Projet R - Car Insurance classification

Guillaume Lesvesque & Valentin Maillo

Janvier 2022

Table des matières

1 - Présentation du jeu de données

1.1 - Description du jeu de données

Nous avons traité un ensemble de données provenant d'une banque aux États-Unis.

Cette banque fournit également des services d'assurance automobile et organise régulièrement des campagnes pour attirer de nouveaux clients. Les employés de la banque appellent les clients potentiels afin de leur faire connaître les options d'assurance automobile disponibles.

Nous avons des informations générales sur les clients (âge, emploi, etc.) ainsi que des informations plus spécifiques sur la campagne de vente d'assurance en cours (type de communication, dernier jour de contact, ...) et sur les campagnes précédentes (informations sur les tentatives précédentes et leurs résultats).

Nous disposons des données de 4 000 clients qui ont été contactés lors de la dernière campagne et pour lesquels les résultats de la campagne (le client a-t-il acheté une assurance ou non) sont connus. Nous disposons aussi d'un second jeu de données de 1000 autres clients, celui-ci ne comporte pas le résultat de la campagne actuelle.

L'objectif que nous nous sommes fixé est de prédire pour 1000 clients l'achat ou non une assurance automobile. Toute notre démarche, du pre-processing à la présentation de notre meilleur modèle en passant par une analyse descriptive des caractéristiques de notre population, a été détaillée dans les pages suivantes.

Nous avons utilisé le jeu de données disponible ici : https://www.kaggle.com/kondla/carinsurance .

1.2 - Tableau de description des variables

Variable	Description	Exemple
Id	Numéro unique d'identification attribué à chaque individu.	'1' … '5000'
Age	L'âge du client.	
Job	La catégorie professionnelle du client.	'admin' (administration), 'blue-collar' (ouvrier), 'entrepreneur', 'housemaid' (femme de chambre), 'management', 'retired' (retraité), 'self-employed' (travailleur indépendant), 'services', 'student' (étudiant), 'technician' (technicien), 'unemployed' (sans emploi)
Marital	La situation maritale du client.	'divorced' (divorcé), 'married' (marié), 'single' (célibataire)
Education	Le niveau d'étude du client.	'primary' (primaire), 'secondary' (collège-lycée), 'tertiary' (supérieur)
Default	Le client a déjà eu un défaut de paiement auprès de la banque.	'oui' 1 , 'non' 0
Balance	Solde annuel moyen en dollars américain	
HHInsurance	Le client a-t-il une assurance habitation ?	'oui' 1 , 'non' 0
CarLoan	Le client a-t-il un prêt automobile ?	'oui' 1 , 'non' 0
Communication	Type de communication utilisé.	'cellular'(téléphone portable), 'telephone' (téléphone fixe), 'NA'
LastContactMonth	Le dernier mois pendant lequel le client a été contacté.	'jan' , 'feb', …
LastContactDay	Le dernier jour pendant lequel le client a été contacté.	
CallStart	L'heure à laquelle l'appel a commencé	12:43:15
CallEnd	L'heure à laquelle l'appel a fini	12:43:15
NoOfContacts	Nombre de fois que le client a été contacté durant cette campagne.	
DaysPassed	Nombre de jours écoulés après que le client a été contacté pour la dernière fois lors de la précédente campagne (-1 signifie que le client n'a jamais été contacté).	
PrevAttempts	Nombre de fois que le client a été contacté avant cette campagne.	
Outcome	Résultat de la dernière campagne.	'failure' (échec), 'success' (succès), 'other'(autre)
CarInsurance	Le client a-t-il souscrit à une assurance automobile ?	'oui' 1 , 'non' 0

2 - Importation des jeux de données train et test

```
data_train =  read.csv(file = 'carInsurance_train.csv')
data_test =  read.csv(file = 'carInsurance_test.csv')
```

CarInsurance est la variable que l'on souhaite prédire.

```
y_train = (select(data_train, CarInsurance))
y_test = (select(data_test, CarInsurance))
```

Le problème ici est que nous n'avons pas la target sur l'échantillon de test.

Après quelques recherches sur internet nous avons trouvé l'output d'une modélisation qui semble sérieuse : https://www.kaggle.com/yonatanrabinovich/car-insurance-cold-calls .

Nous utiliserons ce dernier pour évaluer notre méthode.

```
data_from_serious_forecast = read.csv(file = "Insurance Purchase Forcast.csv")
y_test = select(data_from_serious_forecast, CarInsurance)
y_test = as.numeric(y_test == "wiil buy insurance")
data_test["CarInsurance"] = y_test
```

3 - Prise en main du jeu de données

Dimension des jeux de données :

```
nrow(data_train)
```

```
## [1] 4000
```

```
ncol(data_train)
```

```
## [1] 19
```

```
nrow(data_test)
```

```
## [1] 1000
```

```
ncol(data_test)
```

```
## [1] 19
```

Nombre de données manquantes au sein de la target.

```
length(y_train[is.na(y_train)])
```

```
## [1] 0
```

```r
length(y_test[is.na(y_test)])
```

```
## [1] 0
```

Il n'y a donc aucune valeur manquante au sein de la target, son pré-processing est donc terminé.

3.1 - Enrichissement du jeu de données

En observant les variables explicatives nous remarquons que CallEnd et CallStart sont inutilisables telles quelles. Nous allons plutôt garder la durée de l'appel et le moment dans la journée où il a été passé.

```r
translating_in_sec_train = t(matrix(rep(c(3600,60,1),dim(data_train)[1]),nrow=3))
translating_in_sec_test = t(matrix(rep(c(3600,60,1),dim(data_test)[1]),nrow=3))

call_end_train = strsplit(data_train$CallEnd,':')
call_end_train = t(matrix(as.numeric(unlist(call_end_train)),nrow=3))
call_end_train = call_end_train * translating_in_sec_train
call_end_train_final = call_end_train[,1]+call_end_train[,2]+call_end_train[,3]

call_start_train = strsplit(data_train$CallStart,':')
call_start_train = t(matrix(as.numeric(unlist(call_start_train)),nrow=3))
when_call_start_train = call_start_train[,1]
call_start_train = call_start_train * translating_in_sec_train
call_start_train_final = call_start_train[,1]+call_start_train[,2]+call_start_train[,3]

call_end_test = strsplit(data_test$CallEnd,':')
call_end_test = t(matrix(as.numeric(unlist(call_end_test)),nrow=3))
call_end_test = call_end_test * translating_in_sec_test
call_end_test_final = call_end_test[,1]+call_end_test[,2]+call_end_test[,3]

call_start_test = strsplit(data_test$CallStart,':')
call_start_test = t(matrix(as.numeric(unlist(call_start_test)),nrow=3))
when_call_start_test = call_start_test[,1]
call_start_test = call_start_test * translating_in_sec_test
call_start_test_final = call_start_test[,1]+call_start_test[,2]+call_start_test[,3]

call_duration_test = call_end_test_final - call_start_test_final
call_duration_train =  call_end_train_final - call_start_train_final
```

Afin d'anticiper par avance les problèmes d'échelles nous allons passer l'unité de mesure des variables Call_duration en minutes.

```r
call_duration_test = call_duration_test/60
call_duration_train = call_duration_train/60
```

Voici les tableaux des heures de la journée où le dernier contact téléphonique a eu lieu.

```
knitr::kable(table(when_call_start_train))
```

when_call_start_train	Freq
9	458
10	455
11	371
12	444
13	464
14	457
15	448
16	454
17	449

```
knitr::kable(table(when_call_start_test))
```

when_call_start_test	Freq
9	114
10	130
11	105
12	101
13	117
14	125
15	108
16	104
17	96

Nous avons donc créé une variable qualitative à 8 modalités.

```
data_train$CallStart <- NULL
data_train$CallEnd <- NULL
data_train["CallDuration"] =  call_duration_train
data_train["WhenIsTheCall"] =  when_call_start_train


data_test$CallStart <- NULL
data_test$CallEnd <- NULL
data_test["CallDuration"] =  call_duration_test
data_test["WhenIsTheCall"] =  when_call_start_test
```

3.2 - Traitement des variables mal définies

Nous allons tout d'abord traiter les variables LasContactMonth et LastContactDay :

Nous aurions aimer pouvoir construire une nouvelle variable "jour de la semaine" afin de pouvoir déterminer si par exemple les personnes contactées un lundi sont plus susptible de souscrire à notre assurance plûtot que les personnes qui seront contacté un vendredi.

Cependant, nous n'avons pas accès à l'année du dernier contact. De plus, la variable DayPassed ne nous permet pas de déduire le jour de création de la base de données. Voici une illustration de ce problème :

```r
index_test_daypassed = which(data_train$DaysPassed != -1 & data_train$DaysPassed < 10)
days_passed_recently = data_train[index_test_daypassed,]
days_passed_recently[1:3, c("LastContactDay", "LastContactMonth", "DaysPassed")]
```

```
##     LastContactDay LastContactMonth DaysPassed
## 156              4              aug          1
## 582              4              aug          1
## 626             30              jan          2
```

Nous remarquons sur les deux premières lignes qu'il y a un jour qui s'est écoulé depuis le dernier contact. De plus, le dernier contact était le 4 août ainsi, la base de données aurait été créer le 5 août.

Cependant, lorsque l'on regarde la 3ème ligne on voit que le dernier contact était le 30 janvier et la variable DaysPassed nous indique qu'il y a 2 jours qui se sont écoulés depuis cette date.

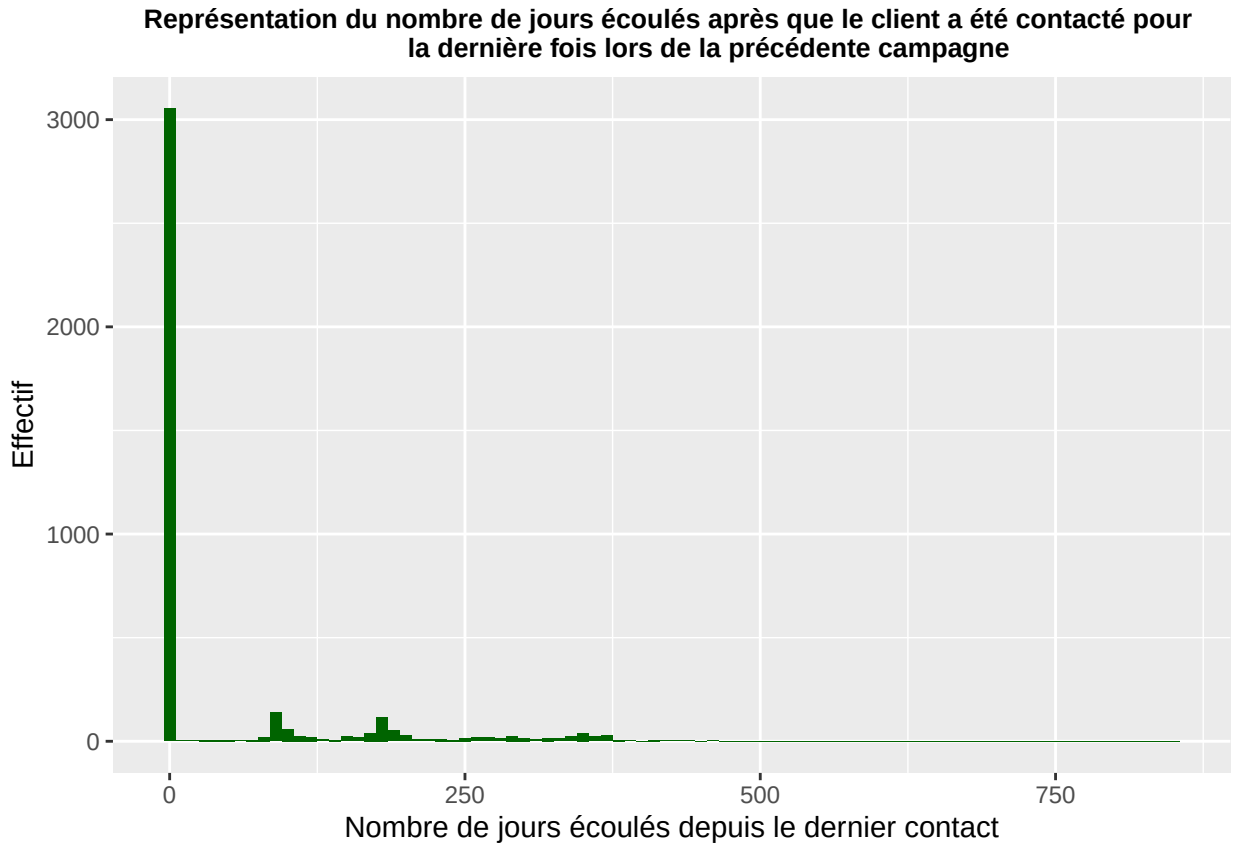
Ainsi, nous devrions être le 1er février, ce qui est incohérent avec les deux premières lignes.

Nous allons supprimer les variables LastContactDay et LastContactMonth qui sont de ce fait inutilisables.

```r
data_test$LastContactDay <- NULL
data_train$LastContactDay <- NULL

data_test$LastContactMonth <- NULL
data_train$LastContactMonth <- NULL
```

De même la variable DaysPassed va poser problème sans transformation de notre part. En effet, la valeur -1 est rentrée pour un individu jamais contacté. Sinon cette variable nous donne le nombre de jours écoulés depuis la dernière démarche. Pour résoudre ce problème et rendre la variable plus adaptée et utilisable pour une modélisation future nous allons la transformer en une variable catégorielle. Tout d'abord, observons sa distribution :

```r
ggplot(data_train, aes(x = DaysPassed) ) +
   geom_histogram(binwidth = 10, fill = "#006400") +
   ggtitle("Représentation du nombre de jours écoulés après que le client a été contacté pour
          la dernière fois lors de la précédente campagne") +
   theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 10)) +
   scale_x_continuous(name ="Nombre de jours écoulés depuis le dernier contact") +
   scale_y_continuous(name ="Effectif")
```

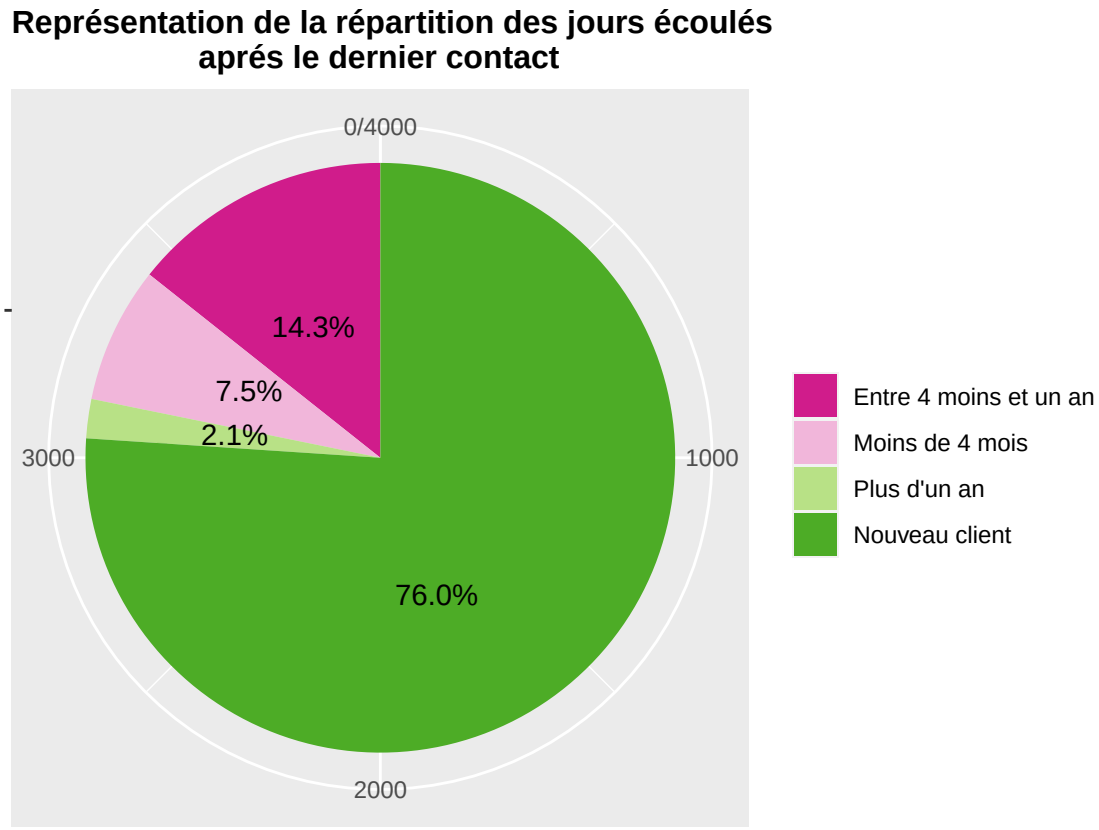


Suivant cet histogramme nous allons construire 4 modalités : “Jamais contacté”, “Contacté il y a moins de 4 mois”, “Contacté entre 4 mois et un an” et “Contacté il y a plus d’un an”.

```
ind_new_client = which(data_train$DaysPassed == -1)
ind_4_month = which(data_train$DaysPassed != -1 & data_train$DaysPassed < 120)
ind_4_month_year = which(data_train$DaysPassed >= 120 & data_train$DaysPassed < 365)
ind_more_year = which(data_train$DaysPassed != -1 & data_train$DaysPassed >= 365)

data_train$DaysPassed[ind_new_client] = "New_Client"
data_train$DaysPassed[ind_4_month] = "Less_than_4_month"
data_train$DaysPassed[ind_4_month_year] = "Between_4_month_and_1_year"
data_train$DaysPassed[ind_more_year] = "More_than_a_year"


ind_new_client_test = which(data_test$DaysPassed == -1)
ind_4_month_test = which(data_test$DaysPassed != -1 & data_test$DaysPassed < 120)
ind_4_month_year_test = which(data_test$DaysPassed >= 120 & data_test$DaysPassed < 365)
ind_more_year_test = which(data_test$DaysPassed != -1 & data_test$DaysPassed >= 365)


data_test$DaysPassed[ind_new_client_test] = "New_Client"
data_test$DaysPassed[ind_4_month_test] = "Less_than_4_month"
data_test$DaysPassed[ind_4_month_year_test] = "Between_4_month_and_1_year"
data_test$DaysPassed[ind_more_year_test] = "More_than_a_year"
```

