8))  
    ax = fig.add_subplot(111)  
    plt.title('BoxenPlot de la variable ' + var + ' par rapport à la variable  
    ↴deposit\n')  
    sns.boxenplot(y = var, x = 'deposit', data = df, ax = ax)  
    # Enregistrement du boxenplot dans le répertoire courant  
    filename = 'BoxenPlot_' + var + '.png'  
    plt.savefig(filename, dpi = 1000, bbox_inches = "tight")  
  
    fig = plt.figure(figsize = (15, 4))  
    ax = fig.add_subplot(111)  
    plt.title('Diagramme en boîte de la variable ' + var + ' par rapport à la  
    ↴variable deposit\n')  
    sns.boxplot(x = var, hue = "deposit", data = df, ax = ax)  
    # Enregistrement du boxplot dans le répertoire courant  
    filename = 'Boxplot_' + var + '.png'  
    plt.savefig(filename, dpi = 1000, bbox_inches = 'tight')  
  
    plt.figure(figsize = (15, 4))  
    plt.title('Histogramme représentant la distribution de la variable ' + var  
    ↴+ ' en fonction de la variable cible deposit\n')  
    ax = sns.histplot(x = var, bins = 30, data = df, hue = 'deposit', kde =  
    ↴True, multiple = 'dodge')  
    plt.setp(ax.get_xticklabels(), rotation = 30)  
    plt.xlabel(var)  
    plt.ylabel('occurrence')  
    plt.legend()  
    # Enregistrement de l'histogramme dans le répertoire courant  
    filename = 'Histogramme_' + var + '.png'  
    plt.savefig(filename, dpi = 600, bbox_inches = "tight")
```

Application de la fonction num-scan à chacune des variables quantitatives du dataset df

```
[ ]: num_scan('age')
[ ]: num_scan('duration')
[ ]: num_scan('pdays')
[ ]: num_scan('balance')
[ ]: num_scan('day')
[ ]: # On transforme manuellement la variable month en variable indicatrice avant d'appliquer la fonction
      ↪d'appliquer la fonction
      df.month = df.month.replace({'may' : 5, 'jun' : 6, 'jul' : 7, 'aug' : 8,
                                    'oct' : 10, 'nov' : 11, 'dec' : 12, 'jan' : 1,
                                    'feb' : 2, 'mar' : 3, 'apr' : 4, 'sep' : 9})
      num_scan('month')
[ ]: num_scan('campaign')
[ ]: num_scan('previous')
```

De prime abord, On constate la présence de valeurs aberrantes particulièrement pour 3 variables quantitatives, à savoir « balance », « duration », et « pdays ». On peut observer cela dans les colonnes « diff_médiane_moyenne » et « Ecart max-min » notamment, ainsi que dans l'écart entre les différents quantiles pour ces mêmes variables. Cependant, cela reste à nuancer : effectivement, il y a peu de points isolés dans les différents graphiques de visualisation. Les diagrammes en boîte ainsi que les histogrammes présentent souvent des alignements de points qui ressemblent plus à des distributions exponentielles qu'à de vrais outliers.

6.2 Corrélation entre les variables quantitatives

Tout d'abord, l'étude de la corrélation entre les différentes variables quantitatives s'effectuera graphiquement, par le biais d'un pairplot et d'une matrice de corrélation. Par la suite, nous utiliserons le test statistique de Pearson pour confirmer nos résultats.

```
[ ]: sns.pairplot(df[['balance', 'day', 'duration', 'age', 'campaign', 'previous',
      ↪'pdays']], diag_kind = 'kde', data = df);
      # Enregistrement du pairplot dans le répertoire courant
      filename = 'Pairplot.png'
      plt.savefig(filename, dpi = 600, bbox_inches = "tight")
```

Comme on peut l'observer, les variables sont très peu corrélées entre elles, mais on constate la présence de valeurs extrêmes.

```
[ ]: # Matrice de corrélation

cor = df.corr().round(2)
display(cor)
plt.figure(figsize=(12,10))
ax = sns.heatmap(cor, annot = True, cmap = "RdYlGn", vmin = -0.7, vmax = 0.7)
ax.set_title('MATRICE DE CORRELATION\n');
# Enregistrement de la matrice dans le répertoire courant
filename = 'Matrice_corr.png'
plt.savefig(filename, dpi = 600, bbox_inches = "tight")
```

D'après la matrice de confusion, il y a très peu de corrélation entre les variables quantitatives.

Cependant, on peut constater que la corrélation est plus importante entre la variable ‘duration’ et la variable cible ‘deposit’.

Par ailleurs, on constate également une corrélation assez forte entre la variable ‘pdays’ et la variable ‘previous’ : ceci est cohérent dans la mesure où pdays correspond au nombre de jours qui se sont écoulés depuis que le client a été contacté et ‘previous’ correspond au nombre de contacts téléphoniques lors des campagnes précédentes.

Sur la base de ce constat, on va donc effectuer un test de Pearson entre la variable cible et ‘duration’ :

```
[ ]: # from scipy.stats import pearsonr

print('Test de corrélation de Pearson variables deposit-duration\n\n',pd.
    DataFrame(pearsonr(df['deposit'], df['duration']),

    index=['pearson_coeff','p-value'],

    columns=['resultat_test']).round(2),

    '\n\n Le coefficient de Pearson n est pas négligeable, comme on a déjà pu
    observer dans la matrice de corrélation.\n',
    'La p-value étant strictement inférieure à 0.5, on rejette H0 : les deux
    variables testées ne sont pas indépendantes.\n',
    'De plus, la corrélation est assez significative selon le coefficient de
    Pearson.')
# On peut également effectuer le test avec une autre variable quantitative au
# hasard, la variable balance par exemple :

print('\n\nTest de corrélation de Pearson variables deposit-balance\n\n',pd.
    DataFrame(pearsonr(df.deposit, df.balance),
```

```

↳     index = ['pearson_coeff', 'p-value'],
↳
↳     columns = ['resultat_test']).round(2),
↳
↳     '\n\nLa p-value étant strictement inférieure à 0.5, on rejette H0 : les
↳deux variables testées ne sont pas indépendantes.\n',
↳
↳     'Cependant, la corrélation est très peu significative selon le
↳coefficients de Pearson.')

```

Notre constat concernant la corrélation entre la variable ‘duration’ et deposit est vérifié

7 PREPROCESSING 1 (pour modélisation sans sélection de variables)

7.1 TRANSFORMATION DES VARIABLES CATEGORIELLES EN VARIABLES INDICATRICES

```
[ ]: # On encode les variables default, loan et housing ('deposit' a déjà été
↳encodée)
```

```

from sklearn.preprocessing import LabelEncoder
L_E = LabelEncoder()
df['default'] = L_E.fit_transform(df['default'])
df.deposit = L_E.fit_transform(df.deposit)
df.loan = L_E.fit_transform(df.loan)
df.housing = L_E.fit_transform(df.housing)

df.head(2)

```

```
[ ]: # On effectue une dichotomisation des variables job, marital, education,
↳contact et poutcome
# On est ensuite obligé de mettre le code en commentaire car code erreur
↳puisque les
# variables ont été supprimées.
```

```

for i, j in zip((df.job, df.marital, df.education, df.contact, df.poutcome),
                 ('job', 'marital', 'education', 'contact', 'poutcome')):
    dicho_ = pd.get_dummies(i, prefix = j)
    df = df.join(dicho_)
    df = df.drop(j, axis = 1)

# Il ne reste que la variable month à convertir en variable indicatrice :

```

```

df.month.unique()
df.month = df.month.replace({'may' : 5, 'jun' : 6, 'jul' : 7, 'aug' : 8,
                            'oct' : 10, 'nov' : 11, 'dec' : 12, 'jan' : 1,
                            'feb' : 2, 'mar' : 3, 'apr' : 4, 'sep' : 9})

# On vérifie à présent à nouveau qu'aucun Nan n'est présent dans le dataframe :
df.isna().sum()

```

[]: # On affiche le dataset pour vérifier les résultats :

```
df.head(3)
```

7.2 SUPPRESSION DES VALEURS ABERRANTES

[]: # On élimine ensuite les valeurs aberrantes :
Même si quelque peu inutile

```

df = df.drop(df.loc[df.pdays > 750].index)
df = df.drop(df.loc[df.balance > 60000].index)
df = df.drop(df.loc[df.campaign > 35].index)
df = df.drop(df.loc[df.previous > 35].index)

```

7.3 NORMALISATION DES DONNEES

[]: # On normalise les données
from sklearn import preprocessing

```

# On ne normalise pas la variable cible, donc on la supprime
# On la conserve d'abord dans une variable appelée deposit
deposit = df.deposit
df = df.drop('deposit', axis = 1)

scaler = preprocessing.StandardScaler().fit(df)
df = scaler.transform(df)

df = pd.DataFrame(df, columns = ['age', 'default', 'balance', 'housing',
                                  'loan', 'day', 'month',
                                  'duration', 'campaign', 'pdays',
                                  'previous', 'job_admin.', 'job_blue-collar',
                                  'job_entrepreneur', 'job_housemaid',
                                  'job_management', 'job_retired',
                                  'job_self-employed', 'job_services',
                                  'job_student', 'job_technician',
                                  'job_unemployed', 'job_unknown'],
                           )

```

