Détecter des faux billets

CME

I- Contexte

Vous êtes consultant Data Analyst dans une entreprise spécialisée dans la data. Votre entreprise a décroché une prestation en régie au sein de l'**Organisation nationale de lutte contre le faux-monnayage (ONCFM)**.

Cette institution a pour objectif de mettre en place des méthodes d'identification des contrefaçons des billets en euros. Ils font donc appel à vous, spécialiste de la data, pour mettre en place une modélisation qui serait capable d'identifier automatiquement les vrais des faux billets. Et ce à partir simplement de certaines dimensions du billet et des éléments qui le composent.

Voici le cahier des charges de l'ONCFM ainsi que le jeu de données

Le client souhaite que vous travailliez directement depuis ses locaux sous la responsabilité de Marie, responsable du projet d'analyse de données à l'ONCFM. Elle vous laissera une grande autonomie pendant votre mission, et vous demande simplement que vous lui présentiez vos résultats une fois la mission terminée. Elle souhaite voir quels sont les traitements et analyses que vous avez réalisés en amont, les différentes pistes explorées pour la construction de l'algorithme, ainsi que le modèle final retenu.

Après avoir lu en détail le cahier des charges, vous vous préparez à vous rendre à l'ONCFM pour prendre vos nouvelles fonctions. Vous notez tout de même un post-it qui se trouve sur le coin de votre bureau, laissé par un de vos collègues :

II- Importation des fichiers

Définir le miroir CRAN pour l'installation de packages R

```
options(repos = c(CRAN = "https://cran.rstudio.com/"))
```

Lecture du fichier csv

```
data <- read.csv("data_raw/billets.csv", sep =";")
```

III- Résumé des datas

```
summary(data)
```

```
  is_genuine           diagonal      height_left     height_right
 Length:1500        Min.   :171.0   Min.   :103.1   Min.   :102.8
 Class :character   1st Qu.:171.8   1st Qu.:103.8   1st Qu.:103.7
 Mode  :character   Median :172.0   Median :104.0   Median :103.9
                    Mean   :172.0   Mean   :104.0   Mean   :103.9
                    3rd Qu.:172.2   3rd Qu.:104.2   3rd Qu.:104.2
                    Max.   :173.0   Max.   :104.9   Max.   :105.0

   margin_low      margin_up         length
 Min.   :2.980   Min.   :2.270   Min.   :109.5
 1st Qu.:4.015   1st Qu.:2.990   1st Qu.:112.0
 Median :4.310   Median :3.140   Median :113.0
 Mean   :4.486   Mean   :3.151   Mean   :112.7
 3rd Qu.:4.870   3rd Qu.:3.310   3rd Qu.:113.3
 Max.   :6.900   Max.   :3.910   Max.   :114.4
 NA's   :37
```

Nous avons un dataframe de 7 colonnes et 1 500 lignes 1 colonne de type character 6 colonnes numériques

IV- Description des variables

```
if (!require(skimr)) install.packages("skimr")
```

```
Le chargement a nécessité le package : skimr
```

```
library(skimr)
skim(data)
```

Table 1: Data summary

Name	data
Number of rows	1500
Number of columns	7
_____	
Column type frequency:	
character	1
numeric	6
_____	
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
is_genuine	0	1	4	5	0	2	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
diagonal	0	1.00	171.96	0.31	171.04	171.75	171.96	172.17	173.01	
height_left	0	1.00	104.03	0.30	103.14	103.82	104.04	104.23	104.88	
height_right	0	1.00	103.92	0.33	102.82	103.71	103.92	104.15	104.95	
margin_low	37	0.98	4.49	0.66	2.98	4.02	4.31	4.87	6.90	
margin_up	0	1.00	3.15	0.23	2.27	2.99	3.14	3.31	3.91	
length	0	1.00	112.68	0.87	109.49	112.03	112.96	113.34	114.44	

Nous avons 37 valeurs manquantes dans la colonne margin_low

On affiche les lignes ou “margin_low” est NA

```
missing_margin_low <- data[is.na(data$margin_low), ]
print(missing_margin_low)
```

```
     is_genuine diagonal height_left height_right margin_low margin_up length
73         True   171.94      103.89       103.45         NA      3.25 112.79
100        True   171.93      104.07       104.18         NA      3.14 113.08
152        True   172.07      103.80       104.38         NA      3.02 112.93
198        True   171.45      103.66       103.80         NA      3.62 113.27
```

```
242        True   171.83      104.14      104.06         NA      3.02 112.36
252        True   171.80      103.26      102.82         NA      2.95 113.22
285        True   171.92      103.83      103.76         NA      3.23 113.29
335        True   171.85      103.70      103.96         NA      3.00 113.36
411        True   172.56      103.72      103.51         NA      3.12 112.95
414        True   172.30      103.66      103.50         NA      3.16 112.95
446        True   172.34      104.42      103.22         NA      3.01 112.97
482        True   171.81      103.53      103.96         NA      2.71 113.99
506        True   172.01      103.97      104.05         NA      2.98 113.65
612        True   171.80      103.68      103.49         NA      3.30 112.84
655        True   171.97      103.69      103.54         NA      2.70 112.79
676        True   171.60      103.85      103.91         NA      2.56 113.27
711        True   172.03      103.97      103.86         NA      3.07 112.65
740        True   172.07      103.74      103.76         NA      3.09 112.41
743        True   172.14      104.06      103.96         NA      3.24 113.07
781        True   172.41      103.95      103.79         NA      3.13 113.41
799        True   171.96      103.84      103.62         NA      3.01 114.44
845        True   171.62      104.14      104.49         NA      2.99 113.35
846        True   172.02      104.21      104.05         NA      2.90 113.62
872        True   171.37      104.07      103.75         NA      3.07 113.27
896        True   171.81      103.68      103.80         NA      2.98 113.82
920        True   171.92      103.68      103.45         NA      2.58 113.68
946        True   172.09      103.74      103.52         NA      3.02 112.78
947        True   171.63      103.87      104.66         NA      3.27 112.68
982        True   172.02      104.23      103.72         NA      2.99 113.37
1077      False   171.57      104.27      104.44         NA      3.21 111.87
1122      False   171.40      104.38      104.19         NA      3.17 112.39
1177      False   171.59      104.05      103.94         NA      3.02 111.29
1304      False   172.17      104.49      103.76         NA      2.93 111.21
1316      False   172.08      104.15      104.17         NA      3.40 112.29
1348      False   171.72      104.46      104.12         NA      3.61 110.31
1436      False   172.66      104.33      104.41         NA      3.56 111.47
1439      False   171.90      104.28      104.29         NA      3.24 111.49
```

Quelles sont les valeurs de “is_genuine” ?

```r
valeur_unique <- unique(data$is_genuine)
print(valeur_unique)
```

```
[1] "True"  "False"
```

Nous avons 2 valeurs uniques dans la colonne is_genuine => “True” ou “False”

Combien y a-t-il de chacune de ces valeurs ?

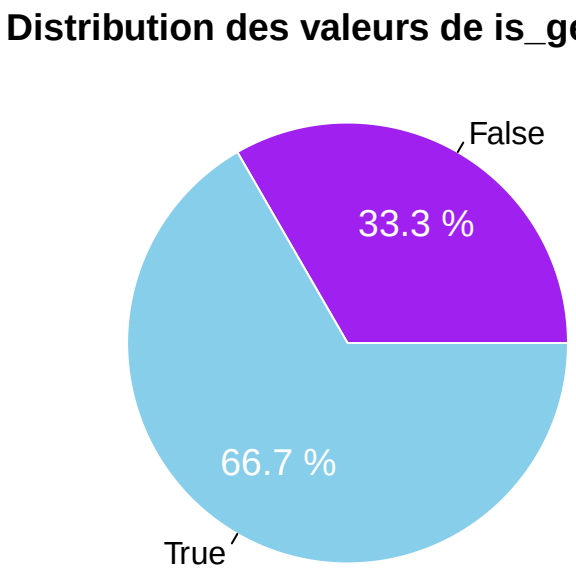
```
valeur_compte <- table(data$is_genuine)
print(valeur_compte)
```

```
False  True
  500  1000
```

Il y a **500 valeurs False** et **1 000 valeurs True**

Visualisation sous forme de diagramme en secteurs

```
# Calcul des pourcentages
pourcentage <- round(valeur_compte / sum(valeur_compte) * 100, 1)
pourcentage_labels <- paste(pourcentage, "%")

# Définir les marges du graphique (gauche, droite, bas, haut)
par(mar = c(1, 1, 1, 1))

# Définir le rapport d'aspect pour le graphique circulaire
par(pty = "s")

# Création du pie chart avec les labels (sans les valeurs)
pie(valeur_compte,
    labels = names(valeur_compte),
    main = "Distribution des valeurs de is_genuine",
    col = c("purple", "skyblue"),
    border = "white")

# Calcul des positions des étiquettes
positions <- cumsum(valeur_compte) - valeur_compte / 2
text(x = cos(2 * pi * positions / sum(valeur_compte)) * 0.5,
     y = sin(2 * pi * positions / sum(valeur_compte)) * 0.5,
     labels = pourcentage_labels,
     cex = 1.2, col = "white")
```



En résumé

Nous avons un tableau regroupant les données de 1 500 billets

1 colonne décrivant s'il s'agit de vrais ou faux billets :
- il y a 1 000 vrais billets 66.7% et 500 faux billets 33.3%

6 colonne décrivants le format de ces billets :
- diagonale
- hauteur gauche
- hauteur droite
- marge basse
- marge haute
- longueur

37 billets n'ont pas l'information de la marge basse ("margin_low") dans le tableau.

Nous allons aggréger les données sur la colonnne is_genuine
Afficher la valeur moyenne de chaque variable pour les lignes False et True afin de voir si nous pouvons observer des différences significatives sur une ou plusieurs variables.

```
if (!require(dplyr)) install.packages("dplyr")
```

```
Le chargement a nécessité le package : dplyr
```

```
Attachement du package : 'dplyr'

Les objets suivants sont masqués depuis 'package:stats':

    filter, lag

Les objets suivants sont masqués depuis 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(dplyr)
```

```r
resultats <- data %>%
  group_by(is_genuine) %>%
  summarise(across(where(is.numeric), \(x) mean(x, na.rm = TRUE)))

print(resultats)
```

```
# A tibble: 2 x 7
  is_genuine diagonal height_left height_right margin_low margin_up length
  <chr>         <dbl>       <dbl>        <dbl>      <dbl>     <dbl>  <dbl>
1 False          172.        104.         104.       5.22      3.35   112.
2 True           172.        104.         104.       4.12      3.05   113.
```

Il y a bien quelques différences de mesures entre les vrais et faux billets mais il est difficile d'utiliser ces moyennes pour notre objectif final qui sera de déterminer d'après les mesures de billets s'il s'agit de "vrais" ou de "faux".