Voici la répartition post-traitement de cette variable :

```
tableau = table(factor(data_train$DaysPassed))
ggplot(data_train, aes(x = factor(1) , fill = factor(DaysPassed)) ) +
  geom_bar(width = 1) + coord_polar("y", start = 0)  +
  ggtitle("Représentation de la répartition des jours écoulés
                    aprés le dernier contact") +
  theme(plot.title = element_text(hjust = 0, face = "bold", size = 12),
        axis.title = element_blank(), legend.title = element_blank(),
        axis.text.y = element_blank() ) +
  scale_fill_brewer(palette = "PiYG", labels = c("Entre 4 moins et un an", "Moins de 4 mois",
                                                 "Plus d'un an", "Nouveau client")) +
  annotate(geom="text", x=1, y=1750, label = percent(as.vector(tableau/nrow(data_train)))[4]) +
   annotate(geom="text", x=1, y=3100, label = percent(as.vector(tableau/nrow(data_train)))[3]) +
   annotate(geom="text", x=1, y=3300, label = percent(as.vector(tableau/nrow(data_train)))[2]) +
   annotate(geom="text", x=1, y=3700, label = percent(as.vector(tableau/nrow(data_train)))[1])
```

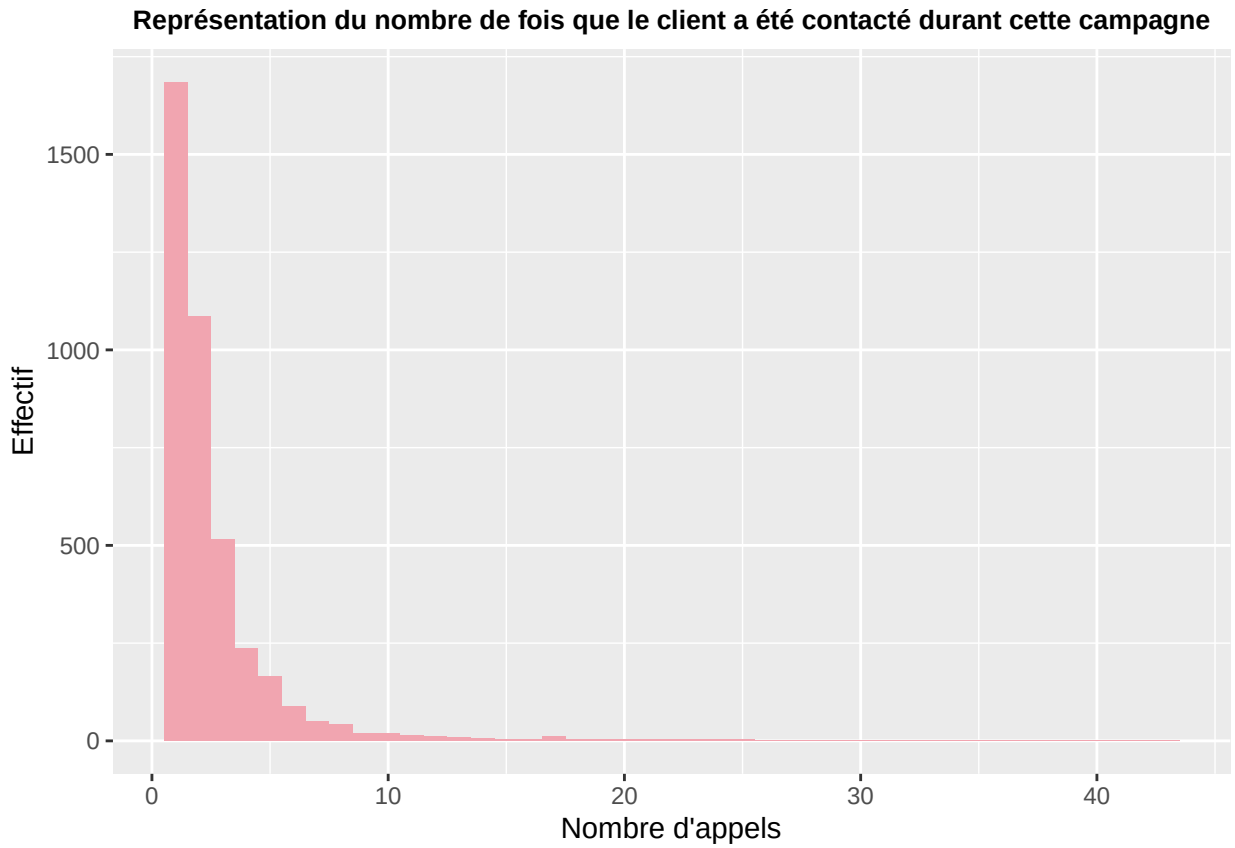


Une nouvelle fois, et suivant le même principe, nous allons passer NoOfContacts en variable catégorielle afin d'éviter le sur-apprentissage.

En effet, nous ne souhaitons pas que les modèles créés par la suite fassent une différence entre une personne contacter 34 fois et une personne contactée 8 fois. Dans les deux cas, nous pouvons considérer que cette personne sera lassée de répondre aux appels alors que le modèle risque d'interpréter la première personne comme "plus de 4 fois plus lassée" que la deuxième.

Commençons donc par observer la distribution de NoOfContacts :

```r
ggplot(data_train, aes(x = NoOfContacts) ) +
   geom_histogram(binwidth = 1, fill = "#F1A5B0") +
   ggtitle("Représentation du nombre de fois que le client a été contacté durant cette campagne") +
   theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 10)) +
   scale_x_continuous(name ="Nombre d'appels") +
   scale_y_continuous(name ="Effectif")
```



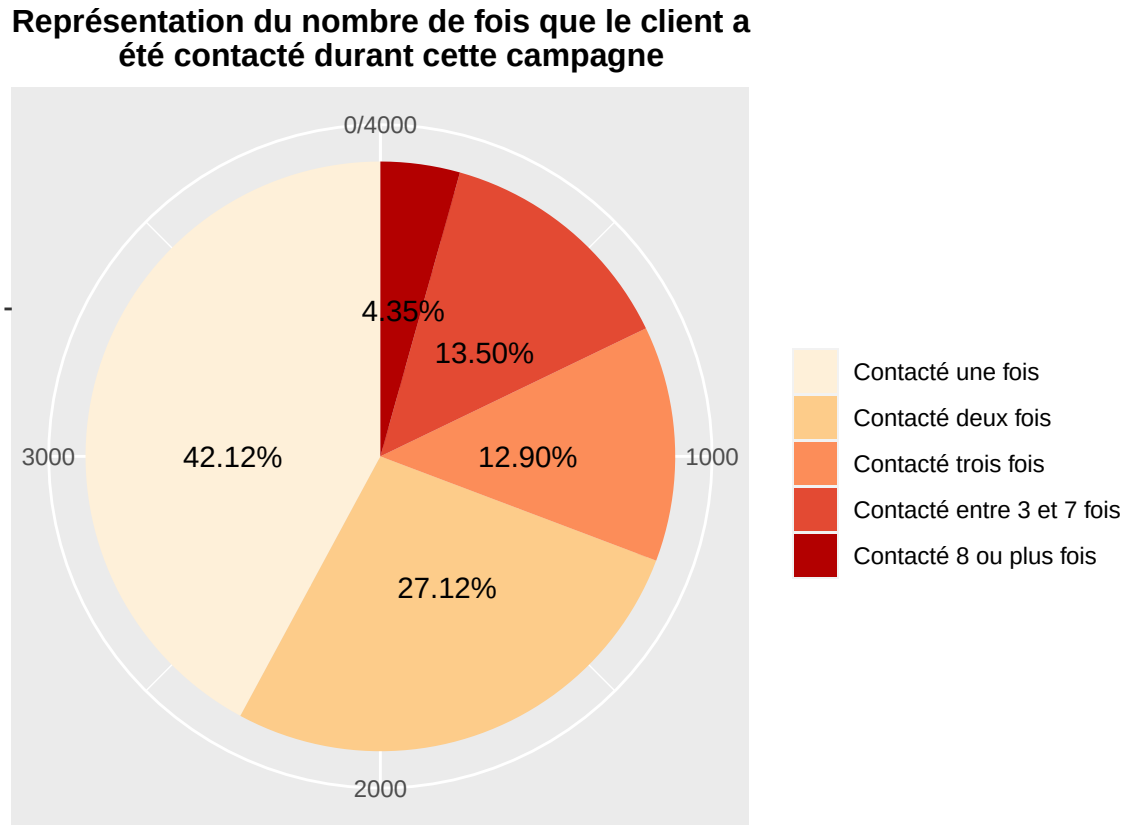
Suivant cet histogramme, nous allons construire 5 modalités : “Contacté 1 fois”, “Contacté 2 fois”, “Contacté 3 fois”, “Contacté entre 4 et 7 fois”, “Contacté plus de 7 fois”.

```r
ind_nb_contact_1 = which(data_train$NoOfContacts == 1)
ind_nb_contact_2 = which(data_train$NoOfContacts == 2)
ind_nb_contact_3 = which(data_train$NoOfContacts == 3)
ind_nb_contact_4 = which(data_train$NoOfContacts >= 4 & data_train$NoOfContacts <=7 )
ind_nb_contact_5 = which(data_train$NoOfContacts >= 8)

data_train$NoOfContacts[ind_nb_contact_1] = "1_Contact"
data_train$NoOfContacts[ind_nb_contact_2] = "2_Contact"
data_train$NoOfContacts[ind_nb_contact_3] = "3_Contact"
data_train$NoOfContacts[ind_nb_contact_4] = "4_7Contact"
data_train$NoOfContacts[ind_nb_contact_5] = "8_Contact"

ind_nb_contact_1_test = which(data_test$NoOfContacts == 1)
ind_nb_contact_2_test = which(data_test$NoOfContacts == 2)
ind_nb_contact_3_test = which(data_test$NoOfContacts == 3)
```

```
ind_nb_contact_4_test = which(data_test$NoOfContacts >= 4 & data_test$NoOfContacts <=7 )
ind_nb_contact_5_test = which(data_test$NoOfContacts >= 8)

data_test$NoOfContacts[ind_nb_contact_1_test] = "1_Contact"
data_test$NoOfContacts[ind_nb_contact_2_test] = "2_Contact"
data_test$NoOfContacts[ind_nb_contact_3_test] = "3_Contact"
data_test$NoOfContacts[ind_nb_contact_4_test] = "4_7Contact"
data_test$NoOfContacts[ind_nb_contact_5_test] = "8_Contact"
```

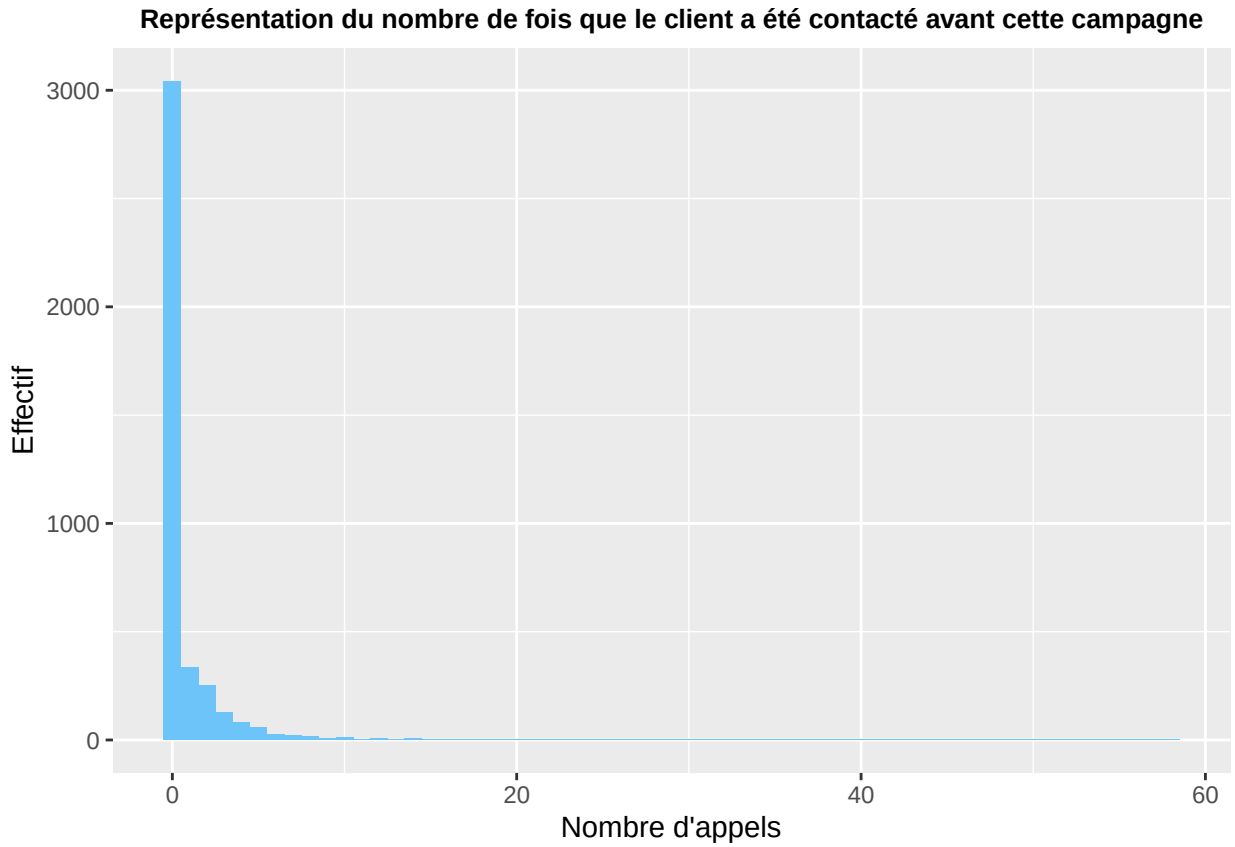
Voici la répartition post-traitement de cette variable :

```
tableau = table(factor(data_train$NoOfContacts))
ggplot(data_train, aes(x = factor(1) , fill = factor(NoOfContacts)) ) +
  geom_bar(width = 1) + coord_polar("y", start=0)  +
  ggtitle("Représentation du nombre de fois que le client a
            été contacté durant cette campagne") +
  theme(plot.title = element_text(hjust = 0, face = "bold", size = 12),
        axis.title = element_blank(), legend.title = element_blank(),
        axis.text.y = element_blank()) +
   scale_fill_brewer(palette = "OrRd", labels = c("Contacté une fois", "Contacté deux fois",
                                                  "Contacté trois fois", "Contacté entre 3 et 7 fois",
                                                  "Contacté 8 ou plus fois")) +
  annotate(geom="text", x=1, y=100, label = percent(as.vector(tableau/nrow(data_train)))[5]) +
   annotate(geom="text", x=1, y=500, label = percent(as.vector(tableau/nrow(data_train)))[4]) +
   annotate(geom="text", x=1, y=1000, label = percent(as.vector(tableau/nrow(data_train)))[3]) +
   annotate(geom="text", x=1, y=1700, label = percent(as.vector(tableau/nrow(data_train)))[2]) +
   annotate(geom="text", x=1, y=3000, label = percent(as.vector(tableau/nrow(data_train)))[1])
```



Pour finir, suivant la même démarche, nous allons passer PrevAttempts en une variable categorielle.

```
ggplot(data_train, aes(x = PrevAttempts) ) +
   geom_histogram(binwidth = 1, fill = "#6DC4F9") +
   ggtitle("Représentation du nombre de fois que le client a été contacté avant cette campagne") +
   theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 10)) +
   scale_x_continuous(name ="Nombre d'appels") +
   scale_y_continuous(name ="Effectif")
```

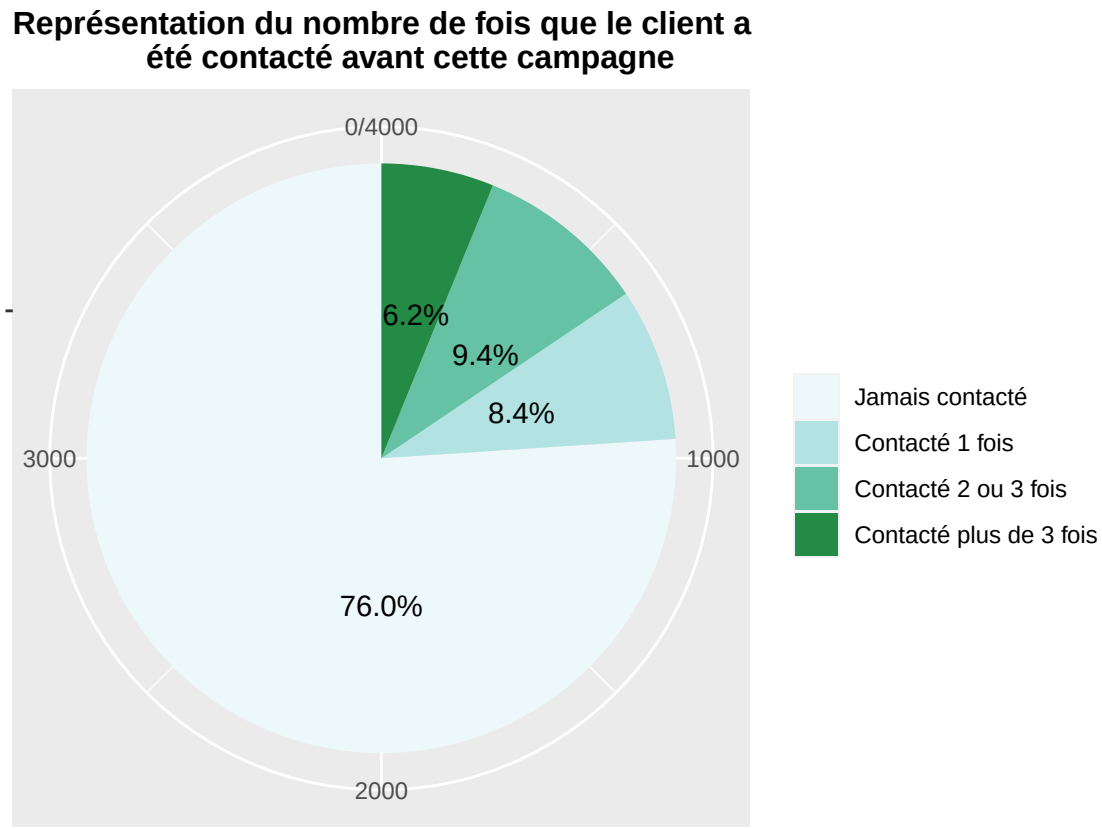


Suivant cet histogramme nous allons construire 4 modalités : “Jamais contacté”, “Contacté 1 fois”, “Contacté 2 ou 3 fois”, “Contacté plus de 3 fois”

```
ind_nb_prevA_0 = which(data_train$PrevAttempts == 0)
ind_nb_prevA_1 = which(data_train$PrevAttempts == 1)
ind_nb_prevA_2 = which(data_train$PrevAttempts >= 2 & data_train$PrevAttempts <= 3)
ind_nb_prevA_3 = which(data_train$PrevAttempts > 3)

data_train$PrevAttempts[ind_nb_prevA_0] = "0_PrevC"
data_train$PrevAttempts[ind_nb_prevA_1] = "1_PrevC"
data_train$PrevAttempts[ind_nb_prevA_2] = "2_3PrevC"
data_train$PrevAttempts[ind_nb_prevA_3] = "4_PrevC"

ind_nb_prevA_0_test = which(data_test$PrevAttempts == 0)
ind_nb_prevA_1_test = which(data_test$PrevAttempts == 1)
ind_nb_prevA_2_test = which(data_test$PrevAttempts >= 2 & data_test$PrevAttempts <= 3)
ind_nb_prevA_3_test = which(data_test$PrevAttempts > 3)

data_test$PrevAttempts[ind_nb_prevA_0_test] = "0_PrevC"
data_test$PrevAttempts[ind_nb_prevA_1_test] = "1_PrevC"
data_test$PrevAttempts[ind_nb_prevA_2_test] = "2_3PrevC"
data_test$PrevAttempts[ind_nb_prevA_3_test] = "4_PrevC"
```