```

        'marital_divorced', 'marital_married', □
↳ 'marital_single',
                'education_primary', □
↳ 'education_secondary', 'education_tertiary',
                'education_unknown', 'contact_cellular', □
↳ 'contact_telephone',
                'contact_unknown', 'poutcome_failure', □
↳ 'poutcome_other',
                'poutcome_success', 'poutcome_unknown'])
)

df.head()

```

7.4 SEPARATION DES DONNEES EN JEU D'ENTRAINEMENT ET DE TEST

```

[ ]: # On commence par éliminer la variable duration qui est très corrélée avec la
      ↳variable cible :
df = df.drop('duration', axis = 1)

# On sépare les données en data et en target
# Ici, on ne fait que de les renommer
target = deposit
# On a déjà supprimé deposit, on change juste de nom
data = df

# On sépare les données en jeu d'entraînement et de test
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(data, target, test_size = 0.
      ↳2)

X_train

```

7.5 VERIFICATION DE LA DISTRIBUTION DE LA VARIABLE CIBLE DEPOSIT (OVERFITTING)

```

[ ]: # On vérifie que la variable deposit est bien équilibrée
# afin d'éviter le sur-entraînement des modèles de machine-learning
print('distribution de la variable cible target\n', target.
      ↳value_counts(normalize=True))

# D'après les résultats, la variable deposit est à peu
# près équilibrée, on applique tout de même un over-sampling
# avec RandomOverSampler et SMOTE
from imblearn.over_sampling import RandomOverSampler, SMOTE

```

```

# RANDOM OVER SAMPLER
ROS = RandomOverSampler()
# Tuple assignment
X_train, Y_train = ROS.fit_resample(X_train, Y_train)
print('\nClasses échantillon oversampled :\n', dict(pd.Series(Y_train).
    value_counts(normalize = True)))

# SMOTE
SMO = SMOTE()
# Tuple assignment
X_train, Y_train = SMO.fit_resample(X_train, Y_train)
print('\nClasses échantillon SMOTE :\n', dict(pd.Series(Y_train).
    value_counts(normalize = True)))

# Les échantillons sont maintenant totalement équilibrés.

```

8 MODELES DE CLASSIFICATION

8.1 MODELISATION STATISTIQUE (SANS AVOIR SELECTIONNE LES VARIABLES)

8.1.1 MODELES INDIVIDUELS selon RandomizedSearchCV (sans sélection de variables)

```

[ ]: # AVEC RandomizedSearchCV

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import classification_report

clf_logistic = LogisticRegression()
clf_rdf = RandomForestClassifier()
clf_grad = GradientBoostingClassifier()

params_logistic = {'C' : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]}

params_rdf = {'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000],
              'max_features': [.5, .7],
              'bootstrap' : [False, True],
              'max_depth' : [3, 6]}

params_grad = {'n_estimators' : [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000],
               'max_depth' : [2,3,5,6],
               'learning_rate' : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]}

```

```

clf = [clf_logistic, clf_rdf, clf_grad]
params = [params_logistic, params_rdf, params_grad]

for i, j in zip(clf, params):
    from sklearn.model_selection import RandomizedSearchCV
    grid = RandomizedSearchCV(estimator = i, param_distributions = j, cv = 10)
    grid = grid.fit(X_train, Y_train)
    y_pred = grid.predict(X_test)
    score = grid.score(X_test, Y_test)
    print('\n\nRESULTATS POUR LE MODELE', i, 'sans sélection de variables :'
        '\n\n', classification_report(Y_test, y_pred),
        '\nMeilleurs paramètres retenus :', grid.best_params_)

```

8.1.2 MODELES GROUPES (sans sélection de variables)

StackingClassifier avec et sans RandomizedSearchCV

```

[ ]: ##### SANS VALIDATION CROISEE NI
      ↵RandomizedSearchCV #####
      ↵

# Import des librairies
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, ↵
    ↵GradientBoostingClassifier, StackingClassifier
from sklearn.metrics import classification_report

# Instanciation de tous les classifieurs
# On met les meilleurs paramètres retenus dans les modèles précédents
clf_log = LogisticRegression(C = 0.3)
clf_random_f = RandomForestClassifier(n_estimators = 100, max_features = 0.5, ↵
    ↵max_depth = 6, bootstrap = True)
clf_grad_boos = GradientBoostingClassifier(n_estimators = 200, max_depth = 6, ↵
    ↵learning_rate = 0.1)
sclf = StackingClassifier(estimators = [('LOGISTIC', clf_log), ↵
    ↵('RANDOM_FOREST', clf_random_f), ↵
    ↵('GRADIENT_BOOSTING', clf_grad_boos)], ↵
    ↵final_estimator = clf_grad_boos)

# Entraînement du modèle StackingClassifier
sclf.fit(X_train, Y_train)
y_pred = sclf.predict(X_test)

# Affichage des résultats
print('\n\nRESULTATS POUR LE MODELE StackingClassifier sans RandomizedSearchCV'
    ↵(sans sélection de variables) : \n\n',
    classification_report(Y_test, y_pred), '\n\n')

```

```

#####
##### AVEC RANDOMIZED
#####

# Import des librairies
from sklearn.ensemble import RandomForestClassifier,
    GradientBoostingClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import classification_report

# Instanciation de tous les classifieurs
clf_log = LogisticRegression()
clf_random_f = RandomForestClassifier()
clf_grad_boos = GradientBoostingClassifier()
sclf = StackingClassifier(estimators = [('LOGISTIC', clf_log),
                                         ('RANDOM_FOREST', clf_random_f),
                                         ('GRADIENT_BOOSTING', clf_grad_boos)],
                           final_estimator = clf_grad_boos)

params = {'LOGISTIC__C' : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1],
          'RANDOM_FOREST__n_estimators' : [100, 200, 300, 400, 500, 600, 700,
                                          800, 900, 1000],
          'RANDOM_FOREST__max_features' : [.5, .7],
          'RANDOM_FOREST__bootstrap' : [False, True],
          'RANDOM_FOREST__max_depth' : [3, 6],
          'GRADIENT_BOOSTING__n_estimators' : [100, 200, 300, 400, 500, 600,
                                              700, 800, 900, 1000],
          'GRADIENT_BOOSTING__max_depth' : [2,3,5,6],
          'GRADIENT_BOOSTING__learning_rate' : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,
                                              0.8, 0.9, 1]}

grid = RandomizedSearchCV(estimator = sclf, param_distributions = params, cv = 10)
grid = grid.fit(X_train, Y_train)
y_pred = grid.predict(X_test)
print('RESULTATS POUR LE MODELE StackingClassifier avec RandomizedSearchCV',
      '(sans sélection de variables) : \n\n',
      classification_report(Y_test, y_pred),
      '\nMeilleurs paramètres retenus : \n\n', grid.best_params_, '\n\n')

```

8.2 MODELISATION STATISTIQUE (APRES AVOIR SELECTIONNE LES VARIABLES)

8.2.1 REDUCTION DE DIMENSION par la méthode PCA (NON NECESSAIRE ICI)

```
[ ]: # A présent, on va tenter d'utiliser la cpa afin de sélectionner les variables
# On n'a pas éliminé encore les outliers pour l'instant

from sklearn.decomposition import PCA

pca = PCA(n_components = 0.95)
pca.fit(X_train, X_test)
print("\nNombre de composantes retenues avec PCA en conservant une variance_"
    "expliquée à 95%:", pca.n_components_)

# 25 variables du modèle ont été retenues, ce qui n'est pas étonnant au vu de_
    "leur
# faible corrélation (l'intérêt de la pca est d'éliminer les corrélations
# entre les variables par le biais de la combinaison linéaire).
# Dans le cas présent, il n'y a aucun intérêt à effectuer une pca.