V- Remlacement des NaN de la colonne "margin_low"

Pour commencer il est nécessaire de traiter les valeurs manquantes de notre jeu de données.

Nous avons la possibilité de retirer ces 37 lignes ou bien de déterminer les valeurs manquantes de la colonne "margin_low" en fonction des autres variables présentes à l'aide d'une régréssion linéaire.

Regréssion linéaire multiple : Nous utilisons toutes les variables pour commencer et nous allons voir si elles sont toutes significatives pour la détermination de "margin_low". Nous fixons notre seuil de test à 5% Toutes les p-valeur inférieure à 0.05 seront donc considérée

comme significatives. Si toutes les variables ne sont pas significatives, nous utiliserons la méthode backward et retirerons une à une toutes les variables non significatives.

Pour réaliser notre modèle de régression linéaire, nous allons travailler sur notre jeu de données sans NA

1- Régréssion linéaire

```
# Suppression des lignes où 'margin_low' est NaN
data_without_na <- data[!is.na(data$margin_low), ]
```

```
# Régression linéaire multiple pour prédire la variable margin_low
reg_lin <- lm(margin_low~diagonal+height_left+height_right+margin_up+length, data=data_withou
summary(reg_lin)
```

```
Call:
lm(formula = margin_low ~ diagonal + height_left + height_right +
    margin_up + length, data = data_without_na)

Residuals:
     Min       1Q   Median       3Q      Max
-1.47234 -0.31707 -0.04168  0.27353  1.97084

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  22.99484    9.65555   2.382  0.01737 *
diagonal     -0.11106    0.04144  -2.680  0.00744 **
height_left   0.18412    0.04477   4.113 4.13e-05 ***
height_right  0.25714    0.04301   5.978 2.84e-09 ***
margin_up     0.25619    0.06437   3.980 7.23e-05 ***
length       -0.40910    0.01808 -22.627  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4807 on 1457 degrees of freedom
Multiple R-squared:  0.4773,    Adjusted R-squared:  0.4755
F-statistic: 266.1 on 5 and 1457 DF,  p-value: < 2.2e-16
```

Les variables sont toutes significatives car leur p-value est inférieure à 5% (0.05) le niveau de test que nous souhaitons. Nous conservons donc toutes les variables pour la détermination de 'margin_low'

Nous aurions pu vouloir sélectionner automatiquement un modèle avec l'ensemble des variables avec le package stat

```
reg_null <- lm(margin_low~1, data=data_without_na)
reg_tot <- lm(margin_low~diagonal+height_left+height_right+margin_up+length, data=data)
```

```
#Méthode backward
reg_backward <- step(reg_tot, direction="backward")
```

```
Start:  AIC=-2137.16
margin_low ~ diagonal + height_left + height_right + margin_up +
    length

               Df Sum of Sq    RSS     AIC
<none>                      336.71 -2137.2
- diagonal      1     1.660 338.37 -2132.0
- margin_up     1     3.661 340.37 -2123.3
- height_left   1     3.909 340.62 -2122.3
- height_right  1     8.259 344.97 -2103.7
- length        1   118.316 455.03 -1698.6
```

Le résultat nous montre que toutes les variables explicatives sont importantes dans le modèle pour prédire margin_low. la suppression de length à l'impact le plus négatif sur le modèle augmentant considérablement le RSS (somme des carrés des résidus) chaque suppression rend le modèle moins performant. Le modèle final sera le modèle complet

Partons sur le modèle "reg_lin" Nous pouvons tester le modèle sous forme de prédiction en renseignant toutes les variables pour déterminer une valeur à margin_low

2- Test du modèle

ici nous prenons une ligne au hasard, prenons les valeurs des 5 autres variables et nous allons prédire "margin_low" et contrôler que la valeur est proche de la valeur connue.

```
data[90,]
```

```
   is_genuine diagonal height_left height_right margin_low margin_up length
90       True   171.94      103.49       103.82       4.24      3.19 113.05
```

```r
prev <- data.frame(diagonal=171.94, height_left=103.49, height_right=103.82, margin_up=3.19,
marg_low_prev  <- predict(reg_lin, prev)
round(marg_low_prev,digits=2)
```

```
   1
4.22
```

cela nous donnerait une valeur de 4.22 de “margin_low” d’après le modèle de régréssion linéaire C’est très proche des 4.24 réel.

Nous pourrions améliorer le modèle en calculant les mesures d’influence afin d’identifier et visualiser les observations atypiques. Cela permettrait de mieux comprendre leur impact sur les résultats de la régression linéaire.

3- Mesures d’influence

a- Les Leviers

Les Leviers mesurent l’influence des points de données sur les valeurs prédites, avec un focus sur l’échantillonnage de chaque observation.

Définition des paramètres

```r
#niveau de test
alpha <- 0.05
#nombre d'individus
n <- dim(data_without_na)[1]
#nombre de variables
p <- 6
```

Création d’un dataframe pour l’analyse

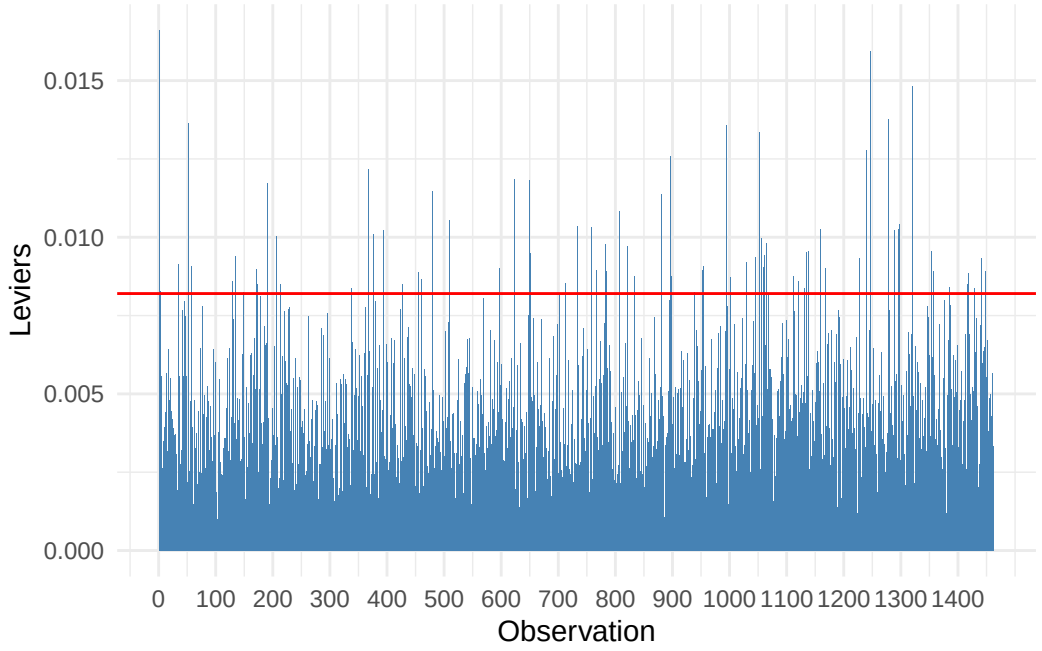
```r
#analyse sur les valeurs atypique ou influantes
analyses <- data.frame(obs=1:n)
```

Calcul des leviers et définition du seuil

```r
#calcul des leviers
analyses$levier <- hat(model.matrix(reg_lin))
seuil_levier <- 2*p/n
```

Visualisation des leviers

```
library(ggplot2)
ggplot(data=analyses, aes(x=obs,y=levier))+
  geom_bar(stat='identity', fill="steelblue")+
  geom_hline(yintercept=seuil_levier, col="red")+
  theme_minimal()+
  xlab("Observation")+
  ylab("Leviers")+
  scale_x_continuous(breaks=seq(0,n,by=100))
```



Ensuite nous pourrions identifier les observations pour lesquelles les leviers sont supérieurs au seuil défini

```
# crée un vecteur logique True = levier qui dépasse le seuil, sinon False
idl <- analyses$levier>seuil_levier
# affiche les premieres valeur du vecteur idl
head(idl)
```

```
[1]  TRUE FALSE  TRUE FALSE FALSE FALSE
```

```r
# Affiche les valeurs de levier qui ont une valeur de vecteur True donc influente
analyses$levier[idl]
```

```
 [1] 0.016606433 0.008297956 0.009156405 0.013635471 0.009087284 0.008615883
 [7] 0.008222240 0.009397300 0.008239495 0.008980313 0.008506194 0.011723023
[13] 0.010037681 0.008513933 0.008384592 0.012165484 0.010106900 0.010243133
[19] 0.008503534 0.008900042 0.008652842 0.011478525 0.010559686 0.009003567
[25] 0.011867888 0.011824475 0.009480505 0.008553764 0.010368443 0.010316756
[31] 0.008959938 0.009778237 0.009661468 0.008907537 0.010832538 0.009722524
[37] 0.008768476 0.011377525 0.012585764 0.008767460 0.008250039 0.008969260
[43] 0.009082513 0.013582547 0.008740641 0.009200472 0.009380825 0.013372012
[49] 0.009973714 0.009054657 0.009427569 0.009814530 0.008751936 0.008610965
[55] 0.008380906 0.009526687 0.009563757 0.010251487 0.009009819 0.009338419
[61] 0.012772324 0.015927673 0.013778658 0.010241817 0.010269781 0.010408617
[67] 0.014823503 0.009557381 0.008913496 0.008397866 0.008521435 0.008870294
[73] 0.008390446 0.009324645 0.008931108
```

Nous pouvons afficher ces observations dans le dataframe initial (data_without_na)

```r
# Récupère les indices des lignes où idl est True
indices_influents <- which(idl)
# Affiche les lignes du DataFrame data_without_na avec ces indices
data_influents <- data_without_na[indices_influents, ]
# Affiche le DataFrame filtré
summary(data_influents)
```

```
  is_genuine           diagonal      height_left     height_right
 Length:75          Min.   :171.0   Min.   :103.2   Min.   :102.9
 Class :character   1st Qu.:171.5   1st Qu.:103.9   1st Qu.:103.6
 Mode  :character   Median :172.0   Median :104.2   Median :104.0
                    Mean   :171.9   Mean   :104.2   Mean   :104.0
                    3rd Qu.:172.3   3rd Qu.:104.5   3rd Qu.:104.4
                    Max.   :173.0   Max.   :104.9   Max.   :105.0
   margin_low      margin_up         length
 Min.   :3.430   Min.   :2.270   Min.   :109.5
 1st Qu.:4.165   1st Qu.:2.885   1st Qu.:111.0
 Median :4.520   Median :3.100   Median :112.8
 Mean   :4.631   Mean   :3.161   Mean   :112.3
 3rd Qu.:5.055   3rd Qu.:3.450   3rd Qu.:113.3
 Max.   :6.080   Max.   :3.910   Max.   :113.9
```

Nous pouvons désormais observer si ces observations concernent plutôt des “vrais” ou “faux” billets.

```
table(data_influents$is_genuine)
```

```
False  True
   32    43
```

Avec ce seuil, On constate qu’il s’agit majoritairement de vrais billets.