Dont voici la répartition post-traitement de cette variable :

```
tableau = table(factor(data_train$PrevAttempts))
ggplot(data_train, aes(x = factor(1) , fill = factor(PrevAttempts)) ) +
  geom_bar(width = 1) + coord_polar("y", start=0) +
  ggtitle("Représentation du nombre de fois que le client a
              été contacté avant cette campagne") +
  theme(plot.title = element_text(hjust = 0, face = "bold", size = 12),
        axis.title = element_blank(), legend.title = element_blank(),
        axis.text.y = element_blank()) +
   scale_fill_brewer(palette = "BuGn", labels =
                       c("Jamais contacté", "Contacté 1 fois",
                         "Contacté 2 ou 3 fois","Contacté plus de 3 fois")) +
   annotate(geom="text", x=1, y=150, label = percent(as.vector(tableau/nrow(data_train)))[4]) +
   annotate(geom="text", x=1, y=500, label = percent(as.vector(tableau/nrow(data_train)))[3]) +
   annotate(geom="text", x=1, y=800, label = percent(as.vector(tableau/nrow(data_train)))[2]) +
   annotate(geom="text", x=1, y=2000, label = percent(as.vector(tableau/nrow(data_train)))[1])
```



Nous pouvons continuer notre pre-processing par la déclaration de toute variable qualitative en facteur. Commençons par identifier les variables qualitatives déclarées comme “character”.

```
datab_train = select_if(data_train,is.character)
Names_train = names(datab_train)
datab_test = select_if(data_test,is.character)
Names_test = names(datab_test)
```

Enfin transformons les en facteurs :

```
data_train[,Names_train] = lapply(data_train[,Names_train], as.factor)
data_test[,Names_test] = lapply(data_test[,Names_test], as.factor)
```

3.3 - Traitement des valeurs manquantes

Consultation d'une ligne au hasard, ici la 345ème au sein du jeu de données :

```
data_train[345,]
```

```
##      Id Age      Job  Marital Education Default Balance HHInsurance CarLoan
## 345 345  33 services divorced secondary       0    -297           1       0
##     Communication NoOfContacts DaysPassed PrevAttempts Outcome CarInsurance
## 345          <NA>    2_Contact New_Client      0_PrevC    <NA>            0
##     CallDuration WhenIsTheCall
## 345          2.7            17
```

Cela nous fait nous interroger sur les colonnes Communication et Outcome et plus globalement sur la présence de valeurs manquantes. Créons une fonction permettant de référencer les valeurs manquantes.

```
is_there_na <- function(database){
  names_col = names(database)
  for(i in 1:(length(database))){
    msg_to_display = paste("La colonne ", names_col[i], " contient ",
                           as.character(sum(is.na(database[,i]))), " valeur(s) manquante(s)")
    print(msg_to_display)
  }
}


is_there_na(data_train)
```

```
## [1] "La colonne  Id  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Age  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Job  contient  19  valeur(s) manquante(s)"
## [1] "La colonne  Marital  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Education  contient  169  valeur(s) manquante(s)"
## [1] "La colonne  Default  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Balance  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  HHInsurance  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  CarLoan  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Communication  contient  902  valeur(s) manquante(s)"
## [1] "La colonne  NoOfContacts  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  DaysPassed  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  PrevAttempts  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Outcome  contient  3042  valeur(s) manquante(s)"
## [1] "La colonne  CarInsurance  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  CallDuration  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  WhenIsTheCall  contient  0  valeur(s) manquante(s)"
```

```r
is_there_na(data_test)
```

```
## [1] "La colonne  Id  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Age  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Job  contient  5  valeur(s) manquante(s)"
## [1] "La colonne  Marital  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Education  contient  47  valeur(s) manquante(s)"
## [1] "La colonne  Default  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Balance  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  HHInsurance  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  CarLoan  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Communication  contient  221  valeur(s) manquante(s)"
## [1] "La colonne  NoOfContacts  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  DaysPassed  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  PrevAttempts  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Outcome  contient  757  valeur(s) manquante(s)"
## [1] "La colonne  CarInsurance  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  CallDuration  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  WhenIsTheCall  contient  0  valeur(s) manquante(s)"
```

Ainsi, nous allons devoir traiter les valeurs manquantes des variables Job, Education, Communication et Outcome.

Après consultation de la table donnant la définition de chaque variable, nous nous rendons compte que les valeurs manquantes de Outcome correspondent a un client n'ayant encore jamais été contacté. Ainsi, il nous suffit juste de remplacer les NA par une nouvelle modalité "Nouveau client".

```r
data_test["Outcome"]=lapply(data_test["Outcome"], as.character)
data_train["Outcome"]=lapply(data_train["Outcome"], as.character)

data_train$Outcome =data_train$Outcome %>% replace_na("NewClient")
data_test$Outcome = data_test$Outcome %>% replace_na("NewClient")

data_test["Outcome"]=lapply(data_test["Outcome"],  as.factor)
data_train["Outcome"]=lapply(data_train["Outcome"],  as.factor)
```

Concernant la variable Communication nous allons d'abord regarder si elle a un impact sur la target car au vu de sa description (cf partie 1.2) ça ne devrait pas être le cas. Pour cela, nous allons utiliser les randoms forest pour faire de la feature importance, ceci peut être fait avec la librairie Boruta :

Commençons par supprimer toutes les observations contenant des valeurs manquantes, temporairement, afin d'appliquer les random forest pour feature importance :

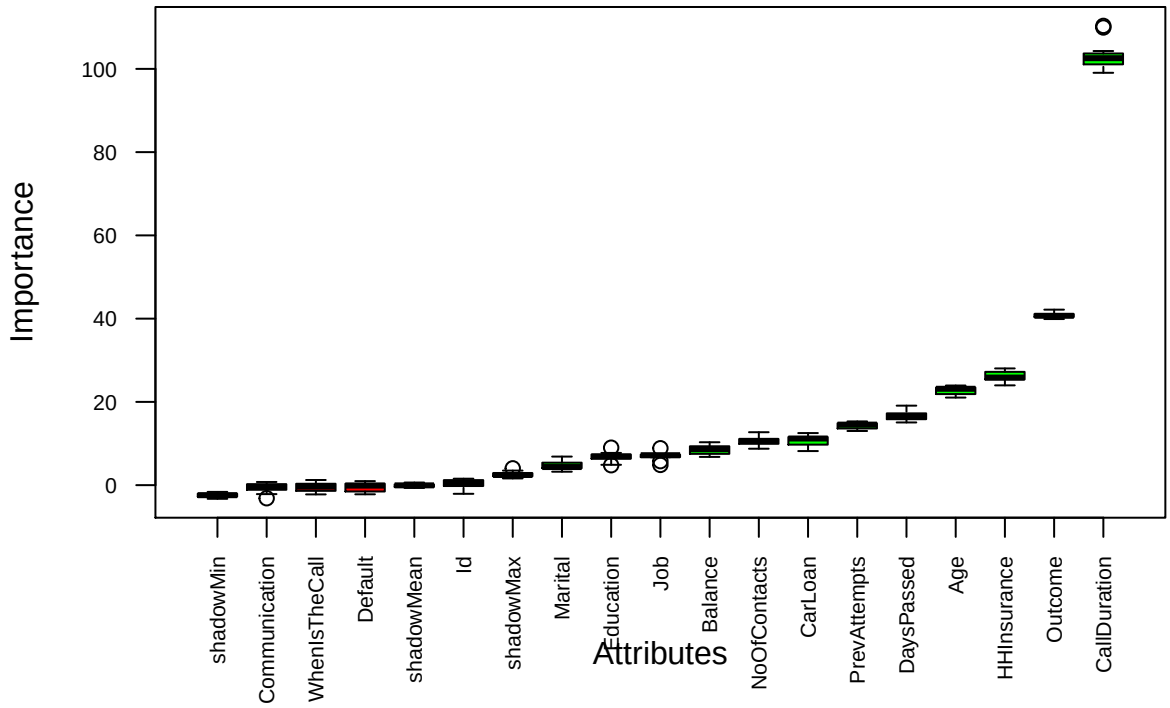
```r
data_train_before_RF = filter(data_train, !is.na(Communication))
data_train_before_RF = filter(data_train_before_RF, !is.na(Job))
data_train_before_RF = filter(data_train_before_RF, !is.na(Education))
```

```r
boruta <- Boruta(CarInsurance ~ ., data = data_train_before_RF, doTrace = 2, maxRuns = 500)
print(boruta)
```

```
## Boruta performed 11 iterations in 20.73596 secs.
##  12 attributes confirmed important: Age, Balance, CallDuration,
```

```
## CarLoan, DaysPassed and 7 more;
##  4 attributes confirmed unimportant: Communication, Default, Id,
## WhenIsTheCall;
```

```
plot(boruta, las = 2, cex.axis = 0.7)
```



On se rend compte que la variable Communication n'a pas "d'importance". Nous allons tout simplement supprimer cette variable ce qui résoud le problème de valeur manquante au sein de cette dernière.

```
data_test$Communication <- NULL
data_train$Communication <- NULL
```

Pour finir, occupons nous des variables Job et Education. Au vu de la définition de ces variables nous allons appliquer une méthode des plus proches voisins.

```
class_mod_Job_test = rpart(Job ~ . -CarInsurance,
                           data = data_test[!is.na(data_test$Job), ], method = "class",
                           na.action = na.omit)
class_mod_Job_train = rpart(Job ~ . -CarInsurance,
                            data = data_train[!is.na(data_train$Job), ], method = "class",
                            na.action = na.omit)

data_train$Job[is.na(data_train$Job)] = predict(class_mod_Job_train,
                                                data_train[is.na(data_train$Job), ], type = "class")
data_test$Job[is.na(data_test$Job)] = predict(class_mod_Job_test,
```

```
                                              data_test[is.na(data_test$Job), ], type = "class")


class_mod_Education_test = rpart(Education ~ . - CarInsurance,
                                 data = data_test[!is.na(data_test$Education), ], method = "class",
                                 na.action = na.omit)
class_mod_Education_train = rpart(Education ~ . - CarInsurance,
                                  data=data_train[!is.na(data_train$Education), ], method = "class",
                                  na.action = na.omit)

data_test$Education[is.na(data_test$Education) ] = predict(class_mod_Education_test,
                                                           data_test[is.na(data_test$Education), ],
                                                           type ="class")
data_train$Education[is.na(data_train$Education) ] = predict(class_mod_Education_train,
                                                             data_train[is.na(data_train$Education), ],
                                                             type ="class")
```

Verifions qu'il n'y a plus de valeurs manquantes.

```
is_there_na(data_train)
```

```
## [1] "La colonne  Id  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Age  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Job  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Marital  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Education  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Default  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Balance  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  HHInsurance  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  CarLoan  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  NoOfContacts  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  DaysPassed  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  PrevAttempts  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Outcome  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  CarInsurance  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  CallDuration  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  WhenIsTheCall  contient  0  valeur(s) manquante(s)"
```

```
is_there_na(data_test)
```

```
## [1] "La colonne  Id  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Age  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Job  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Marital  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Education  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Default  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  Balance  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  HHInsurance  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  CarLoan  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  NoOfContacts  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  DaysPassed  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  PrevAttempts  contient  0  valeur(s) manquante(s)"
```

```
## [1] "La colonne  Outcome  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  CarInsurance  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  CallDuration  contient  0  valeur(s) manquante(s)"
## [1] "La colonne  WhenIsTheCall  contient  0  valeur(s) manquante(s)"
```

Nous pouvons terminer cette partie “prise en main du jeu de données” par un overview rapide des deux datasets obtenus et le passage en “type facteur” des variables explicatives lorsque c’est justifié :

```
Names_train = c("Default", "HHInsurance", "CarLoan", "WhenIsTheCall", "CarInsurance")
Names_test = c("Default", "HHInsurance", "CarLoan", "WhenIsTheCall", "CarInsurance")
data_train[,Names_train] = lapply(data_train[,Names_train], as.factor)
data_test[,Names_test] = lapply(data_test[,Names_test], as.factor)



summary(data_train)
```

```
##        Id            Age                 Job          Marital
##  Min.   :   1   Min.   :18.00   management :895   divorced: 483
##  1st Qu.:1001   1st Qu.:32.00   blue-collar:765   married :2304
##  Median :2000   Median :39.00   technician :670   single  :1213
##  Mean   :2000   Mean   :41.21   admin.     :459
##  3rd Qu.:3000   3rd Qu.:49.00   services   :330
##  Max.   :4000   Max.   :95.00   retired    :250
##                                 (Other)    :631
##      Education    Default     Balance        HHInsurance CarLoan
##  primary  : 561   0:3942   Min.   :-3058.0   0:2029      0:3468
##  secondary:2124   1:  58   1st Qu.:  111.0   1:1971      1: 532
##  tertiary :1315            Median :  551.5
##                            Mean   : 1532.9
##                            3rd Qu.: 1619.0
##                            Max.   :98417.0
##
##      NoOfContacts                       DaysPassed     PrevAttempts
##  1_Contact :1685   Between_4_month_and_1_year: 573   0_PrevC :3042
##  2_Contact :1085   Less_than_4_month         : 299   1_PrevC : 335
##  3_Contact : 516   More_than_a_year          :  86   2_3PrevC: 376
##  4_7Contact: 540   New_Client                :3042   4_PrevC : 247
##  8_Contact : 174
##
##
##       Outcome     CarInsurance  CallDuration      WhenIsTheCall
##  failure  : 437   0:2396       Min.   : 0.08333   13     : 464
##  NewClient:3042   1:1604       1st Qu.: 2.10000   9      : 458
##  other    : 195                Median : 3.86667   14     : 457
##  success  : 326                Mean   : 5.84740   10     : 455
##                                3rd Qu.: 7.66667   16     : 454
##                                Max.   :54.21667   17     : 449
##                                                   (Other):1263
```

```
summary(data_test)
```

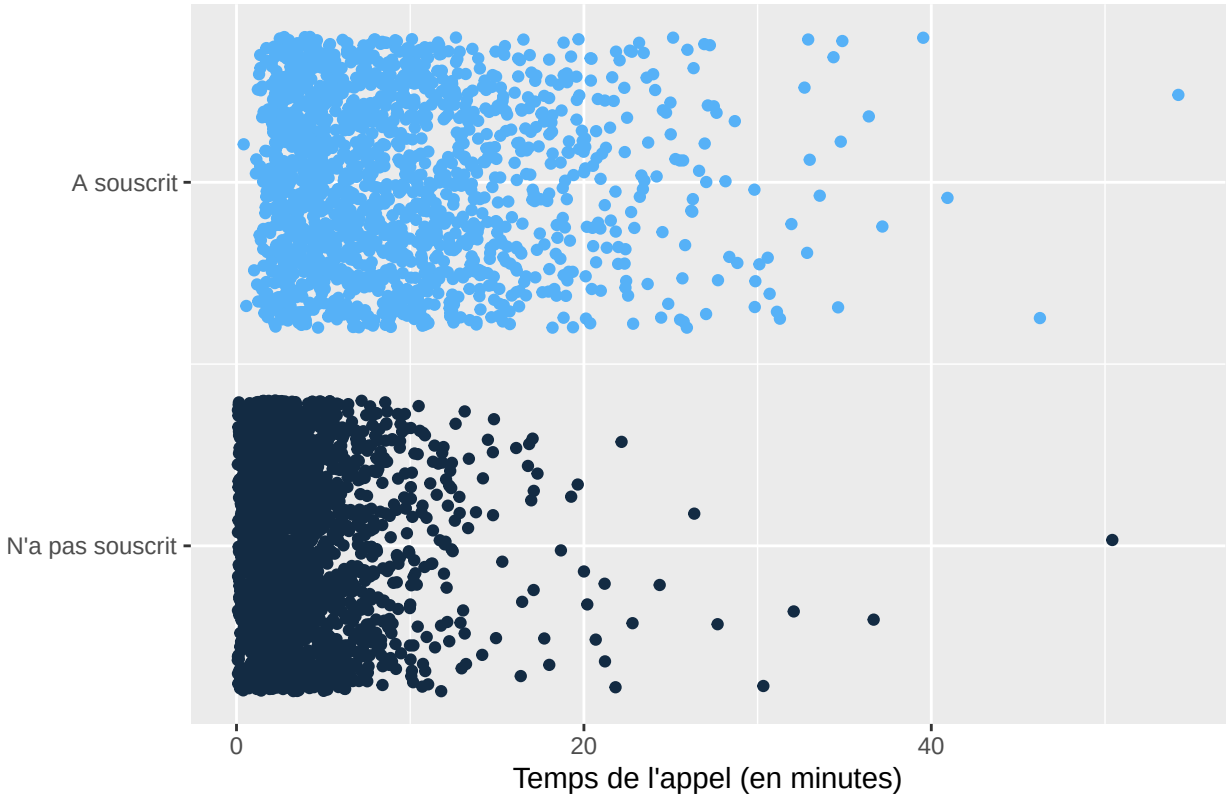
```
##        Id            Age                 Job          Marital
```

```
##  Min.   :4001   Min.   :18.00   management :223   divorced:105
##  1st Qu.:4251   1st Qu.:32.00   blue-collar:178   married :594
##  Median :4500   Median :39.00   technician :165   single  :301
##  Mean   :4500   Mean   :41.47   admin.     :127
##  3rd Qu.:4750   3rd Qu.:49.25   services   : 84
##  Max.   :5000   Max.   :92.00   retired    : 78
##                                 (Other)    :145
##      Education   Default    Balance        HHInsurance CarLoan     NoOfContacts
##  primary  :147   0:987   Min.   :-1980.0   0:487       0:879   1_Contact :438
##  secondary:524   1: 13   1st Qu.:  114.8   1:513       1:121   2_Contact :256
##  tertiary :329           Median :  517.5                       3_Contact :114
##                          Mean   : 1398.3                       4_7Contact:147
##                          3rd Qu.: 1609.8                       8_Contact : 45
##                          Max.   :41630.0
##
##                       DaysPassed    PrevAttempts      Outcome    CarInsurance
##  Between_4_month_and_1_year:159   0_PrevC :757   failure  :111   0:644
##  Less_than_4_month         : 64   1_PrevC : 77   NewClient:757   1:356
##  More_than_a_year          : 20   2_3PrevC: 87   other    : 53
##  New_Client                :757   4_PrevC : 79   success  : 79
##
##
##
##   CallDuration    WhenIsTheCall
##  Min.   : 0.100   10     :130
##  1st Qu.: 2.146   14     :125
##  Median : 3.867   13     :117
##  Mean   : 5.763   9      :114
##  3rd Qu.: 7.487   15     :108
##  Max.   :51.267   11     :105
##                   (Other):301
```

4 - Analyse descriptives de notre population

Visualisons la répartition des deux variables les plus “importantes” d’après la “feature importance” : CallDuration et Outcome.