# Les parts de variance expliquée se cumulent et on peut
# clairement voir à quelle variance expliquée on en est
# au fur et à mesure que le nombre de composantes principales
# évolue. Il faut conserver une variance expliquée assez élevée
# pour ne pas perdre l'information en supprimant les variables
pca.explained_variance_ratio_.cumsum()
```

8.2.2 SELECTION DE VARIABLES AVEC LES WRAPPER METHODS

```
[ ]: # J'ai choisi de ne pas faire les embedded methods car elles utilisent le_
    "modèle sum
# dans le cas de la classification et ce n'est pas optimal en terme de temps

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold
from sklearn.feature_selection import RFECV

clf_logistic = LogisticRegression()
clf_arbre = DecisionTreeClassifier()

crossval = KFold(n_splits = 5, random_state = 2, shuffle = True)
rfecv = RFECV(estimator = clf_logistic, cv = crossval)
rfecv.fit(X_train, Y_train)
# Obtenir les variables sélectionnées par ordre (bolléens)
```

```

mask = rfecv.get_support()

# On trace la courbe de scores en fonction du nombre de variables sélectionnées
plt.plot(rfecv.grid_scores_);
plt.title('Performance du modèle en fonction du nombre de variables sélectionnées par validation croisée')
plt.xlabel('Performance du modèle')
plt.ylabel('Nombre de variables sélectionnées')
print("Nombre de features retenus :", rfecv.n_features_)
# Enregistrement de la courbe des scores dans le répertoire courant
filename = 'Courbe_Score_RFE_LOGISTIC.png'
plt.savefig(filename, dpi = 600, bbox_inches = "tight")

liste_var_select = []
for i, j in zip(mask, data) :
    if i == True :
        liste_var_select.append(j)
print('\n\nLes variables qui ont été sélectionnées par wrapper method sont : \n\n', liste_var_select)

Colonnes_data = ['age', 'default', 'balance', 'housing', 'loan', 'day', 'month',
                  'campaign', 'pdays', 'previous', 'job_admin.', ,
                  'job_blue-collar',
                  'job_entrepreneur', 'job_housemaid', 'job_management', ,
                  'job_retired',
                  'job_self-employed', 'job_services', 'job_student', ,
                  'job_technician',
                  'job_unemployed', 'job_unknown', 'marital_divorced', ,
                  'marital_married',
                  'marital_single', 'education_primary', 'education_secondary', ,
                  'education_tertiary',
                  'education_unknown', 'contact_cellular', 'contact_telephone', ,
                  'contact_unknown',
                  'poutcome_failure', 'poutcome_other', 'poutcome_success', ,
                  'poutcome_unknown']

liste_var_del = []
for j in Colonnes_data :
    if j not in liste_var_select :
        liste_var_del.append(j)

i = 0
for x in liste_var_del :
    i = i + 1
print('\n\nLe nombre de variables supprimées par le modèle est de : ', i)

```

```
print('\n\nLes variables qui ont été supprimées par wrapper method sont :\n\n',  
      liste_var_del, '\n')
```

8.2.3 Variables à ajouter à la liste des variables supprimées par wrapper methods

```
[ ]: # Par ailleurs, j'ai remarqué grâce aux graphiques de la fonction cat_scan que  
    ↴certaines variables  
# catégorielles contiennent vraiment beaucoup de no par rapport aux yes (0 par  
    ↴rapport aux 1),  
# et je souhaiterais les supprimer également pour voir si cela change quelque  
    ↴chose aux résultats  
# et de quelle manière ça les impacte.  
# Je vais rajouter ces variables à supprimer quelles que soient les variables  
    ↴supprimées par les wrapper  
# methods.  
  
# Il s'agit des variables :  
# job_admin., job_blue-collar, job_self-employed, education_secondary,  
    ↴education_primary,  
# contact_unknown, marital_married, month (pour may) et poutcome_unknown.  
  
# On crée une liste avec les variables que l'on souhaite supprimer en plus :  
var_a_supp = ['job_admin.', 'job_blue-collar', 'job_self-employed',  
    ↴'job_unknown',  
        'job_unknown', 'education_secondary', 'education_primary',  
        'education_unknown', 'contact_unknown', 'marital_divorced',  
    ↴'month',  
        'poutcome_unknown', 'pdays', 'age']  
  
# On les ajoute à la liste à supprimer :  
for y in var_a_supp :  
    if y not in liste_var_del :  
        liste_var_del.append(y)  
  
# On va tester pour voir si le code marche :  
z = 0  
for x in liste_var_del :  
    z = z + 1  
  
print('\n\nLe nombre total de variables supprimées par le modèle est de : ', z)  
print('\n\nLe nombre de variables catégorielles qui ont été supprimées en plus  
    ↴est de : ', z - i)  
print('\n\nLes variables totales qui ont été supprimées sont :\n\n',  
      liste_var_del)  
print('\n\nLes variables catégorielles qui ont été supprimées en plus sont :',  
      ↴liste_var_del[-z + i:], '\n')
```

8.2.4 SELECTION DE VARIABLES AVEC LES METHODES TRADITIONNELLES

Variables les plus importantes selon LogisticRegression

```
[ ]: # print('\nVariables les plus importantes selon le modèle de régression logistique :\n')
clf_logistic = LogisticRegression(C = 0.3)
clf_logistic.fit (X_train, Y_train)
pd.Series(clf_logistic.coef_[0], pd.DataFrame(X_train, columns = ['age', 'default',
                     'balance', 'housing', 'loan',
                     'day', 'month',
                     'campaign', 'pdays', 'previous', 'job_admin.', 'job_blue-collar', 'job_entrepreneur', 'job_housemaid', 'job_management', 'job_retired', 'job_self-employed', 'job_services', 'job_student', 'job_technician', 'job_unemployed', 'job_unknown', 'marital_divorced', 'marital_married', 'marital_single', 'education_primary', 'education_secondary', 'education_tertiary', 'education_unknown', 'contact_cellular', 'contact_telephone', 'contact_unknown', 'poutcome_failure', 'poutcome_other', 'poutcome_success', 'poutcome_unknown']).columns).sort_values(ascending=True).plot(kind = 'barh', figsize=(4,8));

plt.title('Les variables les plus importantes selon le modèle de régression logistique :\n')

# Enregistrement de la figure dans le répertoire courant
filename = 'VAR_SELECT_LOG_METHODE_CLASSIQUES.png'
plt.savefig(filename, dpi = 600, bbox_inches = "tight")

# On constate que c'est cohérent avec les variables qui ont été supprimées juste au-dessus par les wrapper methods
```

Variables les plus importantes selon RandomForestClassifier

```
[ ]: from sklearn.ensemble import RandomForestClassifier
clf_random_f = RandomForestClassifier(n_estimators = 100, max_features = 0.5,
                                       max_depth = 6, bootstrap = True)

clf_random_f.fit (X_train, Y_train)
plt.figure(figsize = (10,10))
plt.title('Les variables les plus importantes selon RandomForestClassifier :\n')
feat_importances = pd.Series(clf_random_f.feature_importances_,
                               index = X_train.columns).nlargest(36).
                               sort_values(ascending = False).plot(kind='barh');
```

```
# Enregistrement de la figure dans le répertoire courant
filename = 'VAR_SELECT_RDF METHODE_CLASSIQUES.png'
plt.savefig(filename, dpi = 600, bbox_inches = "tight")
```

Variables les plus importantes selon GradientBoostingClassifier

```
[ ]: from sklearn.ensemble import GradientBoostingClassifier
clf_grad_boos = GradientBoostingClassifier(n_estimators = 200, max_depth = 6,
                                            learning_rate = 0.1)

clf_grad_boos.fit (X_train, Y_train)
plt.figure(figsize = (10,10))
plt.title('Les variables les plus importantes selon GradientBoostingClassifier :
           \n')
feat_importances = pd.Series(clf_grad_boos.feature_importances_,
                               index = X_train.columns).nlargest(36).
                               sort_values(ascending = False).plot(kind='barh');

# Enregistrement de la figure dans le répertoire courant
filename = 'VAR_SELECT_GRADBOOST METHODE_CLASSIQUES.png'
plt.savefig(filename, dpi = 600, bbox_inches = "tight")
```

8.2.5 SELECTION DE VARIABLES - DECISION ARBITRAIRE

```
[ ]: liste_var_supp = ['contact_unknown', 'poutcome_unknown', 'job_unknown',
                      'education_unknown', 'poutcome_other']
```

8.3 PREPROCESSING 2 (pour modélisation après suppression des variables)

```
[ ]: # Pas besoin de normaliser les données, elles l'ont déjà été
      # duration et deposit ont déjà été éliminées dans la partie preprocessing

      # On va supprimer les colonnes présentes dans liste_var_supp
      # Je mets en commentaire pour éviter les codes erreur car une fois supprimées
      # les colonnes n'existent plus
      df = df.drop(liste_var_supp, axis = 1)

      # On vérifie que df contient le bon nombre de colonnes après suppression des
      # variables ci-dessus
      print('\n\nNombre de colonnes de df : ', df.columns.shape)

      # On sépare les données en data et en target
      data = df
      target = deposit
      print('\n\nNombre de colonnes de data : ', data.columns.shape)
```

```

# On sépare les données en jeu d'entraînement et de test
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(data, target, test_size = 0.
    ↵2)

# On vérifie que la variable deposit est bien équilibrée
# afin d'éviter le sur-entraînement des modèles de machine-learning
print('\n\nDistribution de la variable cible target\n', target.
    ↵value_counts(normalize=True))