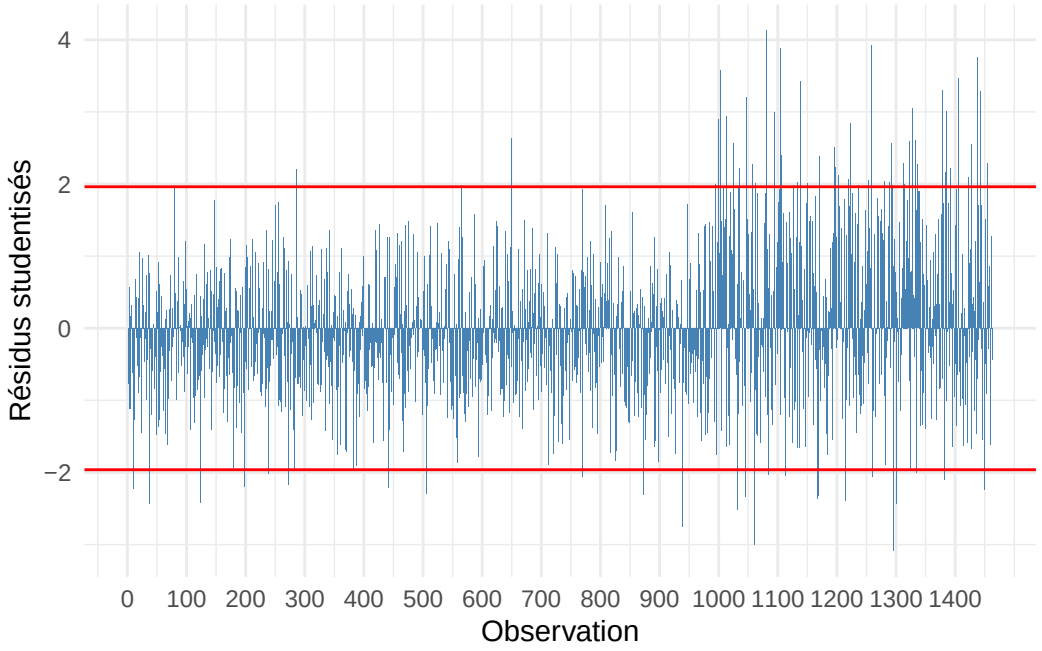
b- Résidus Studentisés

les Résidus Studentisés mesurent combien une observation s’écarte des valeurs attendues en tenant compte de l’échantillon global.

Calculer les résidus studentisés et identifier les résidus extrêmes.

```
analyses$rstudent <- rstudent(reg_lin)
seuil_rstudent <- qt(1-alpha/2,n-p-1)
```

```
ggplot(data=analyses, aes(x=obs,y=rstudent))+
  geom_bar(stat='identity', fill="steelblue")+
  geom_hline(yintercept=-seuil_rstudent, col="red")+
  geom_hline(yintercept=seuil_rstudent, col="red")+
  theme_minimal()+
  xlab("Observation")+
  ylab("Résidus studentisés")+
  scale_x_continuous(breaks=seq(0,n,by=100))
```



Nous pouvons également afficher ces observations dans le dataframe initial (data_without_na)

```
# Crée un vecteur logique True = résidu studentisé au-delà du seuil, sinon False
idr <- abs(analyses$rstudent) > seuil_rstudent
# Récupère les indices des lignes où idl est True
indices_influents_rstudent <- which(idr)
# Affiche les lignes du DataFrame data_without_na avec ces indices
data_influents_rstudent <- data_without_na[indices_influents_rstudent, ]
# Affiche le DataFrame filtré
summary(data_influents_rstudent)
```

```
 is_genuine           diagonal      height_left     height_right
Length:79          Min.   :171.1   Min.   :103.1   Min.   :103.3
Class :character   1st Qu.:171.8   1st Qu.:104.0   1st Qu.:103.8
Mode  :character   Median :171.9   Median :104.2   Median :104.0
                   Mean   :171.9   Mean   :104.2   Mean   :104.1
                   3rd Qu.:172.1   3rd Qu.:104.4   3rd Qu.:104.2
                   Max.   :172.9   Max.   :104.8   Max.   :104.9
  margin_low      margin_up         length
Min.   :2.980   Min.   :2.270   Min.   :110.4
1st Qu.:4.410   1st Qu.:3.105   1st Qu.:111.3
Median :5.820   Median :3.260   Median :112.0
Mean   :5.312   Mean   :3.251   Mean   :112.0
```

```
 3rd Qu.:6.175   3rd Qu.:3.390   3rd Qu.:112.7
 Max.   :6.900   Max.   :3.770   Max.   :113.7
```

et nous pouvons aussi observer si ces observations concernent plutôt des “vrais” ou “faux” billets.

```
table(data_influents_rstudent$is_genuine)
```

```
False  True
   65    14
```

Il s’agit majoritairement de faux billets dans ce cas.

c- Distance de Cook

La Distances de Cook est une combinaison des leviers et des résidus studentisés pour quantifier l’impact total d’une observation sur les coefficients du modèle.

Pour récupérer la distance de Cook

```
influence<-influence.measures(reg_lin)
names(influence)
```

```
[1] "infmat" "is.inf" "call"
```

```
colnames(influence$infmat)
```

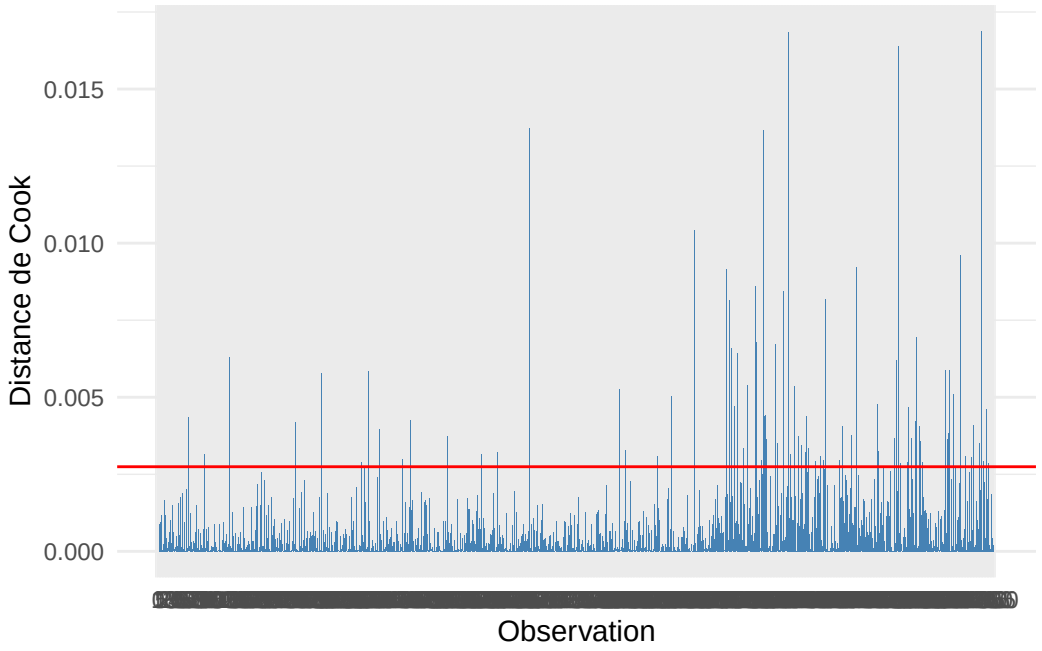
```
 [1] "dfb.1_"     "dfb.dgnl"   "dfb.hght_l" "dfb.hght_r" "dfb.mrg_"
 [6] "dfb.lngt"   "dffit"      "cov.r"      "cook.d"     "hat"
```

```
analyses$dcook <- influence$infmat[,"cook.d"]
seuil_dcook <- 4/(n-p)
```

```
seuil_dcook
```

```
[1] 0.002745367
```

```r
ggplot(data=analyses, aes(x=obs,y=dcook))+
  geom_bar(stat='identity', fill="steelblue")+
  geom_hline(yintercept=seuil_dcook, col="red")+
  theme_minimal()+
  xlab("Observation")+
  ylab("Distance de Cook")+
  scale_x_continuous(breaks=seq(0,n,by=5))
```



observons de quelles lignes il s'agit dans notre jeu de données

```r
# Crée un vecteur logique True = distance de Cook au-delà du seuil, sinon False
idc <- analyses$dcook > seuil_dcook
# Récupère les indices des lignes où idc est True
indices_influents_cook <- which(idc)
# Affiche les lignes du DataFrame data_without_na avec ces indices
data_influents_cook <- data_without_na[indices_influents_cook, ]
# Affiche le DataFrame filtré
summary(data_influents_cook)
```

```
  is_genuine           diagonal      height_left     height_right
 Length:85          Min.   :171.1   Min.   :103.1   Min.   :103.1
 Class :character   1st Qu.:171.6   1st Qu.:104.0   1st Qu.:103.8
```

```
 Mode  :character   Median :171.8   Median :104.2   Median :104.1
                    Mean   :171.9   Mean   :104.2   Mean   :104.1
                    3rd Qu.:172.1   3rd Qu.:104.5   3rd Qu.:104.3
                    Max.   :172.9   Max.   :104.9   Max.   :104.9
   margin_low      margin_up        length
 Min.   :2.980   Min.   :2.27   Min.   :110.2
 1st Qu.:4.260   1st Qu.:3.10   1st Qu.:111.2
 Median :5.520   Median :3.28   Median :112.0
 Mean   :5.196   Mean   :3.26   Mean   :112.0
 3rd Qu.:6.160   3rd Qu.:3.43   3rd Qu.:112.8
 Max.   :6.900   Max.   :3.77   Max.   :113.9
```

Combien concernent des vrais et faux billets ?

```
table(data_influents_cook$is_genuine)
```

```
False  True
   66    19
```

Il s'agit aussi majoritairement de faux billets.

Dans un modèle de régression linéaire, il est important de vérifer deux aspects : - **la colinéarité** La colinéarité se produit lorsque deux ou plusieurs variables indépendantes dans un modèle de régression sont fortement corrélées, ce qui peut fausser les résultats et rendre difficile l'estimation précise des coefficients.