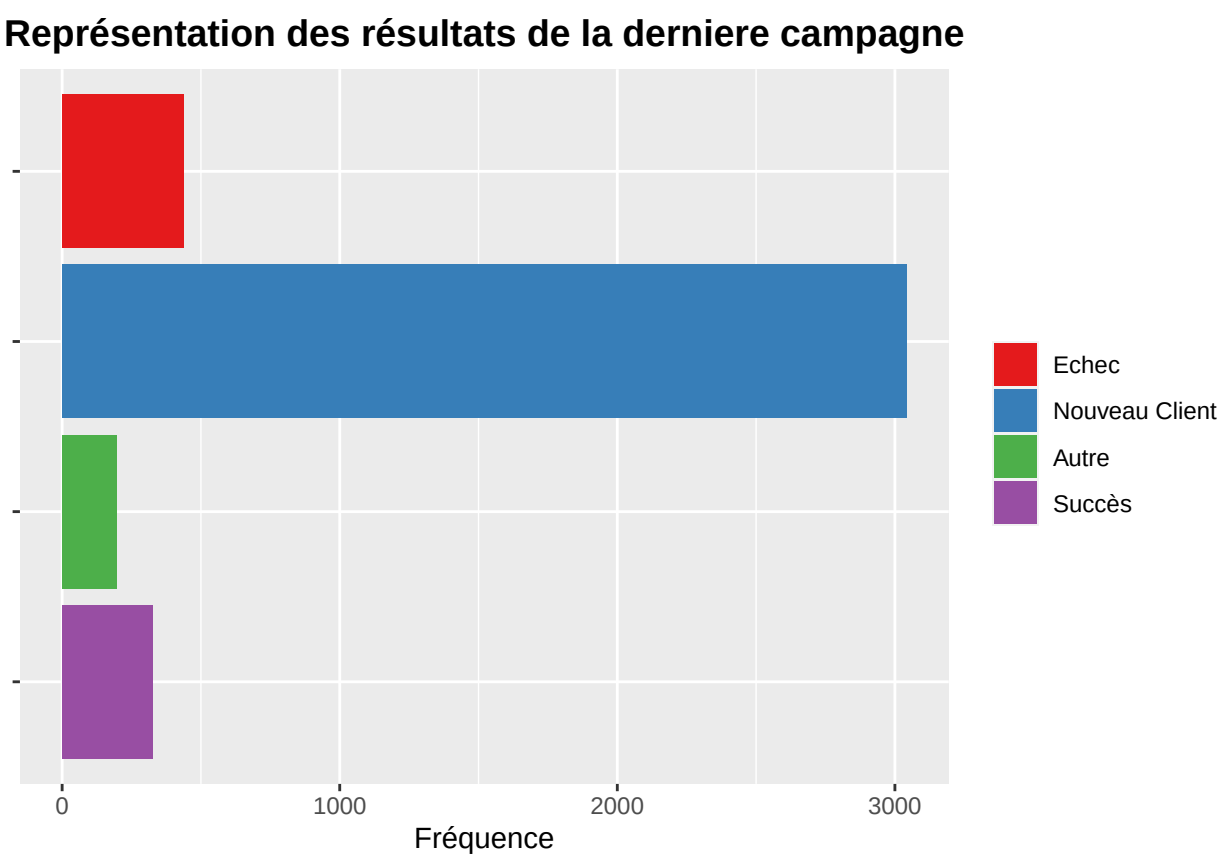
```
ggplot(data_train, aes(y = as.numeric(CarInsurance) , x = CallDuration,
                       colour = as.numeric(CarInsurance)) ) +
   geom_jitter() +
   ggtitle("Représentation de la répartition du temps d'appel en fonction de la variable cible") +
   theme(plot.title = element_text(hjust = 1, face = "bold", size = 12), axis.title.y = element_blank()
         legend.position = "none") +
   labs(x ="Temps de l'appel (en minutes)", fill = "Temps de l'appel (en minutes)") +
   scale_y_continuous(breaks=c(1,2), labels = c("1.0" = "N'a pas souscrit", "2.0" = "A souscrit"))
```



On remarque que les personnes ayant souscrit à une assurance automobile sont restées plus longtemps au téléphone avec les conseillers que les personnes n'ayant pas souscrit, ce qui est plutôt cohérent. Cela vient confirmer le résultat obtenu avec Boruta puisqu'on s'apperçoit que cette variable est clairement discriminante dans la prédiction d'une nouvelle observation.

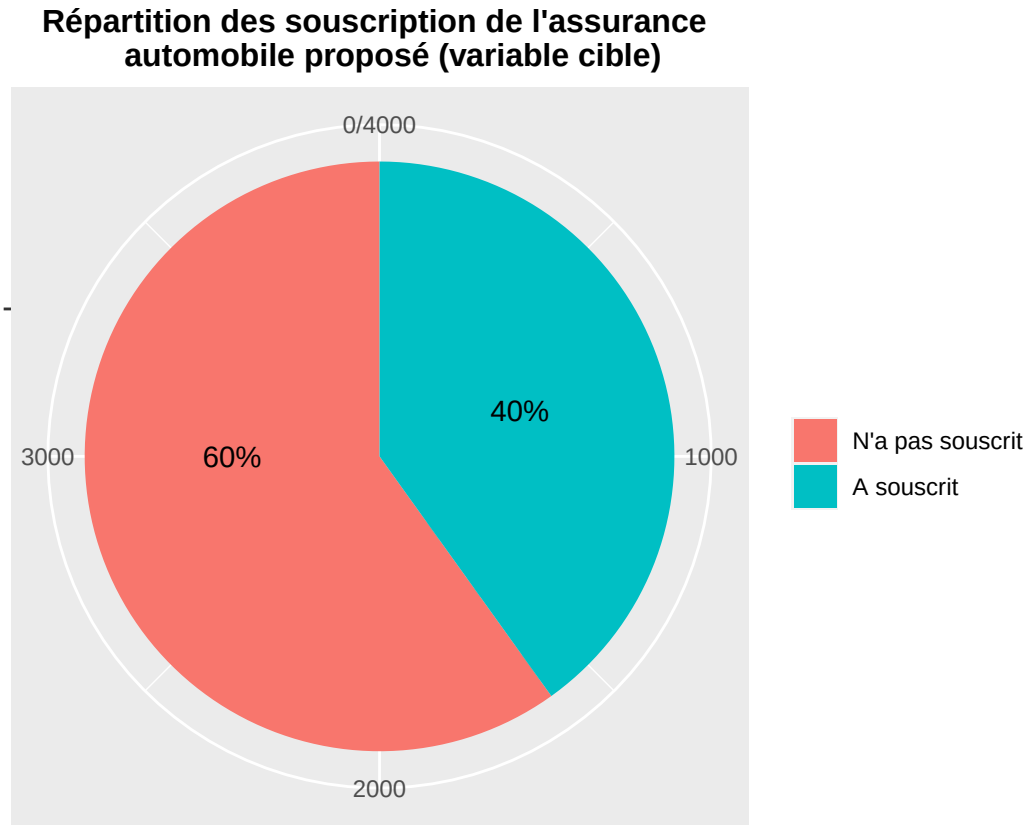
Penchons-nous désormais sur la variable Outcome. Tout d'abord, regardons le résultat de la dernière campagne :

```
outcome = as.data.frame(table(data_train$Outcome))

names(outcome) = c("Modalités", "Fréquence")

ggplot(outcome, aes(x = Modalités , y = Fréquence, fill = Modalités) )  +
  geom_bar(stat="identity") + coord_flip() +
  ggtitle("Représentation des résultats de la derniere campagne") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 14),
        axis.title.y = element_blank(), legend.title = element_blank(),
        axis.text.y = element_blank()) +
   scale_x_discrete(limits=c("success", "other", "NewClient","failure")) +
   scale_fill_brewer(palette="Set1", labels = c("Echec","Nouveau Client","Autre","Succès"))
```



Les modalités “Échec” et “Succès” sont plutôt équilibrés. Determinons s’il en est de même pour la campagne actuelle :

```
tableau = table(factor(data_train$CarInsurance))
ggplot(y_train, aes(x = factor(1) , fill = factor(CarInsurance)) ) +
  geom_bar(width = 1) + coord_polar("y", start=0) +
  ggtitle("Répartition des souscription de l'assurance
   automobile proposé (variable cible)") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 12),
        axis.title = element_blank(), legend.title = element_blank(),
        axis.text.y = element_blank()) +
   scale_fill_discrete(labels = c("N'a pas souscrit", "A souscrit")) +
   annotate(geom="text", x=1, y=800, label = percent(as.vector(tableau/nrow(data_train)))[2]) +
   annotate(geom="text", x=1, y=3000, label = percent(as.vector(tableau/nrow(data_train)))[1])
```



Aucun souci lié à la répartition de la réponse au sein de la target elle n'est donc pas à traiter. Nous n'aurons pas besoin pour ce projet d'adapter les métriques d'évaluation au sein de nos différentes modélisations.

5 - Traitement des variables explicatives qualitatives

Nous allons tout d'abord appliquer une technique de One Hot Encoding à nos variables explicatives qualitatives. Cette méthode consiste à transformer ces variables de type n modalités : 1, 2, 3 … n en n colonnes composées de 0 ou de 1 indiquant si l'observation prend la modalité associée à la colonne.

Le one hot encoding permet donc de supprimer tout potentiel biais dû à l'ordre des modalités en plus de gagner en efficience lors de la période d'apprentissage des différents modèles de machine learning que nous allons implémenter.

```
quali_col <- c("Job", "Marital", "Education", "NoOfContacts", "DaysPassed",
               "PrevAttempts","Outcome","WhenIsTheCall")
data_train <- cbind(data_train,one_hot(as.data.table(data_train[,quali_col])))
data_train[,quali_col] <- NULL

data_test <- cbind(data_test,one_hot(as.data.table(data_test[,quali_col])))
data_test[,quali_col] <- NULL

#Evitons les - qui aboutiront a des problemes de format plus tard

names(data_train)[10] = "Job_blue_collar"
names(data_train)[15] = "Job_self_employed"
```

```r
names(data_test)[10] = "Job_blue_collar"
names(data_test)[15] = "Job_self_employed"

#Transformons maintenant les colonnes du One Hot Encoding en facteur

col_to_factor = names(data_train)[9:51]
data_train[,col_to_factor] = lapply(data_train[,col_to_factor], as.factor)
data_test[,col_to_factor] = lapply(data_test[,col_to_factor], as.factor)

str(data_train)
```

```
## 'data.frame':    4000 obs. of  51 variables:
##  $ Id                                    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Age                                   : int  32 32 29 25 30 32 37 35 30 30 ...
##  $ Default                               : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Balance                               : int  1218 1156 637 373 2694 1625 1000 538 187 3 ...
##  $ HHInsurance                           : Factor w/ 2 levels "0","1": 2 2 2 2 1 1 2 2 2 2 ...
##  $ CarLoan                               : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 2 ...
##  $ CarInsurance                          : Factor w/ 2 levels "0","1": 1 1 2 2 1 2 2 1 1 1 ...
##  $ CallDuration                          : num  1.17 3.08 5.67 13.65 3.2 ...
##  $ Job_admin.                            : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 2 1 2 1 ...
##  $ Job_blue_collar                       : Factor w/ 2 levels "0","1": 1 2 1 1 1 1 1 1 1 2 ...
##  $ Job_entrepreneur                      : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Job_housemaid                         : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Job_management                        : Factor w/ 2 levels "0","1": 2 1 2 1 2 1 1 2 1 1 ...
##  $ Job_retired                           : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Job_self_employed                     : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Job_services                          : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Job_student                           : Factor w/ 2 levels "0","1": 1 1 1 2 1 1 1 1 1 1 ...
##  $ Job_technician                        : Factor w/ 2 levels "0","1": 1 1 1 1 1 2 1 1 1 1 ...
##  $ Job_unemployed                        : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Marital_divorced                      : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 2 1 1 ...
##  $ Marital_married                       : Factor w/ 2 levels "0","1": 1 2 1 1 2 1 1 1 2 2 ...
##  $ Marital_single                        : Factor w/ 2 levels "0","1": 2 1 2 2 1 2 2 1 1 1 ...
##  $ Education_primary                     : Factor w/ 2 levels "0","1": 1 2 1 2 1 1 1 1 1 1 ...
##  $ Education_secondary                   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 2 2 ...
##  $ Education_tertiary                    : Factor w/ 2 levels "0","1": 2 1 2 1 2 2 2 2 1 1 ...
##  $ NoOfContacts_1_Contact                : Factor w/ 2 levels "0","1": 1 1 2 1 2 2 2 1 2 1 ...
##  $ NoOfContacts_2_Contact                : Factor w/ 2 levels "0","1": 2 1 1 2 1 1 1 1 1 2 ...
##  $ NoOfContacts_3_Contact                : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ NoOfContacts_4_7Contact               : Factor w/ 2 levels "0","1": 1 2 1 1 1 1 1 2 1 1 ...
##  $ NoOfContacts_8_Contact                : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ DaysPassed_Between_4_month_and_1_year: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ DaysPassed_Less_than_4_month          : Factor w/ 2 levels "0","1": 1 1 2 1 1 2 1 1 1 1 ...
##  $ DaysPassed_More_than_a_year           : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ DaysPassed_New_Client                 : Factor w/ 2 levels "0","1": 2 2 1 2 2 1 2 2 2 2 ...
##  $ PrevAttempts_0_PrevC                  : Factor w/ 2 levels "0","1": 2 2 1 2 2 1 2 2 2 2 ...
##  $ PrevAttempts_1_PrevC                  : Factor w/ 2 levels "0","1": 1 1 2 1 1 2 1 1 1 1 ...
##  $ PrevAttempts_2_3PrevC                 : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ PrevAttempts_4_PrevC                  : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Outcome_failure                       : Factor w/ 2 levels "0","1": 1 1 2 1 1 2 1 1 1 1 ...
##  $ Outcome_NewClient                     : Factor w/ 2 levels "0","1": 2 2 1 2 2 1 2 2 2 2 ...
##  $ Outcome_other                         : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

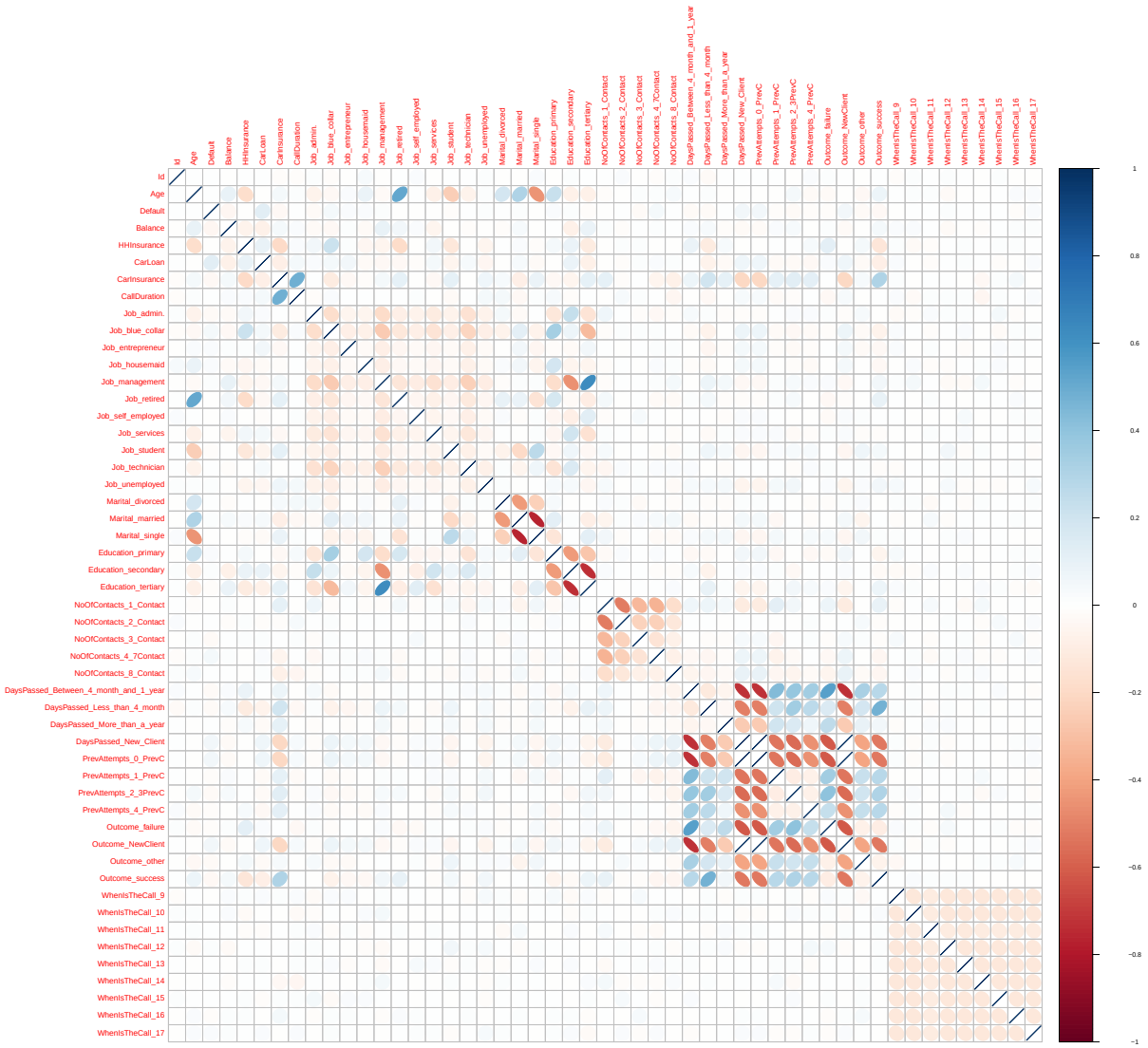
```
##  $ Outcome_success                    : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ WhenIsTheCall_9                    : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 2 ...
##  $ WhenIsTheCall_10                   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ WhenIsTheCall_11                   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ WhenIsTheCall_12                   : Factor w/ 2 levels "0","1": 1 1 1 2 1 1 1 1 2 1 ...
##  $ WhenIsTheCall_13                   : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 2 1 1 1 ...
##  $ WhenIsTheCall_14                   : Factor w/ 2 levels "0","1": 1 2 1 1 2 2 1 1 1 1 ...
##  $ WhenIsTheCall_15                   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 2 1 1 ...
##  $ WhenIsTheCall_16                   : Factor w/ 2 levels "0","1": 1 1 2 1 1 1 1 1 1 1 ...
##  $ WhenIsTheCall_17                   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

```
#Verification ok
```