# D'après les résultats, la variable deposit est à peu
# près équilibrée, on applique tout de même un over-sampling
# avec RandomOverSampler et SMOTE
from imblearn.over_sampling import RandomOverSampler, SMOTE
# RANDOM OVER SAMPLER
ROS = RandomOverSampler()
# Tuple assignment
X_train, Y_train = ROS.fit_resample(X_train, Y_train)
print('\nClasses échantillon oversampled :\n', dict(pd.Series(Y_train).
    ↵value_counts(normalize = True)), '\n\n')
# Les échantillons sont maintenant totalement équilibrés.

```

8.3.1 MODELES INDIVIDUELS selon RandomizedSearchCV (avec sélection de variables)

```

[ ]: # AVEC RandomizedSearchCV

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import classification_report

clf_logistic = LogisticRegression()
clf_rdf = RandomForestClassifier()
clf_grad = GradientBoostingClassifier()

params_logistic = {'C' : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]}

params_rdf = {'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, ↵
    ↵1000],
    'max_features': [.5, .7],
    'bootstrap' : [False, True],
    'max_depth' : [3, 6]}

params_grad = {'n_estimators' : [100, 200, 300, 400, 500, 600, 700, 800, 900, ↵
    ↵1000],
    'max_depth' : [2,3,5,6],
    'max_leaf_nodes' : [5, 10, 20, 50, 100]}

```

```

        'learning_rate' : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, u
 ↪1]}

clf = [clf_logistic, clf_rdf, clf_grad]
params = [params_logistic, params_rdf, params_grad]

for i, j in zip(clf, params):
    from sklearn.model_selection import RandomizedSearchCV
    grid = RandomizedSearchCV(estimator = i, param_distributions = j, cv = 10)
    grid = grid.fit(X_train, Y_train)
    y_pred = grid.predict(X_test)
    score = grid.score(X_test, Y_test)
    print('RESULTATS POUR LE MODELE', i, 'avec sélection de variables :\n\n', u
 ↪classification_report(Y_test, y_pred),
          '\nMeilleurs paramètres retenus :', grid.best_params_, '\n\n')

```

8.3.2 MODELES GROUPES (avec sélection de variables)

StackingClassifier avec et sans RandomizedSearchCV

```

[ ]: # On vérifie d'abord que df contient le nombre de colonnes qui a été u
 ↪sélectionné par les

# wrapper methods plus haut (le preprocessing pour la sélection de variables u
 ↪est effectué

# dans la partie juste au-dessus 'MODELES BASIQUES selon RandomizedSearchCV' u
 ↪(avec sélection de variables)'
print('\n\nNombre de colonnes de df : ', df.columns.shape, '\n\n')

##### SANS VALIDATION CROISEE NI u
↪RandomizedSearchCV #####

```

```

# Import des librairies
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, u
↪GradientBoostingClassifier, StackingClassifier
from sklearn.metrics import classification_report

# Instanciation de tous les classifieurs
# On met les meilleurs paramètres retenus dans les modèles précédents
clf_log = LogisticRegression(C = 0.1)
clf_random_f = RandomForestClassifier(n_estimators = 800, max_features = 0.5, u
↪max_depth = 6, bootstrap = True)
clf_grad_boos = GradientBoostingClassifier(n_estimators = 900, max_depth = 6, u
↪learning_rate = 0.1)
scf = StackingClassifier(estimators = [('LOGISTIC', clf_log), u
↪('RANDOM_FOREST', clf_random_f),

```

```

('GRADIENT_BOOSTING', clf_grad_boos)],

final_estimator = clf_grad_boos)

# Entraînement du modèle StackingClassifier
sclf.fit(X_train, Y_train)
y_pred = sclf.predict(X_test)

# Affichage des résultats
print('RESULTATS POUR LE MODELE StackingClassifier sans RandomizedSearchCV')
    ↪(avec sélection de variables) : \n\n',
classification_report(Y_test, y_pred), '\n\n')

#####
##### AVEC RANDOMIZED
#####

# Import des librairies
from sklearn.ensemble import RandomForestClassifier,
    ↪GradientBoostingClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import classification_report

# Instanciation de tous les classifieurs
clf_log = LogisticRegression()
clf_random_f = RandomForestClassifier()
clf_grad_boos = GradientBoostingClassifier()
sclf = StackingClassifier(estimators = [ ('LOGISTIC', clf_log),
                                         ('RANDOM_FOREST', clf_random_f),
                                         ('GRADIENT_BOOSTING', clf_grad_boos)],
                           final_estimator = clf_grad_boos)

params = {'LOGISTIC__C' : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1],
          'RANDOM_FOREST__n_estimators' : [100, 200, 300, 400, 500, 600, 700,
                                          ↪800, 900, 1000],
          'RANDOM_FOREST__max_features' : [.5, .7],
          'RANDOM_FOREST__bootstrap' : [False, True],
          'RANDOM_FOREST__max_depth' : [3, 6],
          'GRADIENT_BOOSTING__n_estimators' : [100, 200, 300, 400, 500, 600,
                                              ↪700, 800, 900, 1000],
          'GRADIENT_BOOSTING__max_depth' : [2,3,5,6],
          'GRADIENT_BOOSTING__learning_rate' : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.
                                              ↪7, 0.8, 0.9, 1]}

grid = RandomizedSearchCV(estimator = sclf, param_distributions = params, cv =
    ↪10)
grid = grid.fit(X_train, Y_train)

```

```

y_pred = grid.predict(X_test)
print('RESULTATS POUR LE MODELE StackingClassifier avec RandomizedSearchCV')
    ↵(avec sélection de variables) : \n\n',
classification_report(Y_test, y_pred),
'\nMeilleurs paramètres retenus : \n\n', grid.best_params_, '\n\n')

```

9 INTERPRETABILITE

Installation de skater

[]: pip install skater

9.1 PREPROCESSING 3 (pour interprétabilité : enlever la normalisation des données)

```

[ ]: # On reprend le fichier d'origine
df2 = pd.read_csv('bank.csv')

# On encode les variables default, loan, deposit et housing
from sklearn.preprocessing import LabelEncoder
L_E = LabelEncoder()
df2['default'] = L_E.fit_transform(df2['default'])
df2.deposit = L_E.fit_transform(df2.deposit)
df2.loan = L_E.fit_transform(df2.loan)
df2.housing = L_E.fit_transform(df2.housing)

# On effectue une dichotomisation des variables job, marital, education,
# contact et poutcome
# On est ensuite obligé de mettre le code en commentaire car code erreur
# puisque les
# variables ont été supprimées.
for i, j in zip((df2.job, df2.marital, df2.education, df2.contact, df2.
    ↵poutcome),
                 ('job', 'marital', 'education', 'contact', 'poutcome')):
    dicho_ = pd.get_dummies(i, prefix = j)
    df2 = df2.join(dicho_)
    df2 = df2.drop(j, axis = 1)

# Il ne reste que la variable month à convertir en variable indicatrice :
df2.month.unique()
df2.month = df2.month.replace({'may' : 5, 'jun' : 6, 'jul' : 7, 'aug' : 8,
                               'oct' : 10, 'nov' : 11, 'dec' : 12, 'jan' : 1,
                               'feb' : 2, 'mar' : 3, 'apr' : 4, 'sep' : 9})

# On vérifie à présent qu'aucun Nan n'est présent dans le dataframe :
print(df2.isna().sum())

```

```

# On élimine ensuite les valeurs aberrantes avant de supprimer les colonnes :
# Même si quelque peu inutile
df2 = df2.drop(df2.loc[df2.pdays > 750].index)
df2 = df2.drop(df2.loc[df2.balance > 60000].index)
df2 = df2.drop(df2.loc[df2.campaign > 35].index)
df2 = df2.drop(df2.loc[df2.previous > 35].index)

# On supprime d'abord la variable cible 'deposit'
# On la conserve d'abord dans une variable appelée deposit
deposit = df2.deposit
df2 = df2.drop('deposit', axis = 1)

# Liste des variables à supprimer
# On rajoute duration que l'on n'a pas supprimé dans le Preprocessing 3
liste_var_supp = ['duration', 'contact_unknown', 'poutcome_unknown', ↴
    'job_unknown', 'education_unknown', 'poutcome_other']

# On va supprimer les colonnes présentes dans liste_var_supp
# Je mets en commentaire pour éviter les codes erreur car une fois supprimées
# les colonnes n'existent plus
df2 = df2.drop(liste_var_supp, axis = 1)

# On vérifie que df2 contient le bon nombre de colonnes après suppression des ↴
# variables ci-dessus
print('\n\nNombre de colonnes de df : ', df2.columns.shape)

# On sépare les données en data et en target
data = df2
target = deposit
print('\n\nNombre de colonnes de data : ', data.columns.shape)

# On sépare les données en jeu d'entraînement et de test
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(data, target, test_size = 0. ↴
    2)

# On vérifie que la variable deposit est bien équilibrée
# afin d'éviter le sur-entraînement des modèles de machine-learning
print('\n\nDistribution de la variable cible target\n', target.
    value_counts(normalize=True))

# D'après les résultats, la variable deposit est à peu
# près équilibrée, on applique tout de même un over-sampling
# avec RandomOverSampler et SMOTE
from imblearn.over_sampling import RandomOverSampler, SMOTE
# RANDOM OVER SAMPLER
ROS = RandomOverSampler()

```

```

# Tuple assignment
X_train, Y_train = ROS.fit_resample(X_train, Y_train)
print('\nClasses échantillon oversampled :\n', dict(pd.Series(Y_train).
    ↪value_counts(normalize = True)), '\n\n')
# Les échantillons sont maintenant totalement équilibrés.