- **l'homoscédascticité** L'homoscédasticité signifie que la variance des erreurs (ou résidus) est constante pour toutes les valeurs des variables explicatives. Cela est une hypothèse importante en régression linéaire car si cette hypothèse est violée (c'est-à-dire en cas d'hétéroscédasticité), les coefficients du modèle peuvent être biaisés et inefficaces.

4- Test de Non-colinéarité et d'homoscédasticité

Pour valider le modèle de régression linéaire et ces résultats, ces deux aspect doivent donc être respectés.

a- Colinéarité

Recherche de colinéarité (commande VIF)

```r
# Installation du package
install.packages("car")
```

```
Installation du package dans 'C:/Users/Utilisateur/AppData/Local/R/win-library/4.4'
(car 'lib' n'est pas spécifié)

le package 'car' a été décompressé et les sommes MD5 ont été vérifiées avec succés

Les packages binaires téléchargés sont dans
    C:\Users\Utilisateur\AppData\Local\Temp\RtmpOO4JuD\downloaded_packages
```

```r
# Charger le package
library(car)
```

```
Le chargement a nécessité le package : carData


Attachement du package : 'car'

L'objet suivant est masqué depuis 'package:dplyr':

    recode
```

```r
vif(reg_lin)
```

```
    diagonal  height_left height_right    margin_up       length
    1.013613     1.138261     1.230115     1.404404     1.576950
```

Tous les VIF étant inférieur à 10, il ne semble pas y avoir de problème de colinéarité
Les variables explicatives sont suffisamment indépendantes les unes des autres.

On peut également tester l'homoscédasticité, c'est à dire la constance de la variance des erreurs
On peut donc réaliser le test de Breusch-Pagan

b- Homoscédasticité

```
install.packages("lmtest")
```

```
Installation du package dans 'C:/Users/Utilisateur/AppData/Local/R/win-library/4.4'
(car 'lib' n'est pas spécifié)

le package 'lmtest' a été décompressé et les sommes MD5 ont été vérifiées avec succés

Warning: impossible de supprimer l'installation précédente du package 'lmtest'

Warning in file.copy(savedcopy, lib, recursive = TRUE): problème lors de la
copie de
C:\Users\Utilisateur\AppData\Local\R\win-library\4.4\00LOCK\lmtest\libs\x64\lmtest.dll
vers
C:\Users\Utilisateur\AppData\Local\R\win-library\4.4\lmtest\libs\x64\lmtest.dll :
Permission denied

Warning: 'lmtest' restauré


Les packages binaires téléchargés sont dans
    C:\Users\Utilisateur\AppData\Local\Temp\RtmpOO4JuD\downloaded_packages
```

```
library(lmtest)
```

```
Le chargement a nécessité le package : zoo


Attachement du package : 'zoo'

Les objets suivants sont masqués depuis 'package:base':

    as.Date, as.Date.numeric
```

```
bptest(reg_lin)
```

```
    studentized Breusch-Pagan test

data:  reg_lin
BP = 80.163, df = 5, p-value = 7.76e-16
```

Hypothèse nulle (H) : Les résidus du modèle de régression sont homoscédastiques, c'est-à-dire que la variance des erreurs est constante.

Les résultats du test montrent une p-valeur extrêmement faible (p = 7,76e-16), bien inférieure au seuil de 0,05. Par conséquent, nous rejetons l'hypothèse nulle (H) et concluons à la présence d'hétéroscédasticité dans le modèle.

Cela implique que la variance des erreurs n'est pas constante, ce qui peut fausser les estimations des erreurs standards des coefficients de régression. Par conséquent, la validité des tests d'hypothèses et des intervalles de confiance est compromise, rendant le modèle présenté peu fiable dans sa forme actuelle.

Pour résoudre ce problème, nous allons appliquer des transformations aux variables, notamment en utilisant les méthodes de transformation logarithmique, au carré et inverse. Ces transformations visent à stabiliser la variance des résidus et à réduire l'hétéroscédasticité.

En effectuant ces transformations sur certaines variables explicatives et/ou sur la variable dépendante, nous espérons obtenir une distribution d'erreurs plus homogène, ce qui rendra les estimations des coefficients de régression plus robustes. Cette approche améliorera ainsi la fiabilité du modèle, tant en termes d'interprétation des coefficients que de validité des tests d'hypothèse.

5- Application de Transformations pour Stabiliser la Variance des Résidus

a- La transformation logarithmique

```r
# Régression linéaire multiple pour prédire la variable margin_low avec transformation lorga
reg_lin_log <- lm(log(margin_low)~log(diagonal)+log(height_left)+log(height_right)+log(margi
summary(reg_lin_log)
```

```
Call:
lm(formula = log(margin_low) ~ log(diagonal) + log(height_left) +
    log(height_right) + log(margin_up) + log(length), data = data_without_na)

Residuals:
     Min       1Q   Median       3Q      Max
```

```
-0.32146 -0.06889 -0.00503  0.06419  0.36831

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       21.44866   10.18653   2.106   0.0354 *
log(diagonal)     -3.88889    1.53135  -2.540   0.0112 *
log(height_left)   4.07743    1.00085   4.074 4.87e-05 ***
log(height_right)  5.81928    0.96027   6.060 1.73e-09 ***
log(margin_up)     0.17338    0.04339   3.995 6.78e-05 ***
log(length)       -9.75781    0.43520 -22.422  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1033 on 1457 degrees of freedom
Multiple R-squared:  0.4726,    Adjusted R-squared:  0.4708
F-statistic: 261.1 on 5 and 1457 DF,  p-value: < 2.2e-16
```

Reprenons la ligne du test de prédiction de “margin_low” et contrôlons la valeur obtenue.

```
data[90,]
```

```
   is_genuine diagonal height_left height_right margin_low margin_up length
90       True   171.94      103.49       103.82       4.24      3.19 113.05
```

```
prev <- data.frame(diagonal=171.94, height_left=103.49, height_right=103.82, margin_up=3.19,
marg_low_prev_log  <- predict(reg_lin_log, prev)
marg_low_prev <- exp(marg_low_prev_log)
round(marg_low_prev,digits=2)
```

```
   1
4.19
```

Nous pouvons refaire le test d’homoscédasticité

```
bptest(reg_lin_log)
```

```
    studentized Breusch-Pagan test

data:  reg_lin_log
BP = 32.536, df = 5, p-value = 4.652e-06
```

Hypothèse nulle (H0) : Les résidus du modèle de régression sont homoscédastiques (la variance des erreurs est constante). Hypothèse alternative (H1) : Les résidus du modèle de régression ne sont pas homoscédastiques (il y a de l'hétéroscédasticité).

P-value : La p-value obtenue est 4.652e-06, ce qui est extrêmement inférieure au seuil de significativité habituel de 0.05.

Étant donné que la p-value est bien inférieure à 0.05, nous pouvons rejeter l'hypothèse nulle (H0). Cela indique qu'il reste des preuves significatives d'hétéroscédasticité dans le modèle.

b- La transformation quadratique (Racine Carré)

```
reg_lin_sqrt <- lm(sqrt(margin_low) ~ sqrt(diagonal) + sqrt(height_left) + sqrt(height_right)
summary(reg_lin_sqrt)
```

```
Call:
lm(formula = sqrt(margin_low) ~ sqrt(diagonal) + sqrt(height_left) +
    sqrt(height_right) + sqrt(margin_up) + sqrt(length), data = data_without_na)

Residuals:
     Min       1Q   Median       3Q      Max
-0.34343 -0.07369 -0.00776  0.06579  0.42494

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       10.25298    4.45623   2.301  0.02154 *
sqrt(diagonal)    -0.65653    0.25082  -2.618  0.00895 **
sqrt(height_left)  0.86459    0.21077   4.102 4.32e-05 ***
sqrt(height_right) 1.22092    0.20236   6.033 2.03e-09 ***
sqrt(margin_up)    0.21099    0.05268   4.005 6.51e-05 ***
sqrt(length)      -1.99442    0.08833 -22.579  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1109 on 1457 degrees of freedom
Multiple R-squared:  0.4763,    Adjusted R-squared:  0.4745
F-statistic:   265 on 5 and 1457 DF,  p-value: < 2.2e-16
```

```
prev <- data.frame(diagonal=171.94, height_left=103.49, height_right=103.82, margin_up=3.19,
marg_low_prev_sqrt  <- predict(reg_lin_sqrt, prev)
marg_low_prev <- marg_low_prev_sqrt^2
round(marg_low_prev,digits=2)
```

```
   1
4.21
```

```r
bptest(reg_lin_sqrt)
```

```
    studentized Breusch-Pagan test

data:  reg_lin_sqrt
BP = 56.039, df = 5, p-value = 7.978e-11
```

Hypothèse nulle (H0) : Les résidus du modèle de régression sont homoscédastiques (la variance des erreurs est constante). Hypothèse alternative (H1) : Les résidus du modèle de régression ne sont pas homoscédastiques (il y a de l'hétéroscédasticité).

P-value : La p-value obtenue est 7.978e-11, ce qui est extrêmement inférieure au seuil de significativité habituel de 0.05.

Étant donné que la p-value est bien inférieure à 0.05, vous pouvez rejeter l'hypothèse nulle (H0). Cela indique qu'il y a des preuves significatives d'hétéroscédasticité dans les résidus de votre modèle, même après avoir appliqué la transformation racine carrée.

c- La transformation Inverse

```r
# Ajuster le modèle avec la transformation inverse sur la variable dépendante
reg_lin_inverse <- lm(I(1 / margin_low) ~ diagonal + height_left + height_right + margin_up +
summary(reg_lin_inverse)
```

```
Call:
lm(formula = I(1/margin_low) ~ diagonal + height_left + height_right +
    margin_up + length, data = data_without_na)

Residuals:
      Min        1Q    Median        3Q       Max
-0.070225 -0.015351 -0.000103  0.014421  0.089536

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.465519   0.461391  -1.009   0.3132
diagonal      0.004601   0.001980   2.323   0.0203 *
```

```
height_left  -0.008454   0.002139  -3.952 8.13e-05 ***
height_right -0.012304   0.002055  -5.986 2.70e-09 ***
margin_up    -0.012259   0.003076  -3.985 7.07e-05 ***
length        0.018626   0.000864  21.559  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.02297 on 1457 degrees of freedom
Multiple R-squared:  0.4576,    Adjusted R-squared:  0.4558
F-statistic: 245.9 on 5 and 1457 DF,  p-value: < 2.2e-16
```

```
prev <- data.frame(diagonal=171.94, height_left=103.49, height_right=103.82, margin_up=3.19,
marg_low_prev_inverse  <- predict(reg_lin_inverse, prev)
marg_low_prev <- 1 /marg_low_prev_inverse
round(marg_low_prev,digits=2)
```

```
   1
4.17
```

```
bptest(reg_lin_inverse)
```

```
    studentized Breusch-Pagan test

data:  reg_lin_inverse
BP = 4.519, df = 5, p-value = 0.4773
```

Hypothèse du Test Hypothèse nulle (H0) : Les résidus du modèle de régression sont homoscédastiques (la variance des erreurs est constante). Hypothèse alternative (H1) : Les résidus du modèle de régression ne sont pas homoscédastiques (il y a de l'hétéroscédasticité).

p-value : La p-value obtenue est de 0.4773.