6 - Sélection des variables

Dans un premier temps, étudions la présence d'éventuelles corrélations entre les variables explicatives.

```
data_train_cor= as.data.frame(lapply(data_train[,names(data_train)], as.numeric))
r = round(cor(data_train_cor),2)
corrplot(r,method="ellipse")
```



On remarque tout d'abord une corrélation non-négligeable de la target avec la plupart des variables que nous avons créées ce qui justifie une fois de plus leur ajout.

Il est également possible d'observer une corrélation de 1 entre les variables Outcome_NewClient et DaysPassed_New_Client. En effet, elles apportent la même information. De ce fait, nous allons supprimer DaysPassed_New_Clien :

```
data_train[,"DaysPassed_New Client"] <- NULL
data_test[,"DaysPassed_New Client"] <- NULL
```

Également ce tableau met en avant des redondances dans le traitement actuel des variables qualitatives par one hot encoding. On observe une forte corrélation, négativement, entre "Marital_married" et "Marital_single". En effet, soit on est marié soit on est seul. Nous pouvons donc supprimer la varible "Marital_married", et non la variable "Mari-

tal_single" puisqu'on peut être seul et divorcé, ou seul et sans jamais avoir été marié.

Ainsi un one hot encoding sur les variables "Marital_single" et "Marital_divorced" apporte toute l'information contenue actuellement dans les colonnes "Marital_single","Marital_married" et "Marital_divorced".

```
data_train[,"Marital_married"] <- NULL
data_test[,"Marital_married"] <- NULL
```

Suivant la même démarche, nous pouvons supprimer "Education_tertiary"

```
data_train[,"Education_tertiary"] <- NULL
data_test[,"Education_tertiary"] <- NULL
```

Pour l'instant, nous ne ferrons pas d'autre suppression de variables, mais gardons ce tableau en tête pour la suite de ce projet.

7 - Feature scaling

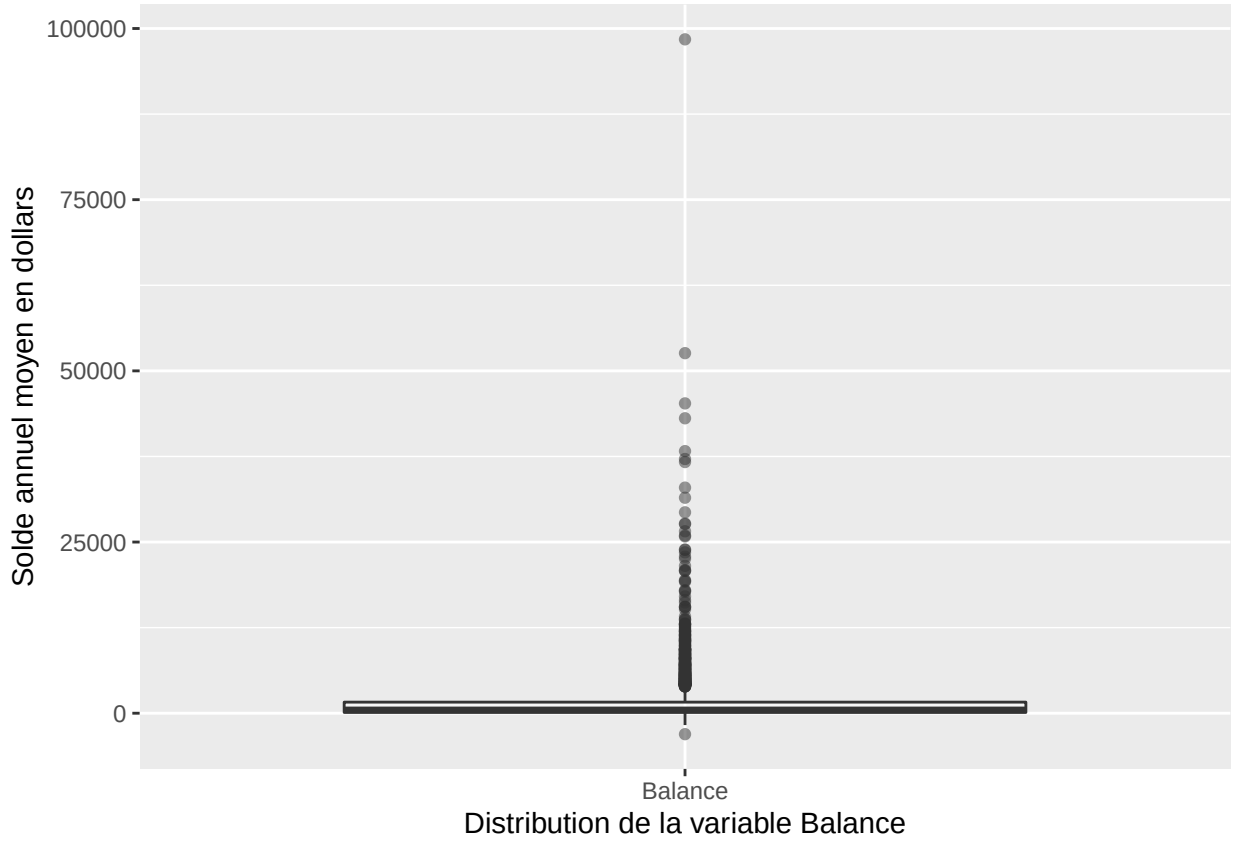
On remarque que la variable explicative Balance prend des valeurs trop "grandes" relativement à d'autres (comme Age ou CallDuration), nous sommes confrontés à des problèmes d'échelle. En effet, cette dernière risque de ralentir la convergence des algorithmes d'apprentissage que nous utiliserons par la suite et donc de les rendre moins performant. Pour éviter ce problème, il faut effectuer au préalable sur cette variable une "renormalisation".

Il existe plusieurs techniques de renormalisation : Min-max normalization, Mean normalization, Standardization, ... Afin d'en choisir une analysons là de plus près :

```
summary(data_train$Balance)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -3058.0   111.0   551.5  1532.9  1619.0 98417.0
```

```
ggplot(data_train) +
  aes(x = "Balance", y = Balance) +
  geom_boxplot(alpha=0.5) +
  xlab("Distribution de la variable Balance") +
  ylab("Solde annuel moyen en dollars")
```



Ce graphique en boite à moustache est très pertinent dans la mesure où il nous indique la présence d'un outlier (Balance = 98 417) en plus de nous donner un bref aperçu de la distribution de cette variable. Avant toute renormalisation, il nous faut donc traiter cet outlier. Observons si on croise une valeur similaire dans les données à prédire :

```
summary(data_test$Balance)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1980.0   114.8   517.5  1398.3  1609.8 41630.0
```

On se rend compte grâce à la commande summary que ce n'est pas le cas. Ainsi, nous allons tout simplement supprimer cet outlier puisqu'il ne peut que biaiser notre apprentissage.

```
ind_outlier = which(data_train$Balance > 75000)
data_train = data_train[-ind_outlier,]
y_train = y_train[-ind_outlier,]
```

Au vu du boxplot précédent et après ce traitement de l'outlier, nous allons appliquer une renormalisation de type "Standardization" à notre variable "Balance" :

```
data_train$Balance = (data_train$Balance - mean(data_train$Balance))/sd(data_train$Balance)

data_test$Balance = (data_test$Balance - mean(data_test$Balance))/sd(data_test$Balance)
```

Nous considérons désormais notre jeu de données prêt pour la modélisation.

8 - Modélisation

Au vu du nombre de variables (important) ayant un fort pouvoir explicatif sur le modèle, nous n'allons pas faire de modélisation du type SVM et KNN.

8.1 - Régression logistique

Commençons par une modélisation utilisant toutes les variables explicatives à notre disposition.

```
regression_log = glm(CarInsurance ~ . -Id, data_train, family = 'binomial')
summary(regression_log)
```

```
##
## Call:
## glm(formula = CarInsurance ~ . - Id, family = "binomial", data = data_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.3993  -0.6136  -0.3292   0.5792   2.5313
##
## Coefficients: (8 not defined because of singularities)
##                                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)                          -2.480795   0.458201  -5.414 6.16e-08
## Age                                   0.003402   0.005117   0.665 0.506111
## Default1                             -0.417920   0.410010  -1.019 0.308065
## Balance                               0.003403   0.043843   0.078 0.938123
## HHInsurance1                         -1.012902   0.096135 -10.536  < 2e-16
## CarLoan1                             -0.688590   0.144909  -4.752 2.02e-06
## CallDuration                          0.332805   0.012335  26.981  < 2e-16
## Job_admin.1                          -0.037710   0.267936  -0.141 0.888073
## Job_blue_collar1                     -0.652899   0.263047  -2.482 0.013062
## Job_entrepreneur1                    -0.814193   0.357446  -2.278 0.022738
## Job_housemaid1                       -0.353073   0.361264  -0.977 0.328406
## Job_management1                      -0.468764   0.266719  -1.758 0.078829
## Job_retired1                          0.396783   0.301477   1.316 0.188131
## Job_self_employed1                   -0.641869   0.333681  -1.924 0.054405
## Job_services1                        -0.361324   0.283885  -1.273 0.203096
## Job_student1                          0.673403   0.329214   2.045 0.040807
## Job_technician1                      -0.257135   0.259539  -0.991 0.321814
## Job_unemployed1                            NA         NA      NA       NA
## Marital_divorced1                     0.168363   0.138041   1.220 0.222592
## Marital_single1                       0.335430   0.111167   3.017 0.002550
## Education_primary1                   -0.622282   0.181946  -3.420 0.000626
## Education_secondary1                 -0.491026   0.125310  -3.918 8.91e-05
## NoOfContacts_1_Contact1               0.939128   0.264953   3.545 0.000393
## NoOfContacts_2_Contact1               0.531986   0.269581   1.973 0.048452
## NoOfContacts_3_Contact1               0.698247   0.282684   2.470 0.013509
## NoOfContacts_4_7Contact1              0.286979   0.285811   1.004 0.315338
## NoOfContacts_8_Contact1                    NA         NA      NA       NA
## DaysPassed_Between_4_month_and_1_year1  2.827070   0.268230  10.540  < 2e-16
## DaysPassed_Less_than_4_month1           3.418884   0.276538  12.363  < 2e-16
## DaysPassed_More_than_a_year1            4.437274   0.402202  11.032  < 2e-16
## DaysPassed_New_Client1                     NA         NA      NA       NA
```

```
## PrevAttempts_0_PrevC1                       NA         NA      NA       NA
## PrevAttempts_1_PrevC1                  -0.345092   0.237050  -1.456 0.145454
## PrevAttempts_2_3PrevC1                 -0.173420   0.232275  -0.747 0.455295
## PrevAttempts_4_PrevC1                       NA         NA      NA       NA
## Outcome_failure1                       -2.367349   0.235575 -10.049  < 2e-16
## Outcome_NewClient1                          NA         NA      NA       NA
## Outcome_other1                         -2.176720   0.266504  -8.168 3.14e-16
## Outcome_success1                            NA         NA      NA       NA
## WhenIsTheCall_91                        0.123319   0.181941   0.678 0.497901
## WhenIsTheCall_101                       0.118952   0.182723   0.651 0.515046
## WhenIsTheCall_111                       0.049969   0.194467   0.257 0.797217
## WhenIsTheCall_121                      -0.018716   0.185642  -0.101 0.919694
## WhenIsTheCall_131                       0.049055   0.184330   0.266 0.790142
## WhenIsTheCall_141                      -0.130105   0.187023  -0.696 0.486641
## WhenIsTheCall_151                       0.169525   0.183950   0.922 0.356745
## WhenIsTheCall_161                       0.430758   0.179464   2.400 0.016384
## WhenIsTheCall_171                           NA         NA      NA       NA
##
## (Intercept)                           ***
## Age
## Default1
## Balance
## HHInsurance1                          ***
## CarLoan1                              ***
## CallDuration                          ***
## Job_admin.1
## Job_blue_collar1                      *
## Job_entrepreneur1                     *
## Job_housemaid1
## Job_management1                       .
## Job_retired1
## Job_self_employed1                    .
## Job_services1
## Job_student1                          *
## Job_technician1
## Job_unemployed1
## Marital_divorced1
## Marital_single1                       **
## Education_primary1                    ***
## Education_secondary1                  ***
## NoOfContacts_1_Contact1               ***
## NoOfContacts_2_Contact1               *
## NoOfContacts_3_Contact1               *
## NoOfContacts_4_7Contact1
## NoOfContacts_8_Contact1
## DaysPassed_Between_4_month_and_1_year1 ***
## DaysPassed_Less_than_4_month1         ***
## DaysPassed_More_than_a_year1          ***
## DaysPassed_New_Client1
## PrevAttempts_0_PrevC1
## PrevAttempts_1_PrevC1
## PrevAttempts_2_3PrevC1
## PrevAttempts_4_PrevC1
## Outcome_failure1                      ***
```

```
## Outcome_NewClient1
## Outcome_other1                          ***
## Outcome_success1
## WhenIsTheCall_91
## WhenIsTheCall_101
## WhenIsTheCall_111
## WhenIsTheCall_121
## WhenIsTheCall_131
## WhenIsTheCall_141
## WhenIsTheCall_151
## WhenIsTheCall_161                       *
## WhenIsTheCall_171
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5386.3  on 3998  degrees of freedom
## Residual deviance: 3334.2  on 3959  degrees of freedom
## AIC: 3414.2
##
## Number of Fisher Scoring iterations: 5
```

Voici quelques indicateurs statistiques :

```
pR2(regression_log)
```

```
## fitting null model for pseudo-r2
```

```
##           llh       llhNull            G2      McFadden          r2ML
## -1667.1239939 -2693.1476201  2052.0472523     0.3809756     0.4013871
##          r2CU
##     0.5424461
```

On regarde ici le pseudo-R de McFadden : 0.3809756, notre modèle n'est pas optimal. Commençons par effectuer une méthode de step by step afin de sélectionner les variables de modélisation :

```
step(regression_log, direction = "forward")
```

```
step(regression_log, direction = "backward")
```

```
step(regression_log, direction = "both")
```