# On vérifie que X_train et X_test contiennent le bon nombre de colonnes et de
# lignes
print(X_train.columns)
print('\nLes dimensions de X_train sont :', X_train.shape)

```

9.2 INTERPRETABILITE globale du modèle RandomForestClassifier AVEC SKATER

```

[ ]: from skater.model import InMemoryModel
from skater.core.explanations import Interpretation
from sklearn.ensemble import RandomForestClassifier
clf_rdf = RandomForestClassifier(n_estimators = 800, max_features = 0.5,
    ↪max_depth = 6, bootstrap = True)
clf_rdf.fit(X_train, Y_train)

##### Interprétabilité globale avec la sélection de variables
    ↪effectuée au préalable #####
# Explications :

# Maintenant que nous avons mis en place le modèle de la boîte noire, nous
    ↪pouvons essayer plusieurs techniques pour
# comprendre quels sont les principaux moteurs derrière les décisions du modèle.
    ↪Comme ce modèle est accessible
# localement, il suffit de créer un objet InMemoryModel. Le seul argument
    ↪obligatoire pour un InMemoryModel est la
# fonction de génération de prédiction. Dans notre cas, il s'agit de
    ↪predict_proba du Keras ANN.
# Nous créons ensuite un objet Interpretation qui transmet les données à la
    ↪fonction de prédiction. Nous limitons
# notre interprétation à 11200 échantillons pour calculer l'importance, et nous
    ↪demandons également que les
# caractéristiques soient triées par importance dans un ordre croissant.
# La mise en œuvre du calcul de l'importance des attributs est basée sur
    ↪l'analyse d'importance variable (VIA).
# Skater utilise différentes techniques selon le type de modèle (par exemple
    ↪régression, classification multi-classes,
# etc.), mais il s'appuie généralement sur la mesure de l'entropie dans le
    ↪changement des prédictions compte tenu d'une

```

```

# perturbation d'une caractéristique. Voir Wei et al. (2015) pour plus de ↵
# détails.

# Skater est un package prônant l'interprétation globale et locale de toutes ↵
# formes de modèles, afin d'aider à construire
# des systèmes de Machine Learning interprétables pour des cas d'utilisation ↵
# dans le monde réel. Skater utilise uniquement
# un jeu de données, ainsi qu'une fonction de prédition entraînée sur ces ↵
# données pour ces interprétations.

# On utilise X_train.values pour éviter code erreur lié au dataframe pandas
model = InMemoryModel(clf_rdf.predict_proba, examples = X_train.values, ↵
    target_names = ['Non-souscription', 'Souscription'])
interpreter = Interpretation(X_train.values, feature_names = X_train.columns, ↵
    training_labels = Y_train)
plots = interpreter.feature_importance.feature_importance(model, n_jobs = -1, ↵
    ascending = True, n_samples = 11200)
# Affichage des 36 variables les plus importantes du modèle
plots.tail(36).plot.barh(figsize = (10, 12))
plt.title('Interprétabilité globale du modèle RandomForestClassifier avec ↵
    SKATER\n');

# Enregistrement de la figure dans le répertoire courant
filename = 'INTERPRETABILITE_GLOBALE_RDF_SKATER.png'
plt.savefig(filename, dpi = 600, bbox_inches = "tight")

#####
##### Dépendance partielle #####
#####

# Explications :

# La dépendance partielle décrit l'impact marginal d'une caractéristique sur la ↵
# prédition du modèle,
# en maintenant les autres caractéristiques constantes dans le modèle. La ↵
# dérivée de la dépendance
# partielle décrit l'impact d'une caractéristique (analogie à un coefficient de ↵
# caractéristique dans
# un modèle de régression).

# Partial Dependence Plot est une autre méthode visuelle, qui est indépendante ↵
# du modèle et peut être utilisée avec
# succès pour mieux comprendre le fonctionnement interne d'un modèle de boîte ↵
# noire comme un ANN profond. Les PDP
# sont un outil efficace pour évaluer l'effet du changement d'une ou deux ↵
# caractéristiques sur le résultat du modèle.

```

```

# Ils montrent la dépendance entre la cible et un ensemble de caractéristiques ↵
# d'entrée, tout en marginalisant les ↵
# valeurs de toutes les autres caractéristiques.

var = 'age'
interpreter.partial_dependence.plot_partial_dependence([var], model, ↵
    grid_resolution = 100, ↵
                           with_variance = True, ↵
    figsize = (6, 4))
plt.title('Dépendance partielle de la variable ' + var + ' selon le modèle ↵
    RandomForestClassifier avec SKATER\n');

# Enregistrement de la figure dans le répertoire courant
filename = 'INTERPRETABILITE_DEPENDANCE_PARTIELLE_' + var + '_RDF_SKATER.png'
plt.savefig(filename, dpi = 600, bbox_inches = "tight")

```

9.3 INTERPRETABILITE locale du modèle RandomForestClassifier AVEC SKATER

```

[ ]: ##### Interprétation locale avec LIME
      #####
      # Les explications locales interprétables indépendantes du modèle (LIME, ↵
      # Ribeiro, 2016) sont une autre méthode
      # indépendante du modèle qui peut être utilisée avec des modèles de boîte noire ↵
      # pour expliquer la justification
      # des décisions du modèle. Contrairement aux PDP, cependant, LIME fonctionne à ↵
      # l'échelle locale et l'idée
      # sous-jacente est assez simple. Nous pouvons avoir un classificateur qui a ↵
      # globalement une frontière de décision
      # très complexe, mais si nous nous concentrons sur un seul échantillon, le ↵
      # comportement du modèle dans cette localité
      # spécifique peut généralement être expliqué par un modèle interprétable ↵
      # beaucoup plus simple.

      # Après avoir sélectionné l'échantillon d'intérêt, LIME forme un modèle de ↵
      # substitution en utilisant des perturbations
      # des attributs de l'échantillon sélectionné. Perturber les variables ↵
      # indépendantes avec un peu de bruit et surveiller
      # l'impact sur la variable cible est généralement suffisant pour fournir une ↵
      # bonne explication locale. Le modèle de
      # substitution est souvent un modèle linéaire simple ou un arbre de décision, ↵
      # qui sont interprétables de manière innée,
      # de sorte que les données collectées à partir des perturbations et la sortie ↵
      # de classe correspondante peuvent fournir
      # une bonne indication sur ce qui influence la décision du modèle.

```

```

# Skater fournit une implémentation de LIME, accessible via la classe
# ↪LimeTabularExplainer. Tout ce que nous avons à
# faire est d'instancier LimeTabularExplainer et de lui donner accès aux
# ↪données de formation et aux noms de
# fonctionnalités indépendants.

# Nous reconstruisons le même modèle mais avec des numpy arrays pour éviter les
# ↪problème avec LIME dû aux noms des
# features
from skater.core.local_interpretation.lime.lime_tabular import
    ↪LimeTabularExplainer
from sklearn.ensemble import RandomForestClassifier

clf_rdf = RandomForestClassifier(n_estimators = 800, max_features = 0.5,
    ↪max_depth = 6, bootstrap = True)
clf_rdf.fit(X_train.values, Y_train)
preds = clf_rdf.predict(X_test)
explicateur = LimeTabularExplainer(X_test.values, feature_names = list(X_test.
    ↪columns),
    discretize_continuous = False , mode =
    ↪'classification',
    class_names = ['Non-souscription', ↪
    ↪'Souscription'])

# Ensuite, nous choisissons un échantillon pour lequel nous voulons obtenir une
# ↪explication, disons le treizième
# échantillon de notre ensemble de données de test (ID d'échantillon 13). Nous
# ↪appelons les fonctions
# d'explication_instance à partir de l'explicateur et examinons les classes
# ↪réelles et prédictes, ainsi que
# l'influence correspondante des attributs individuels sur la prédiction.