Comme la p-value est bien supérieure au seuil de signification de 0.05, on ne rejete pas l'hypothèse nulle. Cela indique qu'il n'y a pas suffisamment de preuves pour conclure qu'il existe une hétéroscédasticité significative dans les résidus du modèle où seule la variable dépendante a été transformée par l'inverse.

Nous allons utiliser la régression linéaire, en appliquant une transformation inverse à la variable dépendante, afin de garantir la non-colinéarité et l'homoscédasticité des résidus. La non-colinéarité permet d'éviter la redondance entre les variables explicatives, assurant ainsi des estimations fiables des coefficients, tandis que l'homoscédasticité garantit que la variance des résidus reste constante, condition essentielle pour des tests statistiques valides et des intervalles de confiance précis.

V- Remplacement des données manquantes

```
# Prédire les valeurs manquantes
predicted_values_inverse <- predict(reg_lin_inverse, newdata = missing_margin_low)
predicted_values <- 1/predicted_values_inverse
```

```
predicted_values
```

```
      73       100       152       198       242       252       285       335
4.249525 4.319269 4.335315 4.253347 4.545341 3.854438 4.141135 4.099717
     411       414       446       482       506       612       655       676
4.107694 4.125593 4.135570 3.839329 4.050024 4.232304 4.118881 4.067789
     711       740       743       781       799       845       846       872
4.355167 4.381050 4.275951 4.068161 3.739451 4.300033 4.075794 4.190548
     896       920       946       947       982      1077      1122      1177
3.926399 3.819787 4.183214 4.610819 4.107073 4.960844 4.692223 4.969965
    1304      1316      1348      1436      1439
4.950863 4.678245 5.860321 5.135740 5.065545
```

```
# Étape 3: Remplacer les NA par les valeurs prédites
data$margin_low[is.na(data$margin_low)] <- predicted_values
```

```
# Voir le résultat
summary(data$margin_low)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.980   4.020   4.310   4.483   4.870   6.900
```

```
missing_margin_low
```

```
     is_genuine diagonal height_left height_right margin_low margin_up length
73         True   171.94      103.89       103.45         NA      3.25 112.79
100        True   171.93      104.07       104.18         NA      3.14 113.08
152        True   172.07      103.80       104.38         NA      3.02 112.93
198        True   171.45      103.66       103.80         NA      3.62 113.27
242        True   171.83      104.14       104.06         NA      3.02 112.36
252        True   171.80      103.26       102.82         NA      2.95 113.22
285        True   171.92      103.83       103.76         NA      3.23 113.29
335        True   171.85      103.70       103.96         NA      3.00 113.36
```

411	True	172.56	103.72	103.51	NA	3.12	112.95
414	True	172.30	103.66	103.50	NA	3.16	112.95
446	True	172.34	104.42	103.22	NA	3.01	112.97
482	True	171.81	103.53	103.96	NA	2.71	113.99
506	True	172.01	103.97	104.05	NA	2.98	113.65
612	True	171.80	103.68	103.49	NA	3.30	112.84
655	True	171.97	103.69	103.54	NA	2.70	112.79
676	True	171.60	103.85	103.91	NA	2.56	113.27
711	True	172.03	103.97	103.86	NA	3.07	112.65
740	True	172.07	103.74	103.76	NA	3.09	112.41
743	True	172.14	104.06	103.96	NA	3.24	113.07
781	True	172.41	103.95	103.79	NA	3.13	113.41
799	True	171.96	103.84	103.62	NA	3.01	114.44
845	True	171.62	104.14	104.49	NA	2.99	113.35
846	True	172.02	104.21	104.05	NA	2.90	113.62
872	True	171.37	104.07	103.75	NA	3.07	113.27
896	True	171.81	103.68	103.80	NA	2.98	113.82
920	True	171.92	103.68	103.45	NA	2.58	113.68
946	True	172.09	103.74	103.52	NA	3.02	112.78
947	True	171.63	103.87	104.66	NA	3.27	112.68
982	True	172.02	104.23	103.72	NA	2.99	113.37
1077	False	171.57	104.27	104.44	NA	3.21	111.87
1122	False	171.40	104.38	104.19	NA	3.17	112.39
1177	False	171.59	104.05	103.94	NA	3.02	111.29
1304	False	172.17	104.49	103.76	NA	2.93	111.21
1316	False	172.08	104.15	104.17	NA	3.40	112.29
1348	False	171.72	104.46	104.12	NA	3.61	110.31
1436	False	172.66	104.33	104.41	NA	3.56	111.47
1439	False	171.90	104.28	104.29	NA	3.24	111.49

On contrôle ici que les valeurs déterminées par le modèle ont bien remplacées les NA de notre jeu de donnée “data”

```
indices <- rownames(missing_margin_low)
lignes_data <- data[indices, ]
print(lignes_data)
```

	is_genuine	diagonal	height_left	height_right	margin_low	margin_up	length
73	True	171.94	103.89	103.45	4.249525	3.25	112.79
100	True	171.93	104.07	104.18	4.319269	3.14	113.08
152	True	172.07	103.80	104.38	4.335315	3.02	112.93
198	True	171.45	103.66	103.80	4.253347	3.62	113.27

242	True	171.83	104.14	104.06	4.545341	3.02	112.36
252	True	171.80	103.26	102.82	3.854438	2.95	113.22
285	True	171.92	103.83	103.76	4.141135	3.23	113.29
335	True	171.85	103.70	103.96	4.099717	3.00	113.36
411	True	172.56	103.72	103.51	4.107694	3.12	112.95
414	True	172.30	103.66	103.50	4.125593	3.16	112.95
446	True	172.34	104.42	103.22	4.135570	3.01	112.97
482	True	171.81	103.53	103.96	3.839329	2.71	113.99
506	True	172.01	103.97	104.05	4.050024	2.98	113.65
612	True	171.80	103.68	103.49	4.232304	3.30	112.84
655	True	171.97	103.69	103.54	4.118881	2.70	112.79
676	True	171.60	103.85	103.91	4.067789	2.56	113.27
711	True	172.03	103.97	103.86	4.355167	3.07	112.65
740	True	172.07	103.74	103.76	4.381050	3.09	112.41
743	True	172.14	104.06	103.96	4.275951	3.24	113.07
781	True	172.41	103.95	103.79	4.068161	3.13	113.41
799	True	171.96	103.84	103.62	3.739451	3.01	114.44
845	True	171.62	104.14	104.49	4.300033	2.99	113.35
846	True	172.02	104.21	104.05	4.075794	2.90	113.62
872	True	171.37	104.07	103.75	4.190548	3.07	113.27
896	True	171.81	103.68	103.80	3.926399	2.98	113.82
920	True	171.92	103.68	103.45	3.819787	2.58	113.68
946	True	172.09	103.74	103.52	4.183214	3.02	112.78
947	True	171.63	103.87	104.66	4.610819	3.27	112.68
982	True	172.02	104.23	103.72	4.107073	2.99	113.37
1077	False	171.57	104.27	104.44	4.960844	3.21	111.87
1122	False	171.40	104.38	104.19	4.692223	3.17	112.39
1177	False	171.59	104.05	103.94	4.969965	3.02	111.29
1304	False	172.17	104.49	103.76	4.950863	2.93	111.21
1316	False	172.08	104.15	104.17	4.678245	3.40	112.29
1348	False	171.72	104.46	104.12	5.860321	3.61	110.31
1436	False	172.66	104.33	104.41	5.135740	3.56	111.47
1439	False	171.90	104.28	104.29	5.065545	3.24	111.49

Puis on contrôle qu'il n'y a plus de valeur manquante dans "data"

```
skim(data)
```

Table 4: Data summary

Name	data
Number of rows	1500

Number of columns	7
Column type frequency:	
character	1
numeric	6
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
is_genuine	0	1	4	5	0	2	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
diagonal	0	1	171.96	0.31	171.04	171.75	171.96	172.17	173.01	
height_left	0	1	104.03	0.30	103.14	103.82	104.04	104.23	104.88	
height_right	0	1	103.92	0.33	102.82	103.71	103.92	104.15	104.95	
margin_low	0	1	4.48	0.66	2.98	4.02	4.31	4.87	6.90	
margin_up	0	1	3.15	0.23	2.27	2.99	3.14	3.31	3.91	
length	0	1	112.68	0.87	109.49	112.03	112.96	113.34	114.44	

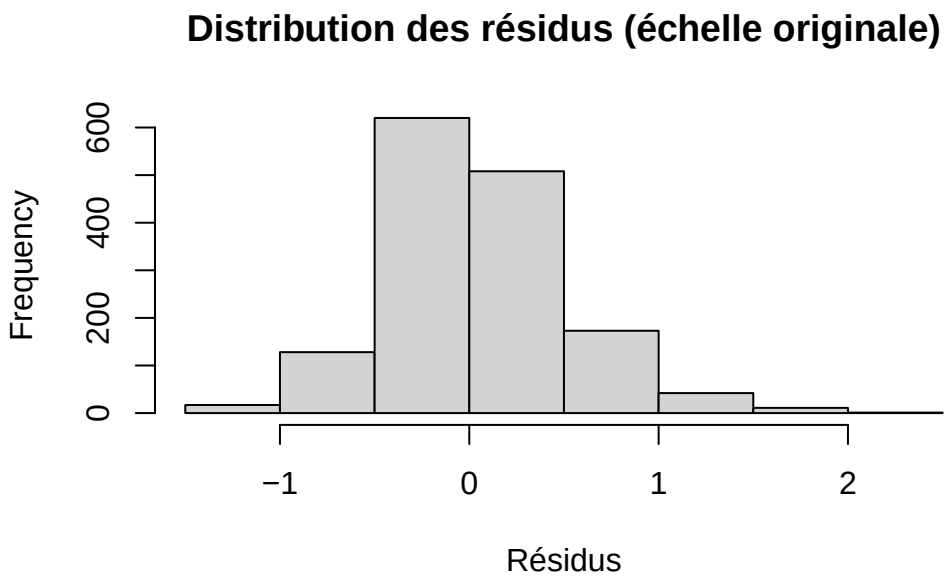
Analyse des résidus de la régression logistique sur l'ensemble de "data"