Grâce à la méthode pas à pas, nous pouvons identifier un modèle pertinent. Pour information, nous en avons effectué 3 (both, forward et backward) pour maximiser nos chances de trouver le modèle "optimal". En effet, dans certains cas, il est possible que ces procédures trouvent des modèles légèrement différents. Ici, 2 ont été identifiés. Essayons les deux :

```
reg_retenue_1  = glm(CarInsurance ~ HHInsurance + CarLoan + CallDuration +
    Job_blue_collar + Job_entrepreneur + Job_management + Job_retired +
    Job_self_employed + Job_student + Marital_single + Education_primary +
```

```
    Education_secondary + NoOfContacts_1_Contact + NoOfContacts_2_Contact +
    NoOfContacts_3_Contact + DaysPassed_Between_4_month_and_1_year +
    DaysPassed_Less_than_4_month + DaysPassed_More_than_a_year +
    Outcome_failure + Outcome_other + WhenIsTheCall_14 + WhenIsTheCall_16,
    family='binomial',data = data_train)

print(reg_retenue_1)
```

```
##
## Call:  glm(formula = CarInsurance ~ HHInsurance + CarLoan + CallDuration +
##     Job_blue_collar + Job_entrepreneur + Job_management + Job_retired +
##     Job_self_employed + Job_student + Marital_single + Education_primary +
##     Education_secondary + NoOfContacts_1_Contact + NoOfContacts_2_Contact +
##     NoOfContacts_3_Contact + DaysPassed_Between_4_month_and_1_year +
##     DaysPassed_Less_than_4_month + DaysPassed_More_than_a_year +
##     Outcome_failure + Outcome_other + WhenIsTheCall_14 + WhenIsTheCall_16,
##     family = "binomial", data = data_train)
##
## Coefficients:
##                              (Intercept)                              HHInsurance1
##                                  -2.2021                                   -1.0143
##                                 CarLoan1                             CallDuration
##                                  -0.7291                                    0.3328
##                           Job_blue_collar1                       Job_entrepreneur1
##                                  -0.4759                                   -0.6192
##                           Job_management1                           Job_retired1
##                                  -0.2795                                    0.6728
##                         Job_self_employed1                           Job_student1
##                                  -0.4466                                    0.8348
##                           Marital_single1                       Education_primary1
##                                   0.2711                                   -0.6302
##                       Education_secondary1                  NoOfContacts_1_Contact1
##                                  -0.4944                                    0.7176
##                  NoOfContacts_2_Contact1                  NoOfContacts_3_Contact1
##                                   0.3047                                    0.4864
## DaysPassed_Between_4_month_and_1_year1           DaysPassed_Less_than_4_month1
##                                   2.6615                                    3.2661
##             DaysPassed_More_than_a_year1                        Outcome_failure1
##                                   4.2334                                   -2.3852
##                             Outcome_other1                       WhenIsTheCall_141
##                                  -2.1605                                   -0.2057
##                         WhenIsTheCall_161
##                                   0.3513
##
## Degrees of Freedom: 3998 Total (i.e. Null);  3976 Residual
## Null Deviance:       5386
## Residual Deviance: 3347  AIC: 3393
```

```
reg_retenue_2  = glm(CarInsurance ~ HHInsurance + CarLoan + CallDuration +
    Job_blue_collar + Job_entrepreneur + Job_management + Job_retired +
    Job_self_employed + Job_student + Marital_single + Education_primary +
    Education_secondary + NoOfContacts_1_Contact + NoOfContacts_2_Contact +
    NoOfContacts_3_Contact + DaysPassed_Between_4_month_and_1_year +
```

```
    DaysPassed_Less_than_4_month + DaysPassed_More_than_a_year +
    Outcome_failure + Outcome_other + WhenIsTheCall_14 + WhenIsTheCall_16
    + Job_admin. , family='binomial',data = data_train)

print(reg_retenue_2)
```

```
##
## Call:  glm(formula = CarInsurance ~ HHInsurance + CarLoan + CallDuration +
##     Job_blue_collar + Job_entrepreneur + Job_management + Job_retired +
##     Job_self_employed + Job_student + Marital_single + Education_primary +
##     Education_secondary + NoOfContacts_1_Contact + NoOfContacts_2_Contact +
##     NoOfContacts_3_Contact + DaysPassed_Between_4_month_and_1_year +
##     DaysPassed_Less_than_4_month + DaysPassed_More_than_a_year +
##     Outcome_failure + Outcome_other + WhenIsTheCall_14 + WhenIsTheCall_16 +
##     Job_admin., family = "binomial", data = data_train)
##
## Coefficients:
##                               (Intercept)                               HHInsurance1
##                                   -2.2532                                    -1.0190
##                                  CarLoan1                               CallDuration
##                                   -0.7333                                     0.3331
##                           Job_blue_collar1                        Job_entrepreneur1
##                                   -0.4124                                    -0.5555
##                            Job_management1                            Job_retired1
##                                   -0.2200                                     0.7337
##                         Job_self_employed1                             Job_student1
##                                   -0.3853                                     0.8991
##                            Marital_single1                         Education_primary1
##                                    0.2735                                    -0.6218
##                       Education_secondary1                  NoOfContacts_1_Contact1
##                                   -0.5055                                     0.7047
##                   NoOfContacts_2_Contact1                  NoOfContacts_3_Contact1
##                                    0.3002                                     0.4810
## DaysPassed_Between_4_month_and_1_year1           DaysPassed_Less_than_4_month1
##                                    2.6620                                     3.2661
##              DaysPassed_More_than_a_year1                           Outcome_failure1
##                                    4.2466                                    -2.3871
##                             Outcome_other1                         WhenIsTheCall_141
##                                   -2.1700                                    -0.2088
##                          WhenIsTheCall_161                               Job_admin.1
##                                    0.3563                                     0.2360
##
## Degrees of Freedom: 3998 Total (i.e. Null);  3975 Residual
## Null Deviance:       5386
## Residual Deviance: 3344  AIC: 3392
```

Au vu de l'AIC, nous pouvons admettre que ces modèles sont équivalents (3392 vs. 3393). Par défaut, nous allons conserver le plus petit.

```
summary(reg_retenue_1)
```

```
##
```

```
## Call:
## glm(formula = CarInsurance ~ HHInsurance + CarLoan + CallDuration +
##     Job_blue_collar + Job_entrepreneur + Job_management + Job_retired +
##     Job_self_employed + Job_student + Marital_single + Education_primary +
##     Education_secondary + NoOfContacts_1_Contact + NoOfContacts_2_Contact +
##     NoOfContacts_3_Contact + DaysPassed_Between_4_month_and_1_year +
##     DaysPassed_Less_than_4_month + DaysPassed_More_than_a_year +
##     Outcome_failure + Outcome_other + WhenIsTheCall_14 + WhenIsTheCall_16,
##     family = "binomial", data = data_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.3389  -0.6189  -0.3317   0.5846   2.5507
##
## Coefficients:
##                                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)                             -2.20206    0.17888 -12.310  < 2e-16 ***
## HHInsurance1                            -1.01428    0.09433 -10.752  < 2e-16 ***
## CarLoan1                                -0.72909    0.14245  -5.118 3.08e-07 ***
## CallDuration                             0.33282    0.01228  27.094  < 2e-16 ***
## Job_blue_collar1                        -0.47593    0.13745  -3.463 0.000535 ***
## Job_entrepreneur1                       -0.61918    0.27705  -2.235 0.025425 *
## Job_management1                         -0.27948    0.13947  -2.004 0.045084 *
## Job_retired1                             0.67284    0.18060   3.726 0.000195 ***
## Job_self_employed1                      -0.44661    0.24472  -1.825 0.068004 .
## Job_student1                             0.83481    0.23454   3.559 0.000372 ***
## Marital_single1                          0.27115    0.09876   2.746 0.006042 **
## Education_primary1                      -0.63017    0.17639  -3.573 0.000354 ***
## Education_secondary1                    -0.49439    0.12394  -3.989 6.63e-05 ***
## NoOfContacts_1_Contact1                  0.71763    0.13132   5.465 4.64e-08 ***
## NoOfContacts_2_Contact1                  0.30472    0.14100   2.161 0.030682 *
## NoOfContacts_3_Contact1                  0.48641    0.16507   2.947 0.003213 **
## DaysPassed_Between_4_month_and_1_year1   2.66150    0.22004  12.096  < 2e-16 ***
## DaysPassed_Less_than_4_month1            3.26611    0.23007  14.196  < 2e-16 ***
## DaysPassed_More_than_a_year1             4.23340    0.36249  11.679  < 2e-16 ***
## Outcome_failure1                        -2.38524    0.23443 -10.174  < 2e-16 ***
## Outcome_other1                          -2.16047    0.26550  -8.137 4.04e-16 ***
## WhenIsTheCall_141                       -0.20566    0.14206  -1.448 0.147707
## WhenIsTheCall_161                        0.35133    0.13247   2.652 0.007996 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5386.3  on 3998  degrees of freedom
## Residual deviance: 3346.6  on 3976  degrees of freedom
## AIC: 3392.6
##
## Number of Fisher Scoring iterations: 5
```

Ainsi, nous sommes en mesure de présenter notre première prédiction, puis de la comparer avec celle trouvée sur Internet.

Évaluation sur l'échantillon d'apprentissage :

```r
predtrain_retenue = predict(reg_retenue_1, data_train, type="response")
ind_positiv = which(predtrain_retenue > 0.5)
predtrain_retenue[ind_positiv] = 1
predtrain_retenue[-ind_positiv] = 0
predtrain_retenue = as.factor(predtrain_retenue)
confusionMatrix(data=predtrain_retenue,reference=data_train$CarInsurance)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2112  471
##          1  283 1133
##
##                Accuracy : 0.8115
##                  95% CI : (0.799, 0.8235)
##     No Information Rate : 0.5989
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5998
##
##  Mcnemar's Test P-Value : 9.75e-12
##
##             Sensitivity : 0.8818
##             Specificity : 0.7064
##          Pos Pred Value : 0.8177
##          Neg Pred Value : 0.8001
##              Prevalence : 0.5989
##          Detection Rate : 0.5281
##    Detection Prevalence : 0.6459
##       Balanced Accuracy : 0.7941
##
##        'Positive' Class : 0
##
```

Nous obtenons une accuracy et une sensibilité correcte (0.8 et 0.88 respectivement). On prédit donc bien les clients qui ne souscriront pas à notre assurance. Cependant, notre spécificité est de 0.7 ce qui est bien, mais sans plus, on prédit assez bien les personnes qui souscriront à notre assurance.

Nous obtenons des résultats similaires sur l'échantillon test, ce qui est synonyme d'un bon pouvoir de généralisation de notre modèle :

```r
predtest_retenue = predict(reg_retenue_1, data_test, type="response")
ind_positiv_test = which(predtest_retenue > 0.5)
predtest_retenue[ind_positiv_test] = 1
predtest_retenue[-ind_positiv_test] = 0
predtest_retenue = as.factor(predtest_retenue)
confusionMatrix(data=predtest_retenue,reference=data_test$CarInsurance)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction   0   1
##          0 571  99
##          1  73 257
##
##                Accuracy : 0.828
##                  95% CI : (0.8032, 0.8509)
##     No Information Rate : 0.644
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.6187
##
##  Mcnemar's Test P-Value : 0.05662
##
##             Sensitivity : 0.8866
##             Specificity : 0.7219
##          Pos Pred Value : 0.8522
##          Neg Pred Value : 0.7788
##              Prevalence : 0.6440
##          Detection Rate : 0.5710
##    Detection Prevalence : 0.6700
##       Balanced Accuracy : 0.8043
##
##        'Positive' Class : 0
##
```

Voici la répartition de notre prédiction avec la méthode de régression logistique :

```
table(predtest_retenue)
```

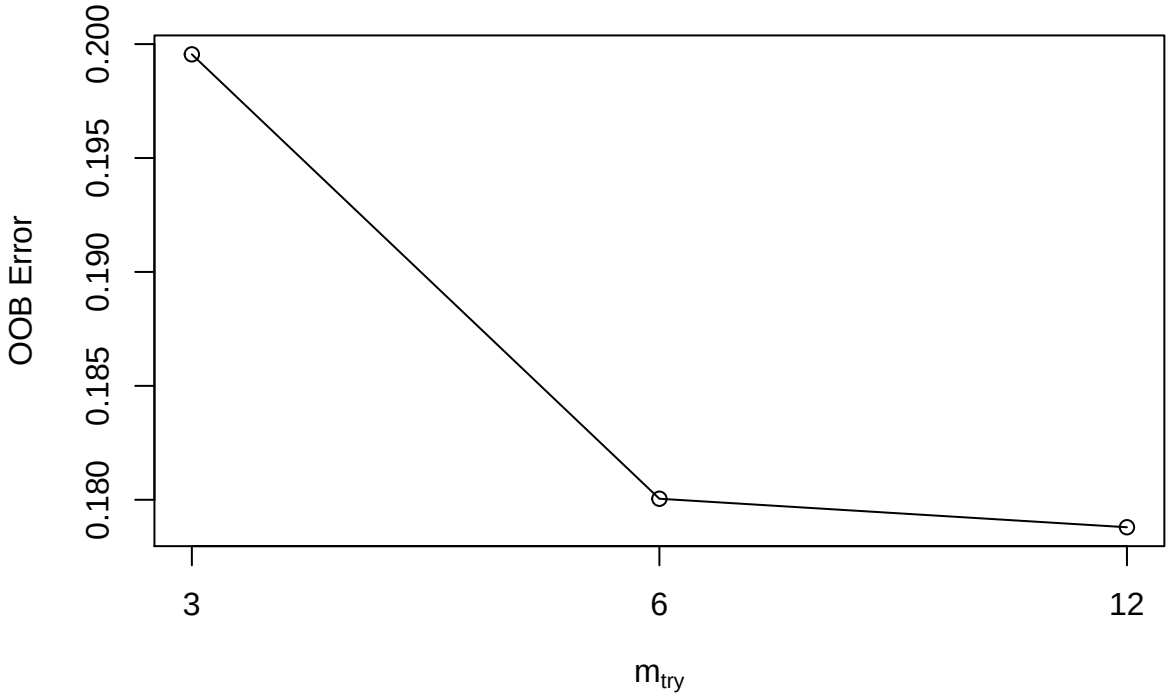
```
## predtest_retenue
##   0   1
## 670 330
```

8.2 - Random Forest

Essayons maintenant les Random Forest. Tout d'abord, nous allons chercher les meilleurs paramètres. On cherche le meilleur "mtry" qui correspond au nombre de régresseur choisi aléatoirement pour la construction des "ntree", arbres qui comprosent la forêt.

```
t = tuneRF(subset(data_train, select = -c( CarInsurance , Id )) ,
           as.factor(data_train$CarInsurance), stepFactor = 0.5, plot = TRUE,
           ntreeTry = 500, trace = FALSE, improve = 0.05)
```

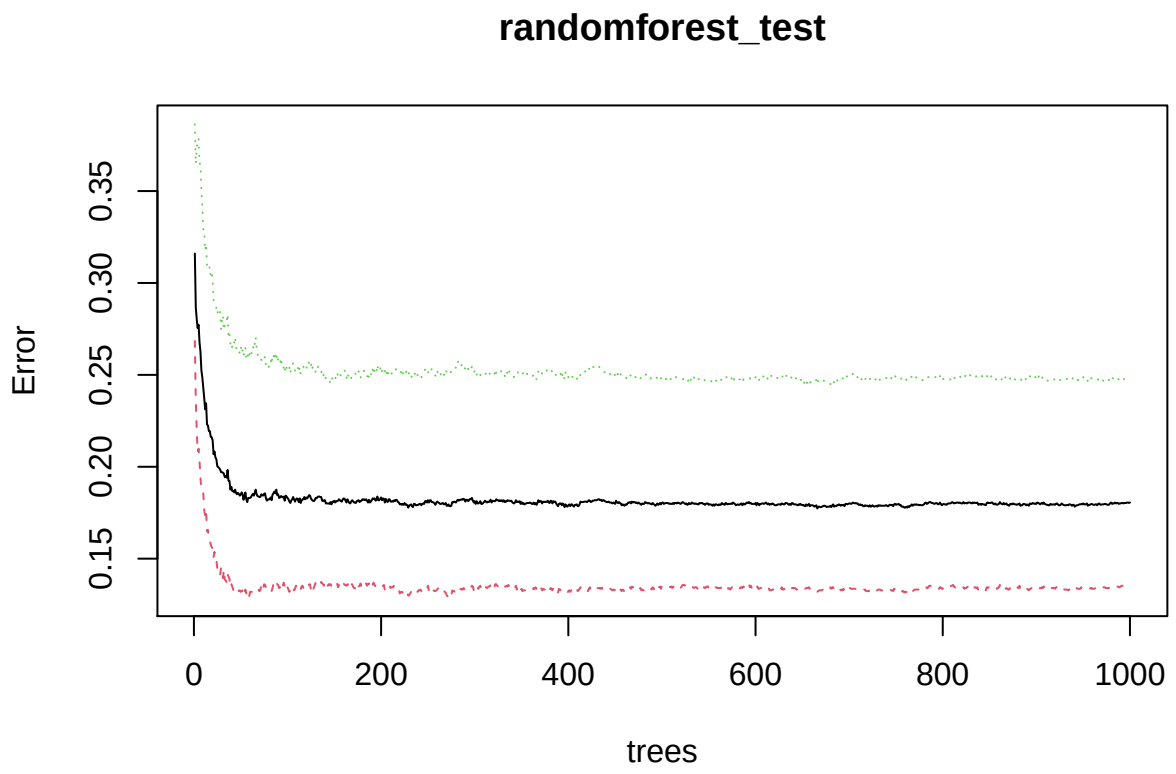
```
## 0.006944444 0.05
## -0.1083333 0.05
```



On remarque que l'erreur est minimale pour un mtry égale à 6.

On cherche ensuite le meilleur "ntree" :

```
randomforest_test = randomForest(CarInsurance~.-Id , data_train , ntree = 1000, mtry = 6)
plot(randomforest_test)
```



On remarque qu'à partir de 200 arbres l'erreur ne diminue pas significativement. Ainsi, nous choisirons le modèle randomForest suivant :

```
randomforest = randomForest(CarInsurance~.-Id, data_train , ntree = 200, mtry = 6)
print(randomforest)
```

```
##
## Call:
##  randomForest(formula = CarInsurance ~ . - Id, data = data_train,      ntree = 200, mtry = 6)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 18.25%
## Confusion matrix:
##      0    1 class.error
## 0 2056  339   0.1415449
## 1  391 1213   0.2437656
```

On remarque que l'erreur sur la classe "Client souscrivant à l'assurance automobile" a une erreur importante, on se trompe environ une fois sur 4.

On va ensuite prédire la réponse de nos clients test :

```r
p1 = predict(randomforest, data_test)
table(p1)
```

```
## p1
##   0   1
## 615 385
```

On compare à la base de données prédite trouvée sur kaggle :

```r
confusionMatrix(p1, as.factor(y_test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 531  84
##          1 113 272
##
##                Accuracy : 0.803
##                  95% CI : (0.777, 0.8272)
##     No Information Rate : 0.644
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.5781
##
##  Mcnemar's Test P-Value : 0.04605
##
##             Sensitivity : 0.8245
##             Specificity : 0.7640
##          Pos Pred Value : 0.8634
##          Neg Pred Value : 0.7065
##              Prevalence : 0.6440
##          Detection Rate : 0.5310
##    Detection Prevalence : 0.6150
##       Balanced Accuracy : 0.7943
##
##        'Positive' Class : 0
##
```

Nous avons une accuracy de 80% et une sensitivité de 83% ce qui est légèrement moins bon que la régression logistique. En revanche, nous augmentons la spécificité avec ce modèle : elle est maintenant de 75% (contre 72% précédemment). Nous faisons donc mieux pour prédire les personnes voulant souscrire à une assurance automobile.

Il est donc intéressant de noter que notre choix de modèle entre Random Forest/ Régression logistique dépendra du destinataire de l'étude. Selon s'il accorde plus d'importance au taux de vrais positifs ou au taux de vrais négatifs.