# On cherche à savoir le numéro d'indice maximum que l'on peut saisir
print('Dimension de X_test', X_test.shape, '\n\n')
# X_test comprend 2228 lignes. On pourrait faire une boucle for pour tester
# ↪chacune d'elle, mais trop long
num_index = 1990
# Extraction de la ligne choisie num_index pour afficher les informations du
# ↪client
print('Le client choisi présente les informations suivantes : \n', pd.
    ↪DataFrame(X_test.iloc[num_index]), '\n\n')
print('Interprétation locale du modèle RandomForestClassifier selon l index :', ↪
    ↪num_index, '\n\n')
print ("Libellé réel : %s" % Y_test.iloc[num_index])
print ("Prédit : %s" % preds[num_index])

```

```
# Application de LIME
explicateur.explain_instance(X_test.iloc[num_index].values, clf_rdf.
    ↪predict_proba, num_features = 10).show_in_notebook()
```

9.4 INTERPRETABILITE globale du modèle StackingClassifier AVEC SKATER

```
[ ]: from skater.model import InMemoryModel
from skater.core.explanations import Interpretation
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, ↪
    ↪GradientBoostingClassifier, StackingClassifier

# Instanciation de tous les classifieurs
# On met les meilleurs paramètres retenus dans les modèles précédents
clf_log = LogisticRegression(C = 0.1)
clf_random_f = RandomForestClassifier(n_estimators = 800, max_features = 0.5, ↪
    ↪max_depth = 6, bootstrap = True)
clf_grad_boos = GradientBoostingClassifier(n_estimators = 900, max_depth = 6, ↪
    ↪learning_rate = 0.1)
sclf = StackingClassifier(estimators = [('LOGISTIC', clf_log), ↪
    ↪('RANDOM_FOREST', clf_random_f),
                           ('GRADIENT_BOOSTING', clf_grad_boos)], ↪
    ↪final_estimator = clf_grad_boos)

sclf.fit(X_train, Y_train)

##### Interprétabilité globale avec la sélection de variables ↪
    ↪effectuée au préalable #####
# Explications :

# Maintenant que nous avons mis en place le modèle de la boîte noire, nous ↪
    ↪pouvons essayer plusieurs techniques pour
# comprendre quels sont les principaux moteurs derrière les décisions du modèle.
    ↪ Comme ce modèle est accessible
# localement, il suffit de créer un objet InMemoryModel. Le seul argument ↪
    ↪obligatoire pour un InMemoryModel est la
# fonction de génération de prédiction. Dans notre cas, il s'agit de ↪
    ↪predict_proba du Keras ANN.
# Nous créons ensuite un objet Interpretation qui transmet les données à la ↪
    ↪fonction de prédiction. Nous limitons
# notre interprétation à 11200 échantillons pour calculer l'importance, et nous ↪
    ↪demandons également que les
# caractéristiques soient triées par importance dans un ordre croissant.
```

```

# La mise en œuvre du calcul de l'importance des attributs est basée sur
# ↪l'analyse d'importance variable (VIA).
# Skater utilise différentes techniques selon le type de modèle (par exemple
# ↪régression, classification multi-classes,
# etc.), mais il s'appuie généralement sur la mesure de l'entropie dans le
# ↪changement des prédictions compte tenu d'une
# perturbation d'une caractéristique. Voir Wei et al. (2015) pour plus de
# ↪détails.

# Skater est un package prônant l'interprétation globale et locale de toutes
# ↪formes de modèles, afin d'aider à construire
# des systèmes de Machine Learning interprétables pour des cas d'utilisation
# ↪dans le monde réel. Skater utilise uniquement
# un jeu de données, ainsi qu'une fonction de prédition entraînée sur ces
# ↪données pour ces interprétations.

# On utilise X_train.values pour éviter code erreur lié au dataframe pandas
model = InMemoryModel(sclf.predict_proba, examples = X_train.values,
# ↪target_names = ['Non-souscription', 'Souscription'])
interpreter = Interpretation(X_train.values, feature_names = X_train.columns,
# ↪training_labels = Y_train)
plots = interpreter.feature_importance.feature_importance(model, n_jobs = -1,
# ↪ascending = True, n_samples = 11200)
# Affichage des 36 variables les plus importantes du modèle
plots.tail(36).plot.barh(figsize = (10, 12))
plt.title('Interprétabilité globale du modèle StackingClassifier avec
# ↪SKATER\n');

# Enregistrement de la figure dans le répertoire courant
filename = 'INTERPRETABILITE_GLOBALE_SCLF_SKATER.png'
plt.savefig(filename, dpi = 600, bbox_inches = "tight")

#####
##### Dépendance partielle
#####

# Explications :

# La dépendance partielle décrit l'impact marginal d'une caractéristique sur la
# ↪prédiction du modèle,
# en maintenant les autres caractéristiques constantes dans le modèle. La
# ↪dérivée de la dépendance
# partielle décrit l'impact d'une caractéristique (analogue à un coefficient de
# ↪caractéristique dans
# un modèle de régression).

```

```

# Partial Dependence Plot est une autre méthode visuelle, qui est indépendante du modèle et peut être utilisée avec
# succès pour mieux comprendre le fonctionnement interne d'un modèle de boîte noire comme un ANN profond. Les PDP
# sont un outil efficace pour évaluer l'effet du changement d'une ou deux caractéristiques sur le résultat du modèle.
# Ils montrent la dépendance entre la cible et un ensemble de caractéristiques d'entrée, tout en marginalisant les
# valeurs de toutes les autres caractéristiques.

var = 'age'
interpreter.partial_dependence.plot_partial_dependence([var], model,
    grid_resolution = 100,
    with_variance = True,
    figsize = (6, 4))
plt.title('Dépendance partielle de la variable ' + var + ' selon le modèle StackingClassifier avec SKATER\n');

# Enregistrement de la figure dans le répertoire courant
filename = 'INTERPRETABILITE_DEPENDANCE_PARTIELLE_' + var + '_SCLF_SKATER.png'
plt.savefig(filename, dpi = 600, bbox_inches = "tight")

```

9.5 INTERPRETABILITE locale du modèle StackingClassifier AVEC SKATER

```

[ ]: ##### Interprétation locale avec LIME
      #####
      # Les explications locales interprétables indépendantes du modèle (LIME, Ribeiro, 2016) sont une autre méthode
      # indépendante du modèle qui peut être utilisée avec des modèles de boîte noire pour expliquer la justification
      # des décisions du modèle. Contrairement aux PDP, cependant, LIME fonctionne à l'échelle locale et l'idée
      # sous-jacente est assez simple. Nous pouvons avoir un classificateur qui a globalement une frontière de décision
      # très complexe, mais si nous nous concentrons sur un seul échantillon, le comportement du modèle dans cette localité
      # spécifique peut généralement être expliqué par un modèle interprétable beaucoup plus simple.

      # Après avoir sélectionné l'échantillon d'intérêt, LIME forme un modèle de substitution en utilisant des perturbations
      # des attributs de l'échantillon sélectionné. Perturber les variables indépendantes avec un peu de bruit et surveiller

```

```

# l'impact sur la variable cible est généralement suffisant pour fournir une
# bonne explication locale. Le modèle de
# substitution est souvent un modèle linéaire simple ou un arbre de décision,
# qui sont interprétables de manière innée,
# de sorte que les données collectées à partir des perturbations et la sortie
# de classe correspondante peuvent fournir
# une bonne indication sur ce qui influence la décision du modèle.

# Skater fournit une implémentation de LIME, accessible via la classe
# LimeTabularExplainer. Tout ce que nous avons à
# faire est d'instancier LimeTabularExplainer et de lui donner accès aux
# données de formation et aux noms de
# fonctionnalités indépendants.

# Nous reconstruisons le même modèle mais avec des numpy arrays pour éviter les
# problème avec LIME dû aux noms des
# features
from skater.model import InMemoryModel
from skater.core.explanations import Interpretation
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
    GradientBoostingClassifier, StackingClassifier

# Instanciation de tous les classifieurs
# On met les meilleurs paramètres retenus dans les modèles précédents
clf_log = LogisticRegression(C = 0.1)
clf_random_f = RandomForestClassifier(n_estimators = 800, max_features = 0.5,
    max_depth = 6, bootstrap = True)
clf_grad_boos = GradientBoostingClassifier(n_estimators = 900, max_depth = 6,
    learning_rate = 0.1)
sclf = StackingClassifier(estimators = [('LOGISTIC', clf_log),
    ('RANDOM_FOREST', clf_random_f),
    ('GRADIENT_BOOSTING', clf_grad_boos)],
    final_estimator = clf_grad_boos)

sclf.fit(X_train.values, Y_train)
preds = sclf.predict(X_test)
explicateur = LimeTabularExplainer(X_test.values, feature_names = list(X_test.
    columns),
    discretize_continuous = False , mode =
    'classification',
    class_names = ['Non-souscription',
    'Souscription'])

# Ensuite, nous choisissons un échantillon pour lequel nous voulons obtenir une
# explication, disons le treizième

```

```

# échantillon de notre ensemble de données de test (ID d'échantillon 13). Nous ↴
    ↴appelons les fonctions
# d'explication_instance à partir de l'explicateur et examinons les classes ↴
    ↴réelles et prédictives, ainsi que
# l'influence correspondante des attributs individuels sur la prédiction.