Nous pouvons visualiser les résidus à l'échelle originale sur l'ensemble du dataframe

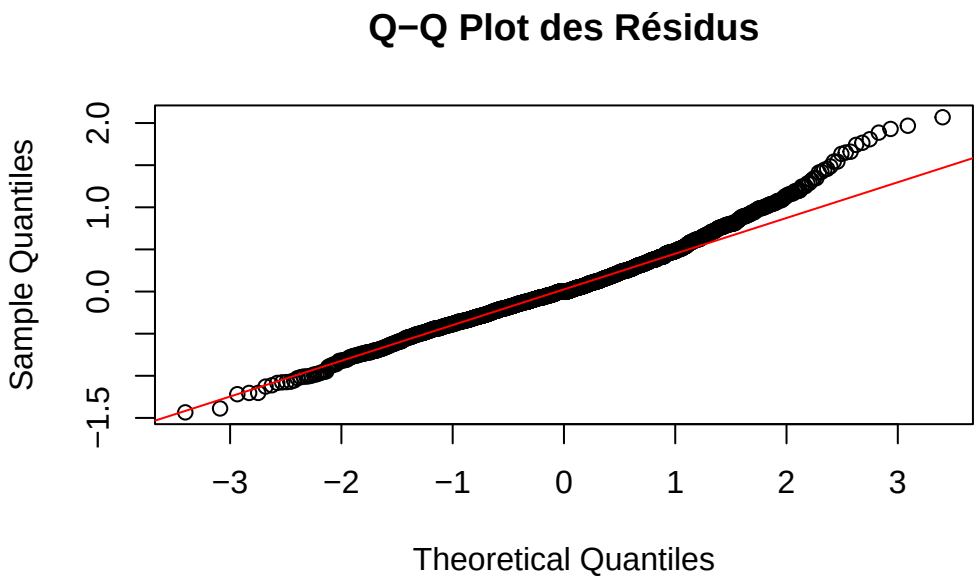
```
predicted_values_inv_data <- predict(reg_lin_inverse, newdata = data)
predicted_values_original_data <- 1 / predicted_values_inv_data
residus_original <- data$margin_low - predicted_values_original_data

# Résumé des résidus dans l'échelle originale
summary(residus_original)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.43393 -0.26051  0.00000  0.04719  0.31127  2.06892
```

```r
# Visualiser les résidus dans l'échelle originale
hist(residus_original, main = "Distribution des résidus (échelle originale)", xlab = "Résidus
```




```r
# Q-Q Plot des résidus pour vérifier la normalité
qqnorm(residus_original, main = "Q-Q Plot des Résidus")
qqline(residus_original, col = "red")
```



Les résidus montrent une distribution qui est globalement normale et symétrique autour de zéro, ce qui est un bon indicateur que les hypothèses de normalité sont respectées. L'histogramme des résidus présente une forme de cloche, et le Q-Q plot confirme que les résidus se rapprochent d'une distribution normale même si on peut observer un détachement en queue par rapport à la normale.

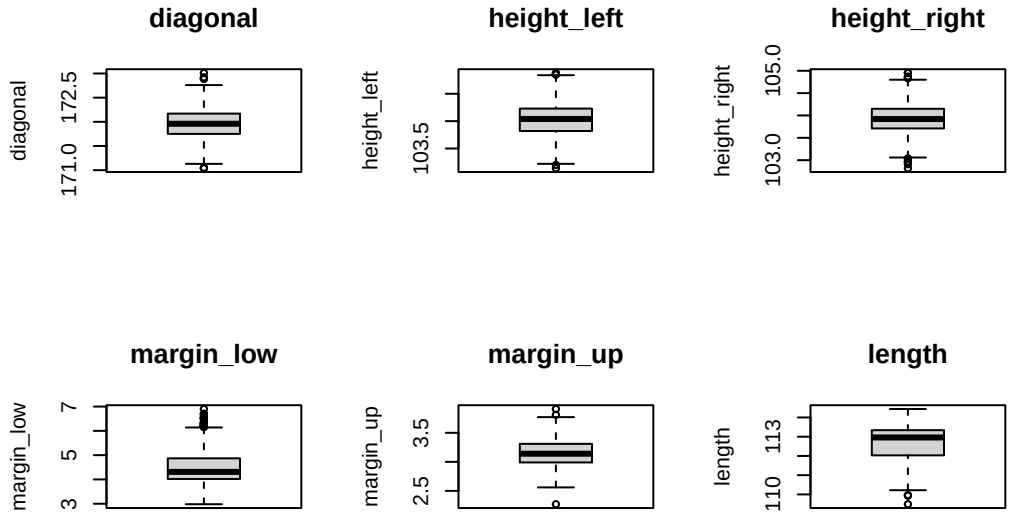
VI- ACP et Clustering

Nous allons désormais réaliser une analyse exploratoire des regroupements possible de billets à l'aide de la méthode des K-means qui est une méthode non supervisée et qui nous donnera un aperçu du regroupement naturel des billets en fonction de leurs mesures sans tenir compte de l'information "vrai" et "faux" de "is_genuine". Et nous allons pour commencer réaliser une ACP pour réduire la dimensionnalité et découvrir les corrélations entre les variables.

1- Recherche de valeurs aberrantes

a- Boxplot des variables

```r
# Boxplot pour chaque variable numérique
data_numeric <- data %>% select(where(is.numeric))

# Afficher boxplots
par(mfrow = c(2, 3))  # Adapter le nombre de graphiques selon le nombre de variables
for (col in colnames(data_numeric)) {
  boxplot(data_numeric[[col]], main = col, ylab = col)
}
```



b- Scatterplot des paires de variables

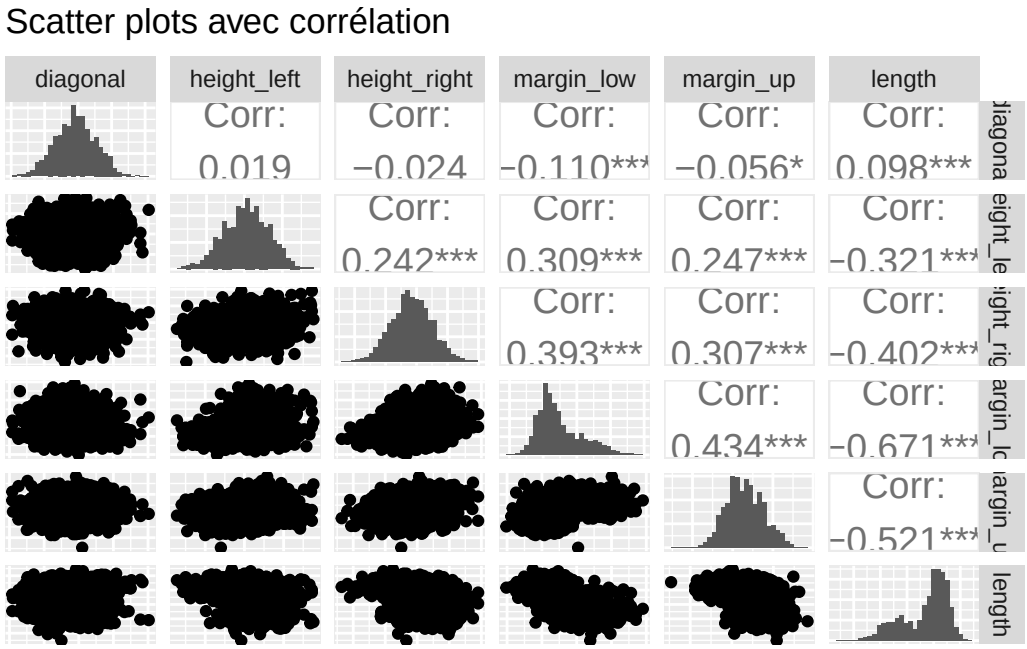
```r
if (!require(GGally)) install.packages("GGally")
```

```
Le chargement a nécessité le package : GGally

Registered S3 method overwritten by 'GGally':
  method from
  +.gg   ggplot2
```

```r
library(GGally)
# Affichage des scatter plots avec corrélation, noms des variables, et sans valeurs des axes
plot <- ggpairs(data[, c("diagonal", "height_left", "height_right", "margin_low", "margin_up
                title = "Scatter plots avec corrélation",
                upper = list(continuous = wrap("cor", size = 5)),   # Corrélation dans la pa
                lower = list(continuous = "points"),               # Nuages de points dans l
                diag = list(continuous = wrap("barDiag", size = 5)),# Distribution diagonale
                axisLabels = "show")                               # Affiche les noms des va

# Personnaliser pour retirer les valeurs numériques des axes mais garder les noms des variab
plot <- plot + theme(
  axis.text.x = element_blank(),  # Supprime les valeurs numériques de l'axe des abscisses
  axis.text.y = element_blank(),  # Supprime les valeurs numériques de l'axe des ordonnées
  axis.ticks = element_blank()    # Supprime les ticks des axes
)

# Afficher le graphique
print(plot)
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



La dispersion des points indique qu'il n'y a pas de corrélation évidente et linéaire entre 2 variables

c- Méthode IQR

```
# Calculer l'IQR pour chaque colonne
iqr <- function(x) {
  q3 <- quantile(x, 0.75)
  q1 <- quantile(x, 0.25)
  q3 - q1
}

# Détecter les outliers pour chaque colonne
outliers <- function(x) {
  q1 <- quantile(x, 0.25)
  q3 <- quantile(x, 0.75)
  iqr_value <- iqr(x)
  lower_bound <- q1 - 1.5 * iqr_value
  upper_bound <- q3 + 1.5 * iqr_value
  x < lower_bound | x > upper_bound
}
```

```r
# Appliquer la fonction pour détecter les outliers
iqr_outliers_data <- data_numeric %>%
  mutate(across(everything(), ~ outliers(.)))