8.3 - Réseaux de neurones

Essayons maintenant d'implémenter un réseau de neurones. Après plusieurs recherches nous avons trouvé que la librairie Keras, connue pour la création et l'entrainement de réseaux de neurones sous Python, est aussi disponible sous R. Notez donc que pour cette partie python doit être installé sur votre ordinateur. Également, afin de reperformer les résultats obtenus dans la suite de cette partie il sera nécessaire de :

1) Installer la dernière version de Rtools via cran-r, et la liée au chemin (cf instruction du cran).

2) Installer reticulate via la commande install.packages de R.

3) Créer un environnement puis installer tensorflow.

4) Finaliser l'installation avec la commande install_tensorflow().

5) Installer Keras et Tensorflow via la commande install.packages de R.

```
# À décommenter si besoins


#install.packages("reticulate")
library(reticulate)
# create a new environment
conda_create("r-reticulate")
```

```
## [1] "/Users/valentinmaillo/Library/r-miniconda/envs/r-reticulate/bin/python"
```

```
# install tensorflow
conda_install("r-reticulate", "tensorflow")
#install_tensorflow()

#install.packages("keras")
#install.packages("tensorflow")

library("keras")
```

```
##
## Attaching package: 'keras'
```

```
## The following object is masked from 'package:party':
##
##     fit
```

```
## The following object is masked from 'package:modeltools':
##
##     fit
```

```
library("tensorflow")
```

```
##
## Attaching package: 'tensorflow'
```

```
## The following object is masked from 'package:caret':
##
##     train
```

Tout d'abord, construisons une matrice, X, correspondant à l'ensemble de nos observations :

Construisons à présent notre réseau de neurones. Tout d'abord, notons que notre échantillon d'apprentissage comporte 47 variables explicatives. La première couche sera donc formée de 47 neurones.

Nous pouvons également de suite affirmer que la dernière couche du réseau ne sera composé que d'un seul neuronne puisque nous faisons ici de la classification binaire. Classiquement, l'essentiel du problème sera de déterminer le nombre de couches cachées et le nombre de neurones les composants.

Commençons par déterminer le nombre de couches adapté à notre modèle (avec une structure pyramidale assez basique pour l'instant en terme de nombre de neurones du type : 47 70 70 20 1). Nous testerons un réseau à 1, 2, 3, 5 et 7 couches cachées. Pour le moment essayer avec un plus grand nombre de couches cachées semble inutile au vu de notre problématique. Si néanmoins, c'est le cas les premières analyses présentées ci-dessous nous l'indiqueront et nous ferons d'autres tests en ce sens.

Essayer avec moins d'une couche cachée sortirait du cadre de cette modélisation réseaux neuronaux, cela nous renvoit tout simplement à un modèle linéaire.

Commençons avec le réseau à une couche cachée :

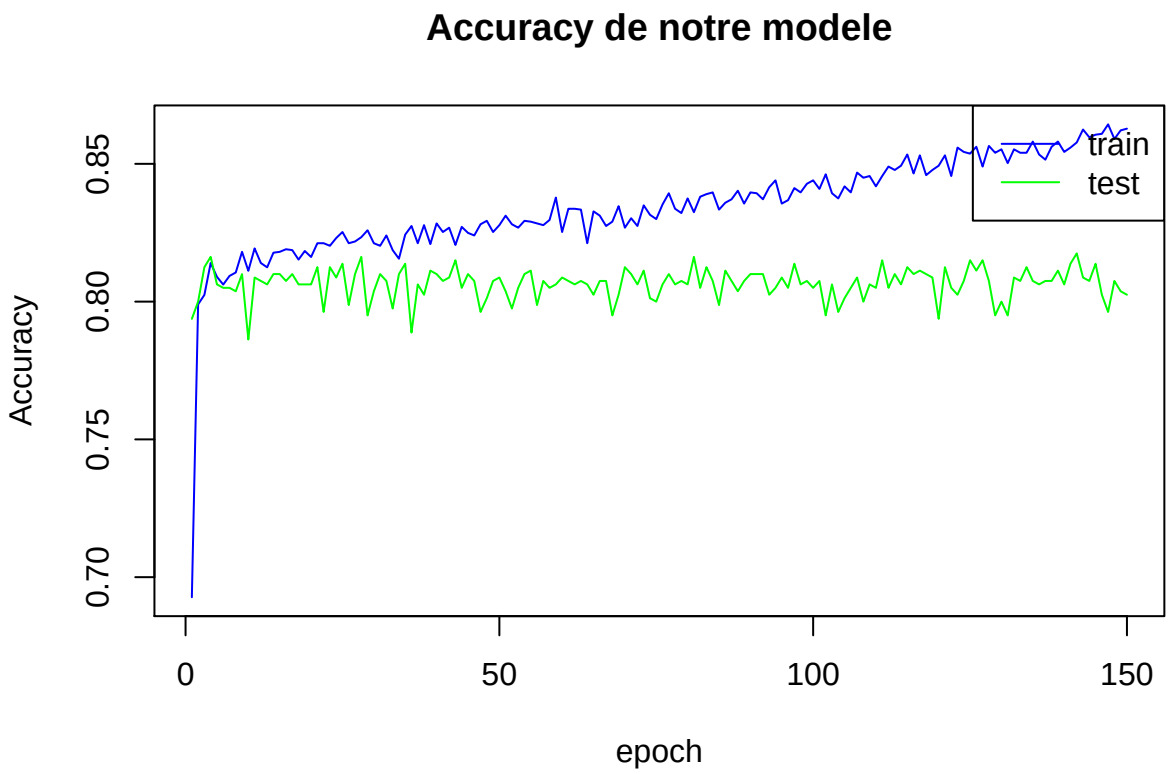
```
#Notez que le choix des paramètres utilisés ici est documenté après ce bloc de code
nb_epochs = 150
choice_batch_size = 30
model_1 <- keras_model_sequential()
```

```
## Loaded Tensorflow version 2.6.2
```

```
model_1 %>%
   layer_dense(units = 70, activation = 'relu', input_shape = ncol(X_app))  %>%
   layer_dense(units = 1,activation = 'sigmoid')

#Choix de la fonction de perte, de l'optimiseur et de la métrique
model_1 %>% compile(loss = 'binary_crossentropy',optimizer = "adam",metrics = c('accuracy'))
```

```
#On entraîne le réseau sur les données d'apprentissage :

my_model = model_1 %>% keras::fit(X_app, y_train, epochs = nb_epochs, batch_size =
                              choice_batch_size, validation_split = 0.2)
```

```
# Plot the model loss of the training data
plot(my_model$metrics$accuracy, main="Accuracy de notre modele", xlab = "epoch", ylab="Accuracy", col="

# Plot the model loss of the test data
lines(my_model$metrics$val_accuracy, col="green")

# Add legend
legend("topright", c("train","test"), col=c("blue", "green"), lty=c(1,1))
```



Notons que loss et accuracy indiquent respectivement la perte et l'accuracy du modèle sur les données d'apprentissage. val_loss et val_acc sont les mêmes métriques, mais sur l'échantillon test construit au sein de notre échantillon d'apprentissage.

Profitons de cette première modélisation pour exposé le choix de nos paramètres. Nous avons fixé le nombre d'epochs à 150, c'est-à-dire que notre réseau va parcourir 150 fois X_app durant sa phase d'apprentissage avec un batch_size de 30. Pour rappel, le batch_size désigne le nombre d'individus utilisé par le réseau pour mettre à jour ses poids au cours d'une itération. Classiquement, pour ce genre de problème, il est fixé à 30, ce que nous avons fait donc.

En outre, suivant la répartition de la target présentée par le dernier graphique de la partie 4, nous avons choisi l'accuracy comme métrique d'évaluation. Étant dans le cas d'une classification binaire notre fonction de perte sera la "binary_crossentropy" suivant les préconisations de la documentation Keras. Le choix de l'algorithme adam se justifie par l'efficience de ce dernier par rapport à une simple descente de gradient par rétro-propagation par exemple.

Concernant les fonctions d'activation du réseau, il faut distinguer 2 cas :

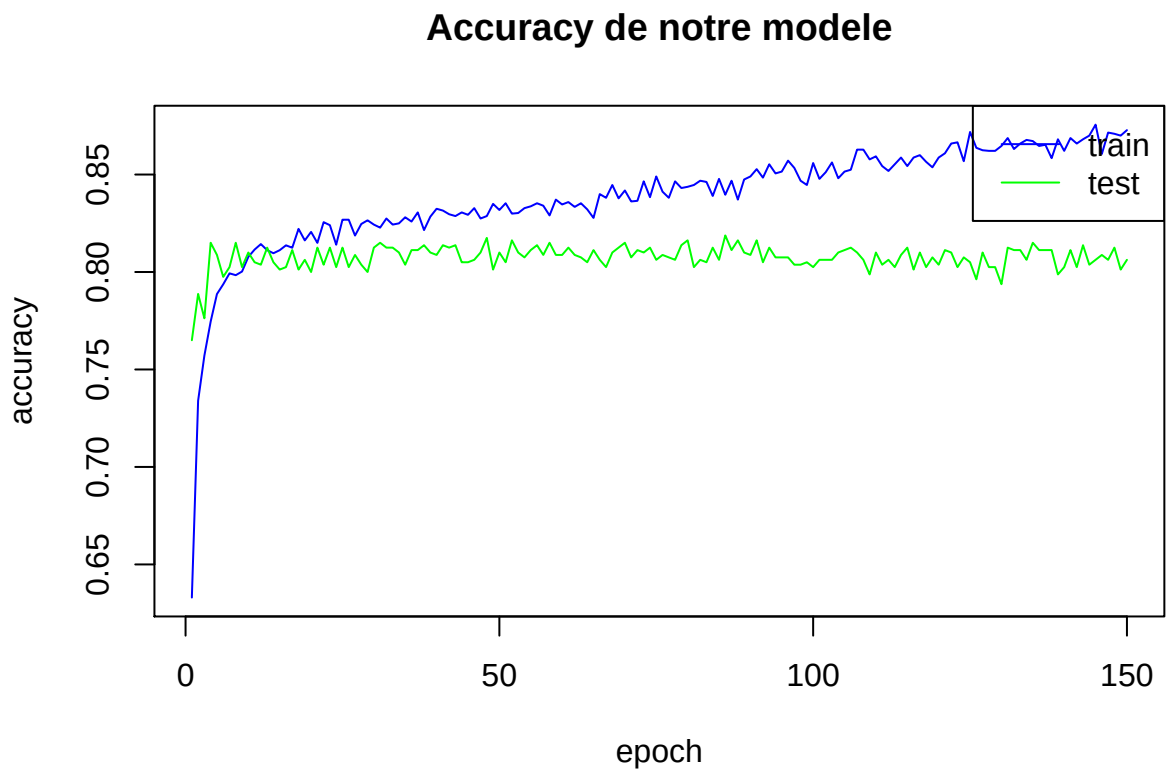
- `ReLu sera choisie pour les couches cachées, en effet, cette fonction d'activation est aujourd'hui t`

- `Pour la dernière couche de notre réseau, étant dans le cadre d'une classification binaire, la fonct`

Passons donc à un réseau de neurones à 2 couches cachées et regardons si cela permet d'augmenter notre accuracy (afin d'éviter le sur-apprentissage, nous insérerons désormais du dropout entre chaque couche cachée) :

```r
model_2 <- keras_model_sequential()


model_2 %>%
   layer_dense(units = 70, activation = 'relu', input_shape = ncol(X_app)) %>%
   layer_dropout(rate=0.4)%>%
   layer_dense(units = 70, activation = 'relu') %>%
   layer_dense(units = 1,activation = 'sigmoid')

model_2 %>% compile(loss = 'binary_crossentropy',optimizer = "adam",metrics =
                      c('accuracy'))

my_model_2 = model_2 %>% keras::fit(X_app, y_train, epochs = nb_epochs,
                                    batch_size = choice_batch_size,
                                    validation_split = 0.2)
```

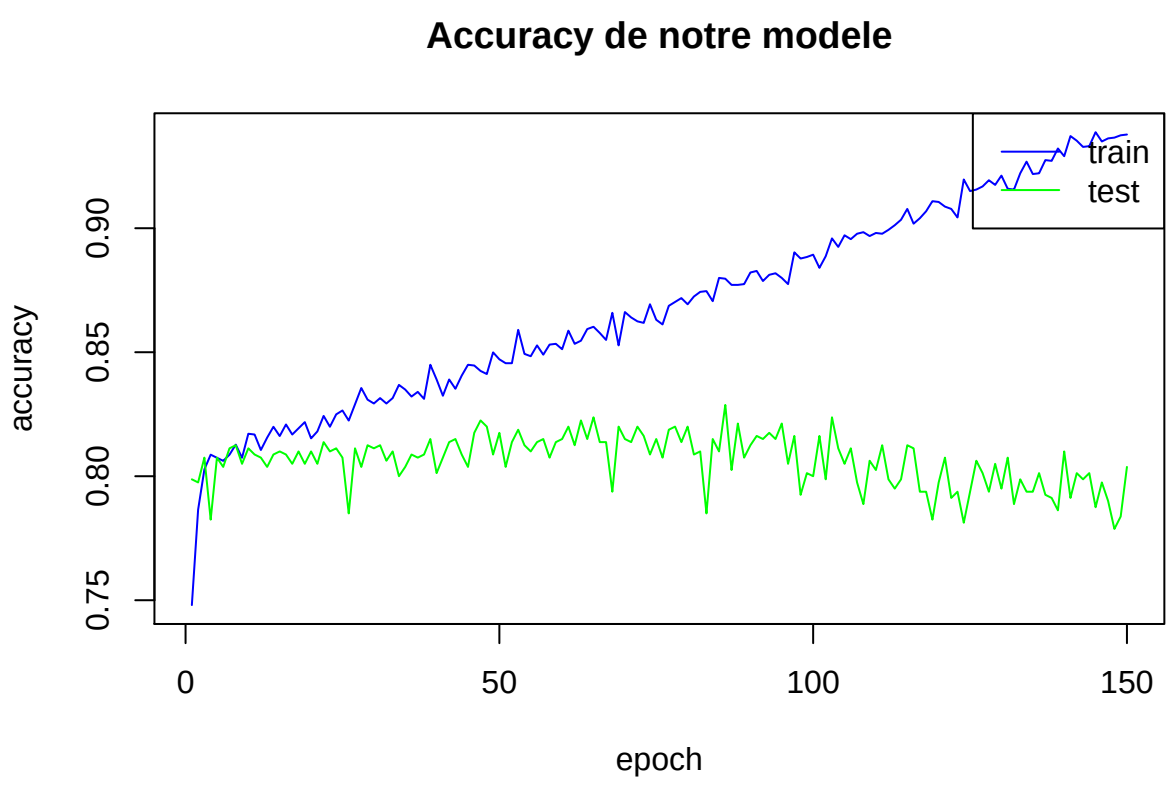
```r
plot(my_model_2$metrics$accuracy, main="Accuracy de notre modele", xlab = "epoch", ylab="accuracy", col=

lines(my_model_2$metrics$val_accuracy, col="green")
legend("topright", c("train","test"), col=c("blue", "green"), lty=c(1,1))
```



Nous pouvons voir que nous n'améliorons pas significativement notre accuracy sur l'échantillon de test. Ainsi, nous allons garder un modèle à deux couches pour la suite. Essayons par curiosité d'enlever le dropout et étudions l'impact de cela :

```r
model_3 <- keras_model_sequential()


model_3 %>%
   layer_dense(units = 70, activation = 'relu', input_shape = ncol(X_app)) %>%
   layer_dense(units = 70, activation = 'relu') %>%
   layer_dense(units = 1,activation = 'sigmoid')

model_3 %>% compile(loss = 'binary_crossentropy',optimizer = "adam",metrics =
                      c('accuracy'))

my_model_3 = model_3 %>% keras::fit(X_app, y_train, epochs = nb_epochs,
                                    batch_size = choice_batch_size,
                                    validation_split = 0.2)
```

```r
plot(my_model_3$metrics$accuracy, main="Accuracy de notre modele", xlab = "epoch", ylab="accuracy", col=

lines(my_model_3$metrics$val_accuracy, col="green")
legend("topright", c("train","test"), col=c("blue", "green"), lty=c(1,1))
```

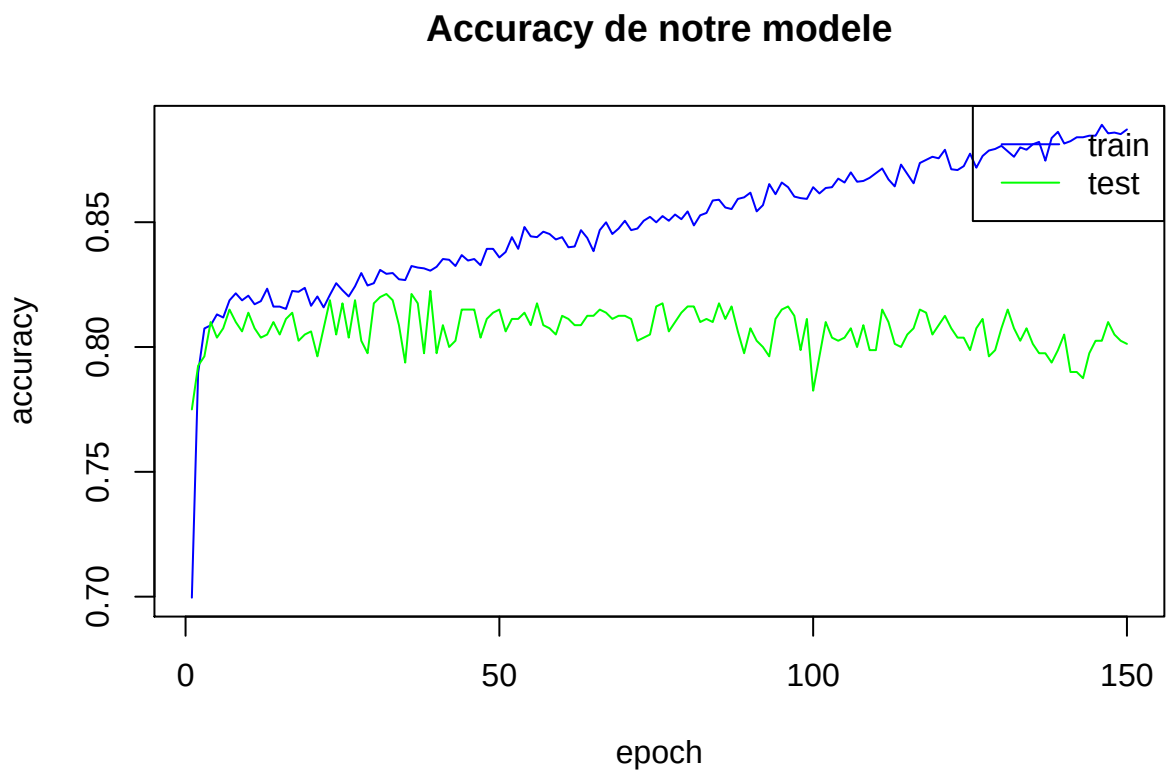


Nous n'avons pas détérioré notre accuracy sur l'échantillon test mais l'avons significativement améliorée sur l'échantillon d'apprentissage. Ainsi, pour un modèle à 2 couches cachées, dans notre cas, nous ne sommes pas sujet au sur-apprentissage. Inutile donc ici de faire du dropout.