# On cherche à savoir le numéro d'indice maximum que l'on peut saisir
print('Dimension de X_test', X_test.shape, '\n\n')
# X_test comprend 2228 lignes. On pourrait faire une boucle for pour tester ↴
    ↴chacune d'elle, mais trop long
num_index = 1990
# Extraction de la ligne choisie num_index pour afficher les informations du ↴
    ↴client
print('Le client choisi présente les informations suivantes : \n', pd.
    ↴DataFrame(X_test.iloc[num_index]), '\n\n')
print('Interprétation locale du modèle stackingClassifier selon l index :', ↴
    ↴num_index, '\n\n')
print ("Libellé réel : %s" % Y_test.iloc[num_index])
print ("Prédit : %s" % preds[num_index])
# Application de LIME
explicateur.explain_instance(X_test.iloc[num_index].values, clf_rdf.
    ↴predict_proba, num_features = 10).show_in_notebook()

```

9.6 INTERPRETABILITE globale du modèle LogisticRegression AVEC SKATER

```

[ ]: from skater.model import InMemoryModel
from skater.core.explanations import Interpretation
from sklearn.linear_model import LogisticRegression

# Instanciation du classifieur
# On met les meilleurs paramètres retenus dans les modèles précédents
clf_log = LogisticRegression(C = 0.1)

# Entraînement du modèle
clf_log.fit(X_train, Y_train)

##### Interprétabilité globale avec la sélection de variables ↴
    ↴effectuée au préalable #####
# Explications :

# Maintenant que nous avons mis en place le modèle de la boîte noire, nous ↴
    ↴pouvons essayer plusieurs techniques pour
# comprendre quels sont les principaux moteurs derrière les décisions du modèle.
    ↴ Comme ce modèle est accessible

```

```

# localement, il suffit de créer un objet InMemoryModel. Le seul argument ↴
    ↴obligatoire pour un InMemoryModel est la
# fonction de génération de prédiction. Dans notre cas, il s'agit de ↴
    ↴predict_proba du Keras ANN.
# Nous créons ensuite un objet Interpretation qui transmet les données à la ↴
    ↴fonction de prédiction. Nous limitons
# notre interprétation à 11200 échantillons pour calculer l'importance, et nous ↴
    ↴demandons également que les
# caractéristiques soient triées par importance dans un ordre croissant.
# La mise en œuvre du calcul de l'importance des attributs est basée sur ↴
    ↴l'analyse d'importance variable (VIA).
# Skater utilise différentes techniques selon le type de modèle (par exemple ↴
    ↴régression, classification multi-classes,
# etc.), mais il s'appuie généralement sur la mesure de l'entropie dans le ↴
    ↴changement des prédictions compte tenu d'une
# perturbation d'une caractéristique. Voir Wei et al. (2015) pour plus de ↴
    ↴détails.

# Skater est un package prônant l'interprétation globale et locale de toutes ↴
    ↴formes de modèles, afin d'aider à construire
# des systèmes de Machine Learning interprétables pour des cas d'utilisation ↴
    ↴dans le monde réel. Skater utilise uniquement
# un jeu de données, ainsi qu'une fonction de prédiction entraînée sur ces ↴
    ↴données pour ces interprétations.

# On utilise X_train.values pour éviter code erreur lié au dataframe pandas
model = InMemoryModel(clf_log.predict_proba, examples = X_train.values, ↴
    ↴target_names = ['Non-souscription', 'Souscription'])
interpreter = Interpretation(X_train.values, feature_names = X_train.columns, ↴
    ↴training_labels = Y_train)
plots = interpreter.feature_importance.feature_importance(model, n_jobs = -1, ↴
    ↴ascending = True, n_samples = 11200)
# Affichage des 36 variables les plus importantes du modèle
plots.tail(36).plot.barh(figsize = (10, 12))
plt.title('Interprétabilité globale du modèle LogisticRegression avec ↴
    ↴SKATER\n');

# Enregistrement de la figure dans le répertoire courant
filename = 'INTERPRETABILITE_GLOBALE_LOG_SKATER.png'
plt.savefig(filename, dpi = 600, bbox_inches = "tight")

#####
##### Dépendance partielle #####
#####

# Explications :

```

```

# La dépendance partielle décrit l'impact marginal d'une caractéristique sur la
# prédiction du modèle,
# en maintenant les autres caractéristiques constantes dans le modèle. La
# dérivée de la dépendance
# partielle décrit l'impact d'une caractéristique (analogue à un coefficient de
# caractéristique dans
# un modèle de régression).

# Partial Dependence Plot est une autre méthode visuelle, qui est indépendante
# du modèle et peut être utilisée avec
# succès pour mieux comprendre le fonctionnement interne d'un modèle de boîte
# noire comme un ANN profond. Les PDP
# sont un outil efficace pour évaluer l'effet du changement d'une ou deux
# caractéristiques sur le résultat du modèle.
# Ils montrent la dépendance entre la cible et un ensemble de caractéristiques
# d'entrée, tout en marginalisant les
# valeurs de toutes les autres caractéristiques.

var = 'age'
interpreter.partial_dependence.plot_partial_dependence([var], model,
grid_resolution = 100,
with_variance = True,
figsize = (6, 4))
plt.title('Dépendance partielle de la variable ' + var + ' selon le modèle'
LogisticRegression avec SKATER\n);

# Enregistrement de la figure dans le répertoire courant
filename = 'INTERPRETABILITE_DEPENDANCE_PARTIELLE_' + var + '_LOG_SKATER.png'
plt.savefig(filename, dpi = 600, bbox_inches = "tight")

```

9.7 INTERPRETABILITE locale du modèle LogisticRegression AVEC SKATER

```

[ ]: ##### Interprétation locale avec LIME
##### Ribeiro, 2016)

# Les explications locales interprétables indépendantes du modèle (LIME,
# indépendante du modèle qui peut être utilisée avec des modèles de boîte noire
# pour expliquer la justification
# des décisions du modèle. Contrairement aux PDP, cependant, LIME fonctionne à
# sous-jacente est assez simple. Nous pouvons avoir un classificateur qui a
# très complexe, mais si nous nous concentrons sur un seul échantillon, le
# comportement du modèle dans cette localité

```

```

# spécifique peut généralement être expliqué par un modèle interprétable ↴
↳ beaucoup plus simple.

# Après avoir sélectionné l'échantillon d'intérêt, LIME forme un modèle de ↴
↳ substitution en utilisant des perturbations
# des attributs de l'échantillon sélectionné. Perturber les variables ↴
↳ indépendantes avec un peu de bruit et surveiller
# l'impact sur la variable cible est généralement suffisant pour fournir une ↴
↳ bonne explication locale. Le modèle de
# substitution est souvent un modèle linéaire simple ou un arbre de décision, ↴
↳ qui sont interprétables de manière innée,
# de sorte que les données collectées à partir des perturbations et la sortie ↴
↳ de classe correspondante peuvent fournir
# une bonne indication sur ce qui influence la décision du modèle.

# Skater fournit une implémentation de LIME, accessible via la classe ↴
↳ LimeTabularExplainer. Tout ce que nous avons à
# faire est d'instancier LimeTabularExplainer et de lui donner accès aux ↴
↳ données de formation et aux noms de
# fonctionnalités indépendants.

# Nous reconstruisons le même modèle mais avec des numpy arrays pour éviter les ↴
↳ problème avec LIME dû aux noms des
# features
from skater.model import InMemoryModel
from skater.core.explanations import Interpretation
from sklearn.linear_model import LogisticRegression

# Instanciation du classifieur
# On met les meilleurs paramètres retenus dans les modèles précédents
clf_log = LogisticRegression(C = 0.1)

# Entraînement du modèle
clf_log.fit(X_train.values, Y_train)
preds = clf_log.predict(X_test)
explicateur = LimeTabularExplainer(X_test.values, feature_names = list(X_test.
↳ columns),
discretize_continuous = False , mode = ↴
↳ 'classification',
class_names = ['Non-souscription', ↴
↳ 'Souscription'])

# Ensuite, nous choisissons un échantillon pour lequel nous voulons obtenir une ↴
↳ explication, disons le treizième
# échantillon de notre ensemble de données de test (ID d'échantillon 13). Nous ↴
↳ appelons les fonctions

```

```

# d'explication_instance à partir de l'explicateur et examinons les classes ↴
    ↴réelles et prédites, ainsi que
# l'influence correspondante des attributs individuels sur la prédition.