# Afficher les outliers
head(iqr_outliers_data)
```

```
  diagonal height_left height_right margin_low margin_up length
1    FALSE        TRUE         TRUE      FALSE     FALSE  FALSE
2    FALSE       FALSE        FALSE      FALSE     FALSE  FALSE
3    FALSE       FALSE        FALSE      FALSE     FALSE  FALSE
4    FALSE       FALSE        FALSE      FALSE     FALSE  FALSE
5    FALSE       FALSE        FALSE      FALSE     FALSE  FALSE
6    FALSE       FALSE        FALSE      FALSE     FALSE  FALSE
```

```r
# Afficher les indices des outliers
iqr_outliers_indices <- which(rowSums(iqr_outliers_data) > 0)
print(iqr_outliers_indices)
```

```
 [1]    1   78  177  194  225  252  293  523  665  730  762  829  843 1023 1024
[16] 1028 1030 1032 1042 1054 1076 1083 1091 1093 1111 1125 1134 1135 1143 1151
[31] 1170 1200 1255 1271 1278 1291 1322 1323 1332 1346 1349 1354 1356 1383 1389
[46] 1421 1427 1442 1454 1460 1465 1474 1485
```

```r
# Afficher les lignes du dataframe original contenant des outliers
data_with_iqr_outliers <- data[iqr_outliers_indices, ]

print(data_with_iqr_outliers)
```

```
     is_genuine diagonal height_left height_right margin_low margin_up length
1          True   171.81      104.86       104.95   4.520000      2.89 112.83
78         True   171.84      104.09       103.03   4.110000      2.77 113.18
177        True   171.75      103.63       102.97   4.460000      2.77 113.22
194        True   172.35      103.73       102.95   4.490000      3.37 112.49
225        True   172.12      103.20       103.92   4.460000      3.26 113.44
252        True   171.80      103.26       102.82   3.854438      2.95 113.22
293        True   172.09      103.14       103.81   4.880000      3.01 113.69
523        True   172.02      104.42       102.91   3.860000      3.12 113.43
665        True   172.05      103.70       103.75   5.040000      2.27 113.55
730        True   171.04      103.84       103.64   4.220000      3.36 112.70
```

762	True	172.16	103.93	103.04	4.140000	2.99	113.26
829	True	172.92	103.55	103.94	4.780000	3.27	113.55
843	True	172.89	103.77	104.24	4.120000	3.01	113.72
1023	False	172.89	104.03	104.03	6.030000	3.00	110.95
1024	False	172.02	104.26	104.20	6.200000	3.58	111.25
1028	False	171.63	104.02	104.66	6.700000	3.28	111.28
1030	False	171.96	104.29	104.03	6.010000	3.91	110.83
1032	False	172.40	104.00	103.82	6.330000	3.10	112.11
1042	False	171.77	104.12	104.42	6.650000	3.63	111.53
1054	False	171.85	104.52	104.05	6.210000	3.43	111.96
1076	False	172.02	104.51	103.69	6.230000	3.39	112.35
1083	False	171.75	103.96	103.83	5.390000	3.54	109.49
1091	False	172.11	104.88	104.10	4.800000	3.73	110.78
1093	False	171.87	104.76	104.02	6.300000	3.61	111.29
1111	False	171.73	104.32	104.07	6.560000	3.30	112.80
1125	False	171.88	103.92	104.27	6.700000	3.11	110.93
1134	False	171.79	103.99	103.67	6.160000	3.52	110.93
1135	False	171.91	103.70	104.41	6.340000	3.50	113.05
1143	False	171.04	104.23	104.22	4.870000	3.56	111.54
1151	False	171.79	104.86	104.34	5.390000	3.14	113.02
1170	False	171.99	104.14	104.15	6.480000	3.42	112.16
1200	False	172.03	104.32	104.87	4.490000	3.77	111.04
1255	False	171.15	104.09	104.30	6.490000	3.20	111.61
1271	False	171.26	104.22	104.07	4.780000	3.81	112.88
1278	False	173.01	104.59	104.31	5.040000	3.05	110.91
1291	False	171.94	104.06	104.22	6.900000	3.36	111.70
1322	False	172.29	104.72	104.86	5.710000	3.16	112.15
1323	False	172.07	104.50	104.23	6.190000	3.07	111.21
1332	False	172.32	104.60	104.83	4.840000	3.51	112.55
1346	False	171.56	104.17	103.87	6.160000	3.38	111.55
1349	False	171.84	104.32	104.50	6.280000	3.00	111.06
1354	False	171.61	104.04	104.06	6.190000	3.08	110.73
1356	False	171.68	103.89	103.70	5.970000	3.03	109.97
1383	False	171.97	104.85	104.52	5.870000	3.56	110.98
1389	False	171.05	104.09	104.50	4.720000	3.10	112.44
1421	False	171.56	104.47	104.04	6.380000	3.43	112.12
1427	False	172.22	103.92	104.03	6.250000	3.14	110.89
1442	False	171.63	104.55	103.81	6.560000	3.10	111.87
1454	False	171.55	104.20	104.49	5.420000	3.54	109.93
1460	False	171.78	104.31	103.82	6.190000	3.25	111.14
1465	False	172.07	104.17	104.37	6.540000	3.54	111.20
1474	False	171.76	104.04	104.12	6.290000	3.20	112.87
1485	False	172.08	103.96	104.95	5.220000	3.45	112.07

```r
library(dplyr)

# Comptage du nombre de lignes par groupe dans la colonne is_genuine
iqr_count_by_is_genuine <- data_with_iqr_outliers %>%
  group_by(is_genuine) %>%
  summarise(count = n())

# Affichage du résultat
print(iqr_count_by_is_genuine)
```

```
# A tibble: 2 x 2
  is_genuine count
  <chr>      <int>
1 False         40
2 True          13
```

Il y a 53 lignes d'outliers avec la méthode IQR
40 lignes concernent des faux billets
13 lignes concernent de vrais billets

d- Méthode Z_score

```r
if (!require(tidyverse)) install.packages("tidyverse")
```

```
Le chargement a nécessité le package : tidyverse
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v forcats   1.0.0     v stringr   1.5.1
v lubridate 1.9.3     v tibble    3.2.1
v purrr     1.0.2     v tidyr     1.3.1
v readr     2.1.5
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
x car::recode()   masks dplyr::recode()
x purrr::some()   masks car::some()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```r
# Charger les bibliothèques nécessaires
library(tidyverse)

# Sélectionner les colonnes numériques
data_numeric <- data %>% select(where(is.numeric))

# Calculer les scores Z
data_z <- scale(data_numeric)

# Vérifier les scores Z
head(data_z)
```

```
       diagonal height_left height_right  margin_low  margin_up     length
[1,] -0.4863774   2.7731984   3.16218582  0.05682068 -1.1279488  0.1735932
[2,] -1.6331847  -2.2357896  -0.79940117 -1.08045148 -0.6965669  0.4715090
[3,]  2.3970239   1.5042548  -1.29076079 -0.12514286 -0.9122578  0.5517171
[4,] -1.9608439  -0.3991607   0.06047818 -1.30790591 -0.6102905  0.9527576
[5,] -0.7485048   0.8363897  -1.41360070 -0.67103350  1.4172048 -0.1586975
[6,]  0.6931959  -0.9668460   0.49041785 -0.09481561 -0.8691196  0.1506766
```

```r
summary(data_z)
```

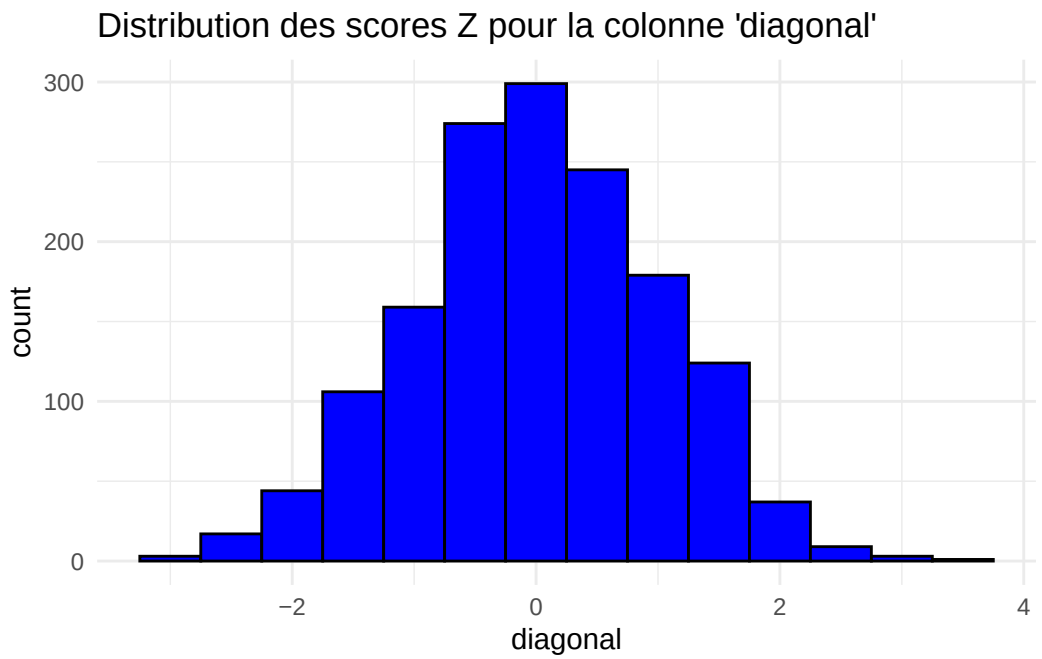
```
    diagonal          height_left        height_right         margin_low
 Min.   :-3.009353   Min.   :-2.97044   Min.   :-3.379039   Min.   :-2.2784
 1st Qu.:-0.682973   1st Qu.:-0.69970   1st Qu.:-0.645851   1st Qu.:-0.7014
 Median : 0.005111   Median : 0.03495   Median :-0.000942   Median :-0.2616
 Mean   : 0.000000   Mean   : 0.00000   Mean   : 0.000000   Mean   : 0.0000
 3rd Qu.: 0.693196   3rd Qu.: 0.66942   3rd Qu.: 0.705388   3rd Qu.: 0.5875
 Max.   : 3.445533   Max.   : 2.83998   Max.   : 3.162186   Max.   : 3.6658
   margin_up           length
 Min.   :-3.80252   Min.   :-3.6535
 1st Qu.:-0.69657   1st Qu.:-0.7431
 Median :-0.04949   Median : 0.3226
 Mean   : 0.00000   Mean   : 0.0000
 3rd Qu.: 0.68385   3rd Qu.: 0.7580
 Max.   : 3.27215   Max.   : 2.0184
```

```r
# Ajouter les scores Z au dataframe original en conservant la colonne de caractères
data_z_full <- cbind(Row_Index = 1:nrow(data), data %>% select(where(is.character)), as.data
```

```r
# Afficher les premières lignes du nouveau dataframe
head(data_z_full)
```

```
  Row_Index is_genuine   diagonal height_left height_right  margin_low
1         1       True -0.4863774   2.7731984   3.16218582  0.05682068
2         2       True -1.6331847  -2.2357896  -0.79940117 -1.08045148
3         3       True  2.3970239   1.5042548  -1.29076079 -0.12514286
4         4       True -1.9608439  -0.3991607   0.06047818 -1.30790591
5         5       True -0.7485048   0.8363897  -1.41360070 -0.67103350
6         6       True  0.6931959  -0.9668460   0.49041785 -0.09481561
   margin_up     length
1 -1.1279488  0.1735932
2 -0.6965669  0.4715090
3 -0.9122578  0.5517171
4 -0.6102905  0.9527576
5  1.4172048 -0.1586975
6 -0.8691196  0.1506766
```