Testons maintenant plusieurs structures de réseaux neuronaux en terme de nombre de neurones par couche. Tout d'abord, que se passe-t-il si nous diminuons le nombre de neurones par couche ?

```r
model_4 <- keras_model_sequential()


model_4 %>%
   layer_dense(units = 50, activation = 'relu', input_shape = ncol(X_app)) %>%
   layer_dense(units = 30, activation = 'relu') %>%
   layer_dense(units = 1,activation = 'sigmoid')

model_4 %>% compile(loss = 'binary_crossentropy',optimizer = "adam",metrics =
                      c('accuracy'))

my_model_4 = model_4 %>% keras::fit(X_app, y_train, epochs = nb_epochs,
                                    batch_size = choice_batch_size,
                                    validation_split = 0.2)
```

```r
plot(my_model_4$metrics$accuracy, main="Accuracy de notre modele", xlab = "epoch", ylab="accuracy", col=

lines(my_model_4$metrics$val_accuracy, col="green")
legend("topright", c("train","test"), col=c("blue", "green"), lty=c(1,1))
```



Nous avons donc trouvé notre bon modèle. Faire encore diminuer le nombre de neurones par couche ne serait, par convention, pas pertinent. La règle étant qu'on ne met pas moins de neurones sur la première couche cachée que la dimension de notre matrice en entrée (ici 47). Ainsi, nous allons conserver cette structure. Passons donc maintenant à la prédiction des données X_test :

```r
p1 = model_4 %>% predict(X_test) %>% `>`(0.5)
p1 = as.numeric(p1)
p1 = as.factor(p1)
table(p1)
```

```
## p1
##   0   1
## 633 367
```

```r
confusionMatrix(p1, as.factor(y_test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 540  93
##          1 104 263
##
##                Accuracy : 0.803
##                  95% CI : (0.777, 0.8272)
##     No Information Rate : 0.644
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5733
##
##  Mcnemar's Test P-Value : 0.4762
##
##             Sensitivity : 0.8385
##             Specificity : 0.7388
##          Pos Pred Value : 0.8531
##          Neg Pred Value : 0.7166
##              Prevalence : 0.6440
##          Detection Rate : 0.5400
##    Detection Prevalence : 0.6330
##       Balanced Accuracy : 0.7886
##
##        'Positive' Class : 0
##
```

Remarquons que nous augmentons encore notre spécificité par rapport aux modélisations précédentes. Pour terminer, essayons un modèle Xgboost.

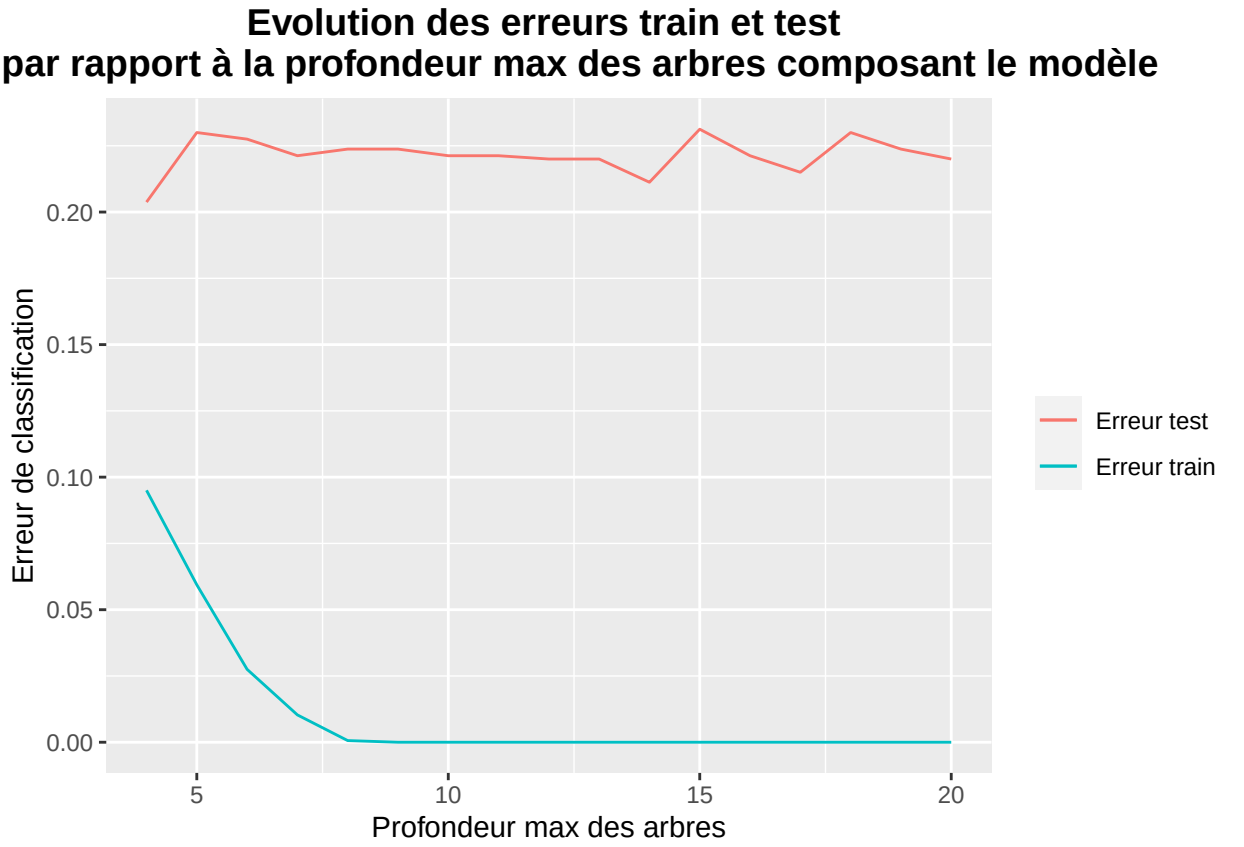
8.4 - Xgboost - Extreme Gradient Boosting

Nous allons commencer par trouver l'hyperparamètre "max.depth", la profondeur maximale des arbres, par validation croisée. Nous fixerons pour l'instant l'autre hyper-paramètre "nrounds" (le nombre d'arbres à entraîner) à 20 pour l'instant.

Pour information, "nthread" etant le nombre de CPU de l'ordinateur alloué à faire tourner ce code, libre à chacun de l'adapter en fonction des caractéristiques de son ordinateur. Pour information, afin de savoir son nombre de CPU : Ctrl+alt+supp -> gestionnaire des taches -> onglet performance -> onglet CPU.

```r
set.seed(101) # afin de pouvoir reproduire les mêmes sorties à chaque fois.

n_row = dim(data_train)[1]
sample <- sample.int(n = n_row, size = floor(0.8*n_row), replace = F)

data_app <- data_train[sample, ]
data_crossV  <- data_train[-sample, ]

y_XGB_app = data_app$CarInsurance
y_XGB_crossV = data_crossV$CarInsurance

data_crossV$CarInsurance <- NULL
data_app$CarInsurance <- NULL
data_test$CarInsurance <- NULL

test_err_list = c()
train_err_list = c()
for (d in 4:20){
   Grad_bost <- xgboost(data = data.matrix(data_app),
                        label = data.matrix(y_XGB_app), max.depth = d,
                        eta = 1, nthread= 4, nrounds = 20,
                        objective = "binary:logistic")

   pred_train <- predict(Grad_bost, data.matrix(data_app))

   err <- mean(as.numeric(pred_train > 0.5) != y_XGB_app)
   train_err_list = c(train_err_list,err)

   pred <- predict(Grad_bost, data.matrix(data_crossV))
   prediction <- as.numeric(pred > 0.5)

   err_test <- mean(prediction != y_XGB_crossV)
   test_err_list = c(test_err_list,err_test)
}
x <- 4:20
df_xgboost <- data.frame(x,test_err_list,train_err_list)
```

```r
ggplot(df_xgboost, aes(x)) +
  geom_line(data = df_xgboost, aes(x = x, y = test_err_list, colour = "Erreur test") )+
  geom_line(data = df_xgboost, aes(x = x, y = train_err_list, colour = "Erreur train") ) +
  ggtitle("Evolution des erreurs train et test
      par rapport à la profondeur max des arbres composant le modèle") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 14),
        legend.title = element_blank() ) +
  xlab("Profondeur max des arbres") +
  ylab("Erreur de classification")
```

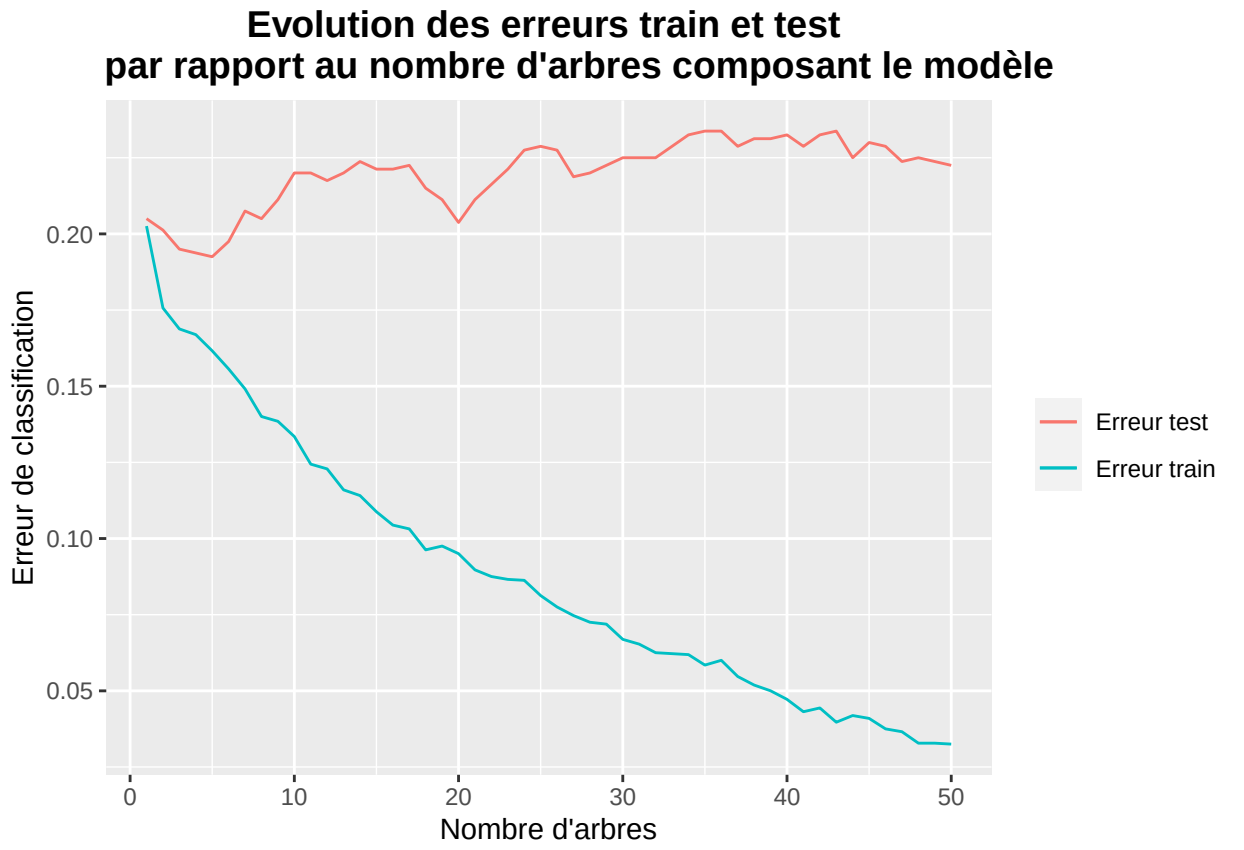



```
#Compromis biais - variance pour max.depth = 4
```

Ainsi, afin d'éviter le sur-apprentissage, nous allons fixer la profondeur maximale des arbres à 4. Cherchons maintenant "nrounds" (le nombre d'arbres à entraîner) minimisant le compromis biais variance par validation croisée.

```
test_err_list = c()
train_err_list = c()
for (i in 1:50){
   Grad_bost <- xgboost(data = data.matrix(data_app),
                        label = data.matrix(y_XGB_app), max.depth = 4,
                        eta = 1, nthread= 4, nrounds = i,
                        objective = "binary:logistic")

   pred_train <- predict(Grad_bost, data.matrix(data_app))

   err <- mean(as.numeric(pred_train > 0.5) != y_XGB_app)
   train_err_list = c(train_err_list,err)

   pred <- predict(Grad_bost, data.matrix(data_crossV))
   prediction <- as.numeric(pred > 0.5)

   err_test <- mean(prediction != y_XGB_crossV)
   test_err_list = c(test_err_list,err_test)
}
x <- 1:50
df_xgboost <- data.frame(x,test_err_list,train_err_list)
```

```
ggplot(df_xgboost, aes(x)) +
  geom_line(data = df_xgboost, aes(x = x, y = test_err_list, colour = "Erreur test") )+
  geom_line(data = df_xgboost, aes(x = x, y = train_err_list, colour = "Erreur train") ) +
  ggtitle("Evolution des erreurs train et test
      par rapport au nombre d'arbres composant le modèle") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 14),
        legend.title = element_blank() ) +
  xlab("Nombre d'arbres") +
  ylab("Erreur de classification")
```




```
#Compromis biais - variance pour nrounds = 19
```

Le compromis biais variance est donc trouvé pour nrounds = 19. Au delà nous sur-apprenons, en dessous nous sous-apprenons. Ainsi notre prédiction par gradient boosting est :

```
data_train$CarInsurance <- NULL
Grad_bost <- xgboost(data = data.matrix(data_train),
                     label = data.matrix(y_train), max.depth = 4,
                     eta = 1, nthread= 4, nrounds = 19,
                     objective = "binary:logistic")
```

```
## [21:20:03] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evalu
## [1]  train-logloss:0.447275
## [2]  train-logloss:0.404776
```

```
## [3]  train-logloss:0.384030
## [4]  train-logloss:0.368787
## [5]  train-logloss:0.357851
## [6]  train-logloss:0.349876
## [7]  train-logloss:0.340735
## [8]  train-logloss:0.331899
## [9]  train-logloss:0.326840
## [10] train-logloss:0.317920
## [11] train-logloss:0.313452
## [12] train-logloss:0.307854
## [13] train-logloss:0.303738
## [14] train-logloss:0.294997
## [15] train-logloss:0.289120
## [16] train-logloss:0.282661
## [17] train-logloss:0.278486
## [18] train-logloss:0.274613
## [19] train-logloss:0.272933
```

```
pred_train <- predict(Grad_bost, data.matrix(data_train))
train_err <- mean(as.numeric(pred_train > 0.5) != y_train)

print(paste("train-error=",train_err))
```

```
## [1] "train-error= 0.117029257314329"
```

```
pred <- predict(Grad_bost, data.matrix(data_test))
prediction <- as.numeric(pred > 0.5)
err_test <- mean(prediction != y_test)

print(paste("test-error=", err_test))
```

```
## [1] "test-error= 0.206"
```

Nous allons maintenant afficher la matrice de confusion afin de comparer facilement cette modélisation au précédente :

```
p1 = Grad_bost %>% predict(data.matrix(data_test)) %>% `>`(0.5)
p1 = as.numeric(p1)
p1 = as.factor(p1)
table(p1)
```

```
## p1
##   0   1
## 688 312
```

```
confusionMatrix(p1, as.factor(y_test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 563 125
```

```
##          1  81 231
##
##                Accuracy : 0.794
##                  95% CI : (0.7676, 0.8187)
##     No Information Rate : 0.644
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.538
##
##  Mcnemar's Test P-Value : 0.002736
##
##             Sensitivity : 0.8742
##             Specificity : 0.6489
##          Pos Pred Value : 0.8183
##          Neg Pred Value : 0.7404
##              Prevalence : 0.6440
##          Detection Rate : 0.5630
##    Detection Prevalence : 0.6880
##       Balanced Accuracy : 0.7616
##
##        'Positive' Class : 0
##
```

Nous obtenons donc une nouvelle fois de bons résultats avec cette nouvelle modélisation. Comparons maintenant toutes celles effectuées au sein de cette section "Modélisation".

9 - Comparaison des modèles et pr''diction du jeu de données test

Pour conclure, voici le tableau de performance de nos différents modèles selon 3 métriques d'évaluation, en notant bien que la classe positive est 0 pour chaque modélisation :

```
result_mod <- t(matrix(c(0.8,0.83,0.76,0.83,0.89,0.72,0.8,0.82,
                         0.76,0.79,0.87,0.65),nrow=3))
result_mod <- as.data.frame(result_mod)
colnames(result_mod) <- c("Accuracy", "Sensibilité", "Spécificité")
row.names(result_mod) <- c("Random Forest", "Régression Logistique", "Réseau de neurones", "Xgboost")
print.data.frame(result_mod)
```

```
##                       Accuracy Sensibilité Spécificité
## Random Forest             0.80        0.83        0.76
## Régression Logistique     0.83        0.89        0.72
## Réseau de neurones        0.80        0.82        0.76
## Xgboost                   0.79        0.87        0.65
```

Ainsi, si nous souhaitons un modèle avec la meilleure accuracy possible nous choisirons la Régression Logistique. Si nous souhaitons avoir un modèle prédisant bien les clients qui ne souscriront pas à notre assurance, nous choisirons une nouvelle fois la régression logistique. Et si nous souhaitons garder le modèle prédisant le mieux les clients qui souscriront à notre assurance, nous garderons le réseau de neurones.

Ceci achève ce projet de classification binaire, nous vous remercions pour votre lecture.