# On cherche à savoir le numéro d'indice maximum que l'on peut saisir
print('Dimension de X_test', X_test.shape, '\n\n')
# X_test comprend 2228 lignes. On pourrait faire une boucle for pour tester ↴
    ↴chacune d'elle, mais trop long
num_index = 1990
# Extraction de la ligne choisie num_index pour afficher les informations du ↴
    ↴client
print('Le client choisi présente les informations suivantes : \n', pd.
    ↴DataFrame(X_test.iloc[num_index]), '\n\n')
print('Interprétation locale du modèle LogisticRegression selon l index :', ↴
    ↴num_index, '\n\n')
print ("Libellé réel : %s" % Y_test.iloc[num_index])
print ("Prédit : %s" % preds[num_index])
# Application de LIME
explicateur.explain_instance(X_test.iloc[num_index].values, clf_log.
    ↴predict_proba, num_features = 10).show_in_notebook()

```

9.8 INTERPRETABILITE globale du modèle GradientBoostingClassifier AVEC SKATER

```

[ ]: from skater.model import InMemoryModel
from skater.core.explanations import Interpretation
from sklearn.ensemble import GradientBoostingClassifier

# Instanciation du classifieur
# On met les meilleurs paramètres retenus dans les modèles précédents
clf_grad = GradientBoostingClassifier(n_estimators = 900, max_depth = 6, ↴
    ↴learning_rate = 0.1)

# Entraînement du modèle
clf_grad.fit(X_train, Y_train)

##### Interprétabilité globale avec la sélection de variables ↴
    ↴effectuée au préalable #####
# Explications :

# Maintenant que nous avons mis en place le modèle de la boîte noire, nous ↴
    ↴pouvons essayer plusieurs techniques pour
# comprendre quels sont les principaux moteurs derrière les décisions du modèle.
    ↴ Comme ce modèle est accessible

```

```

# localement, il suffit de créer un objet InMemoryModel. Le seul argument ↴
↪ obligatoire pour un InMemoryModel est la
# fonction de génération de prédiction. Dans notre cas, il s'agit de ↴
↪ predict_proba du Keras ANN.
# Nous créons ensuite un objet Interpretation qui transmet les données à la ↴
↪ fonction de prédiction. Nous limitons
# notre interprétation à 11200 échantillons pour calculer l'importance, et nous ↴
↪ demandons également que les
# caractéristiques soient triées par importance dans un ordre croissant.
# La mise en œuvre du calcul de l'importance des attributs est basée sur ↴
↪ l'analyse d'importance variable (VIA).
# Skater utilise différentes techniques selon le type de modèle (par exemple ↴
↪ régression, classification multi-classes,
# etc.), mais il s'appuie généralement sur la mesure de l'entropie dans le ↴
↪ changement des prédictions compte tenu d'une
# perturbation d'une caractéristique. Voir Wei et al. (2015) pour plus de ↴
↪ détails.

# Skater est un package prônant l'interprétation globale et locale de toutes ↴
↪ formes de modèles, afin d'aider à construire
# des systèmes de Machine Learning interprétables pour des cas d'utilisation ↴
↪ dans le monde réel. Skater utilise uniquement
# un jeu de données, ainsi qu'une fonction de prédiction entraînée sur ces ↴
↪ données pour ces interprétations.

# On utilise X_train.values pour éviter code erreur lié au dataframe pandas
model = InMemoryModel(clf_grad.predict_proba, examples = X_train.values, ↴
↪ target_names = ['Non-souscription', 'Souscription'])
interpreter = Interpretation(X_train.values, feature_names = X_train.columns, ↴
↪ training_labels = Y_train)
plots = interpreter.feature_importance.feature_importance(model, n_jobs = -1, ↴
↪ ascending = True, n_samples = 11200)
# Affichage des 36 variables les plus importantes du modèle
plots.tail(36).plot.barh(figsize = (10, 12))
plt.title('Interprétabilité globale du modèle GradientBoostingClassifier avec ↴
↪ SKATER\n');

# Enregistrement de la figure dans le répertoire courant
filename = 'INTERPRETABILITE_GLOBALE_GRAD_SKATER.png'
plt.savefig(filename, dpi = 600, bbox_inches = "tight")

#####
##### Dépendance partielle #####
#####

# Explications :

```

```

# La dépendance partielle décrit l'impact marginal d'une caractéristique sur la
# prédiction du modèle,
# en maintenant les autres caractéristiques constantes dans le modèle. La
# dérivée de la dépendance
# partielle décrit l'impact d'une caractéristique (analogue à un coefficient de
# caractéristique dans
# un modèle de régression).

# Partial Dependence Plot est une autre méthode visuelle, qui est indépendante
# du modèle et peut être utilisée avec
# succès pour mieux comprendre le fonctionnement interne d'un modèle de boîte
# noire comme un ANN profond. Les PDP
# sont un outil efficace pour évaluer l'effet du changement d'une ou deux
# caractéristiques sur le résultat du modèle.
# Ils montrent la dépendance entre la cible et un ensemble de caractéristiques
# d'entrée, tout en marginalisant les
# valeurs de toutes les autres caractéristiques.

var = 'age'
interpreter.partial_dependence.plot_partial_dependence([var], model,
grid_resolution = 100,
with_variance = True,
figsize = (6, 4))
plt.title('Dépendance partielle de la variable ' + var + ' selon le modèle
GradientBoostingClassifier avec SKATER\n');

# Enregistrement de la figure dans le répertoire courant
filename = 'INTERPRETABILITE_DEPENDANCE_PARTIELLE_' + var + '_GRAD_SKATER.png'
plt.savefig(filename, dpi = 600, bbox_inches = "tight")

```

9.9 INTERPRETABILITE locale du modèle GradientBoostingClassifier AVEC SKATER

```

[ ]: ##### Interprétation locale avec LIME
##### Ribeiro, 2016)

# Les explications locales interprétables indépendantes du modèle (LIME,
# indépendante du modèle qui peut être utilisée avec des modèles de boîte noire
# pour expliquer la justification
# des décisions du modèle. Contrairement aux PDP, cependant, LIME fonctionne à
# sous-jacente est assez simple. Nous pouvons avoir un classificateur qui a
# globalement une frontière de décision
# très complexe, mais si nous nous concentrons sur un seul échantillon, le
# comportement du modèle dans cette localité

```

```

# spécifique peut généralement être expliqué par un modèle interprétable ↴
↪ beaucoup plus simple.

# Après avoir sélectionné l'échantillon d'intérêt, LIME forme un modèle de ↴
↪ substitution en utilisant des perturbations
# des attributs de l'échantillon sélectionné. Perturber les variables ↴
↪ indépendantes avec un peu de bruit et surveiller
# l'impact sur la variable cible est généralement suffisant pour fournir une ↴
↪ bonne explication locale. Le modèle de
# substitution est souvent un modèle linéaire simple ou un arbre de décision, ↴
↪ qui sont interprétables de manière innée,
# de sorte que les données collectées à partir des perturbations et la sortie ↴
↪ de classe correspondante peuvent fournir
# une bonne indication sur ce qui influence la décision du modèle.

# Skater fournit une implémentation de LIME, accessible via la classe ↴
↪ LimeTabularExplainer. Tout ce que nous avons à
# faire est d'instancier LimeTabularExplainer et de lui donner accès aux ↴
↪ données de formation et aux noms de
# fonctionnalités indépendants.

# Nous reconstruisons le même modèle mais avec des numpy arrays pour éviter les ↴
↪ problème avec LIME dû aux noms des
# features
from skater.model import InMemoryModel
from skater.core.explanations import Interpretation
from sklearn.ensemble import GradientBoostingClassifier

# Instanciation du classifieur
# On met les meilleurs paramètres retenus dans les modèles précédents
clf_grad = GradientBoostingClassifier(n_estimators = 900, max_depth = 6, ↴
↪ learning_rate = 0.1)

# Entraînement du modèle
clf_grad.fit(X_train, Y_train)
preds = clf_grad.predict(X_test)
explicateur = LimeTabularExplainer(X_test.values, feature_names = list(X_test. ↴
↪ columns),
                                    discretize_continuous = False , mode = ↴
↪ 'classification',
                                    class_names = ['Non-souscription', ↴
↪ 'Souscription'])

# Ensuite, nous choisissons un échantillon pour lequel nous voulons obtenir une ↴
↪ explication, disons le treizième

```

```

# échantillon de notre ensemble de données de test (ID d'échantillon 13). Nous ↴
    ↴appelons les fonctions
# d'explication_instance à partir de l'explicateur et examinons les classes ↴
    ↴réelles et prédictives, ainsi que
# l'influence correspondante des attributs individuels sur la prédiction.

# On cherche à savoir le numéro d'indice maximum que l'on peut saisir
print('Dimension de X_test', X_test.shape, '\n\n')
# X_test comprend 2228 lignes. On pourrait faire une boucle for pour tester ↴
    ↴chacune d'elle, mais trop long
num_index = 1990
# Extraction de la ligne choisie num_index pour afficher les informations du ↴
    ↴client
print('Le client choisi présente les informations suivantes : \n', pd.
    ↴DataFrame(X_test.iloc[num_index]), '\n\n')
print('Interprétation locale du modèle GradientBoostingClassifier selon l index ↴
    ↴:', num_index, '\n\n')
print ("Libellé réel : %s" % Y_test.iloc[num_index])
print ("Prédit : %s" % preds[num_index])
# Application de LIME
explicateur.explain_instance(X_test.iloc[num_index].values, clf_grad.
    ↴predict_proba, num_features = 10).show_in_notebook()

```