```r
# Visualiser la distribution des scores Z pour la colonne 'diagonal'
ggplot(as.data.frame(data_z), aes(x=diagonal)) +
  geom_histogram(binwidth=0.5, fill="blue", color="black") +
  theme_minimal() +
  labs(title="Distribution des scores Z pour la colonne 'diagonal'")
```



On fixe à +3 et -3 la limite de z_score pour identifier les outliers

Puis on affiche les lignes pour lesquels il y a au moins une variables considérée comme outlier

```
# Identifier les lignes avec des outliers (Z > 3 ou Z < -3)
z_outliers <- data_z_full %>%
  filter(apply(data_z, 1, function(row) any(row > 3 | row < -3)))

# Afficher les outliers
print(z_outliers)
```

```
   Row_Index is_genuine     diagonal  height_left height_right  margin_low
1          1       True -0.486377368  2.773198409   3.16218582  0.05682068
2        252       True -0.519143291 -2.569722183  -3.37903921 -0.95241302
3        523       True  0.201707025  1.303895246  -3.10264942 -0.94397882
4        665       True  0.300004795 -1.100419020  -0.52301138  0.84532938
5        730       True -3.009353473 -0.632913468  -0.86082112 -0.39808818
6        829       True  3.150640135 -1.601317826   0.06047818  0.45107503
7        843       True  3.052342364 -0.866666244   0.98177748 -0.54972447
8       1023      False  3.052342364  0.001558352   0.33686797  2.34652862
9       1028      False -1.076163990 -0.031834902   2.27159650  3.36249175
10      1030      False  0.005111484  0.869782948   0.33686797  2.31620137
11      1042      False -0.617441061  0.302097635   1.53455706  3.28667361
```

```
12      1083      False -0.682972908 -0.232194424  -0.27733156  1.37605638
13      1111      False -0.748504755  0.969962709   0.45970788  3.15020095
14      1125      False -0.257015903 -0.365767439   1.07390741  3.36249175
15      1143      False -3.009353473  0.669423426   0.92035753  0.58754769
16      1170      False  0.103409254  0.368884143   0.70538769  3.02889192
17      1255      False -2.648928315  0.201917874   1.16603734  3.04405555
18      1278      False  3.445533446  1.871580559   1.19674732  0.84532938
19      1291      False -0.060420363  0.101738113   0.92035753  3.66576433
20      1356      False -0.912334372 -0.465947200  -0.67656126  2.25554685
21      1442      False -1.076163990  1.738007544  -0.33875152  3.15020095
22      1454      False -1.338291377  0.569243665   1.74952690  1.42154727
23      1465      False  0.365536642  0.469063904   1.38100718  3.11987369
24      1485      False  0.398302565 -0.232194424   3.16218582  1.11827469
    margin_up      length
1  -1.1279488  0.17359325
2  -0.8691196  0.62046695
3  -0.1357703  0.86109125
4  -3.8025171  0.99859085
5   0.8995465  0.02463535
6   0.5113027  0.99859085
7  -0.6102905  1.19338195
8  -0.6534287 -1.98056716
9   0.5544409 -1.60244326
10  3.2721473 -2.11806676
11  2.0642778 -1.31598576
12  1.6760340 -3.65347896
13  0.6407173  0.13921835
14 -0.1789085 -2.00348376
15  1.7623104 -1.30452746
16  1.1583756 -0.59411286
17  0.2093353 -1.22431936
18 -0.4377377 -2.02640036
19  0.8995465 -1.12119466
20 -0.5240141 -3.10348056
21 -0.2220467 -0.92640356
22  1.6760340 -3.14931376
23  1.6760340 -1.69410966
24  1.2877902 -0.69723756
```

```r
# Afficher les indices des outliers
z_outliers_indices <- z_outliers$Row_Index
print(z_outliers_indices)
```

```
 [1]    1  252  523  665  730  829  843 1023 1028 1030 1042 1083 1111 1125 1143
[16] 1170 1255 1278 1291 1356 1442 1454 1465 1485
```

```r
# Afficher les lignes du dataframe original contenant des outliers avec le méthode du z-score
data_with_z_outliers <- data[z_outliers_indices, ]

print(data_with_z_outliers)
```

```
     is_genuine diagonal height_left height_right margin_low margin_up length
1          True   171.81      104.86       104.95   4.520000      2.89 112.83
252        True   171.80      103.26       102.82   3.854438      2.95 113.22
523        True   172.02      104.42       102.91   3.860000      3.12 113.43
665        True   172.05      103.70       103.75   5.040000      2.27 113.55
730        True   171.04      103.84       103.64   4.220000      3.36 112.70
829        True   172.92      103.55       103.94   4.780000      3.27 113.55
843        True   172.89      103.77       104.24   4.120000      3.01 113.72
1023      False   172.89      104.03       104.03   6.030000      3.00 110.95
1028      False   171.63      104.02       104.66   6.700000      3.28 111.28
1030      False   171.96      104.29       104.03   6.010000      3.91 110.83
1042      False   171.77      104.12       104.42   6.650000      3.63 111.53
1083      False   171.75      103.96       103.83   5.390000      3.54 109.49
1111      False   171.73      104.32       104.07   6.560000      3.30 112.80
1125      False   171.88      103.92       104.27   6.700000      3.11 110.93
1143      False   171.04      104.23       104.22   4.870000      3.56 111.54
1170      False   171.99      104.14       104.15   6.480000      3.42 112.16
1255      False   171.15      104.09       104.30   6.490000      3.20 111.61
1278      False   173.01      104.59       104.31   5.040000      3.05 110.91
1291      False   171.94      104.06       104.22   6.900000      3.36 111.70
1356      False   171.68      103.89       103.70   5.970000      3.03 109.97
1442      False   171.63      104.55       103.81   6.560000      3.10 111.87
1454      False   171.55      104.20       104.49   5.420000      3.54 109.93
1465      False   172.07      104.17       104.37   6.540000      3.54 111.20
1485      False   172.08      103.96       104.95   5.220000      3.45 112.07
```

```r
library(dplyr)

# Comptage du nombre de lignes par groupe dans la colonne is_genuine
z_count_by_is_genuine <- z_outliers %>%
  group_by(is_genuine) %>%
  summarise(count = n())
```

```r
# Affichage du résultat
print(z_count_by_is_genuine)
```

```
# A tibble: 2 x 2
  is_genuine count
  <chr>      <int>
1 False         17
2 True           7
```

Il y a 24 lignes d'outliers avec la méthode du Z_score
17 lignes concernent des faux billets
7 lignes concernent de vrais billets

2- Suppression des lignes d'outliers

Je vais commencer mon analyse en supprimant les outliers identifiés par la méthode du Z_Score
Je vais donc supprimer 24 lignes du dataframe original "data"
je le nommerai "data_without_ouliers_z"

```r
data_without_ouliers_z <- data[-z_outliers_indices, ]
summary(data_without_ouliers_z)
```

```
  is_genuine           diagonal      height_left     height_right
 Length:1476        Min.   :171.1   Min.   :103.1   Min.   :103.0
 Class :character   1st Qu.:171.8   1st Qu.:103.8   1st Qu.:103.7
 Mode  :character   Median :172.0   Median :104.0   Median :103.9
                    Mean   :172.0   Mean   :104.0   Mean   :103.9
                    3rd Qu.:172.2   3rd Qu.:104.2   3rd Qu.:104.1
                    Max.   :172.8   Max.   :104.9   Max.   :104.9
   margin_low      margin_up        length
 Min.   :2.980   Min.   :2.56   Min.   :110.2
 1st Qu.:4.018   1st Qu.:2.99   1st Qu.:112.1
 Median :4.300   Median :3.14   Median :113.0
 Mean   :4.465   Mean   :3.15   Mean   :112.7
 3rd Qu.:4.840   3rd Qu.:3.31   3rd Qu.:113.3
 Max.   :6.380   Max.   :3.81   Max.   :114.4
```

3- ACP

```r
if (!require(FactoMineR)) install.packages("FactoMineR")
```

```
Le chargement a nécessité le package : FactoMineR
```

```r
if (!require(factoextra)) install.packages("factoextra")
```

```
Le chargement a nécessité le package : factoextra
```

```
Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```r
library(FactoMineR)
library(factoextra)
```

```r
# Extraire les colonnes numériques
datanum_without_ouliers_z <- Filter(is.numeric, data_without_ouliers_z)
```

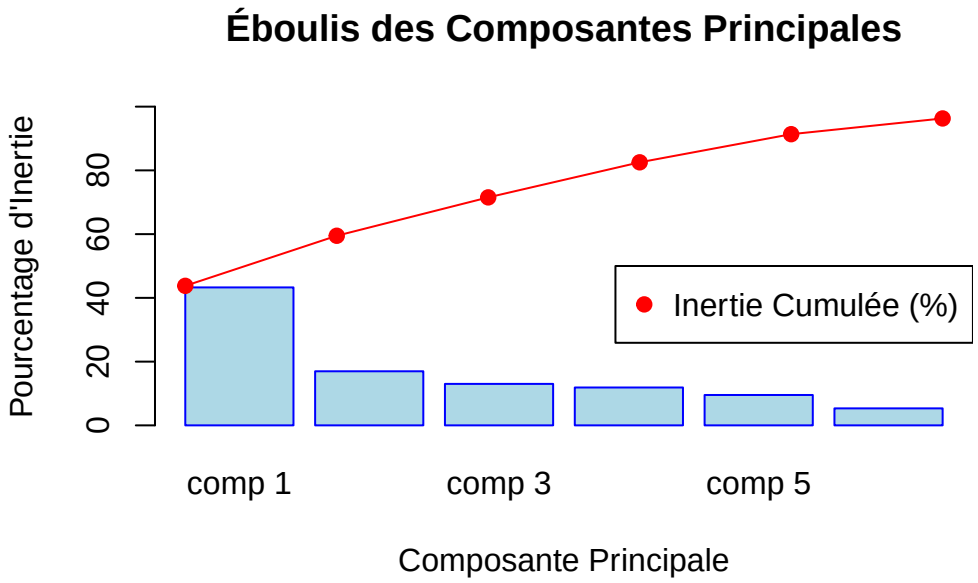
```r
# Réaliser l'ACP sans spécifier ncp
resultat_acp <- PCA(datanum_without_ouliers_z, scale.unit = TRUE, graph = FALSE)

# Extraire les valeurs propres
valeurs_propres <- resultat_acp$eig[,1]
```

```r
# Calculer les pourcentages d'inertie expliquée par chaque composante
pourcentage_inertie <- 100 * valeurs_propres / sum(valeurs_propres)
# Calculer l'inertie cumulée
cumulative_inertia <- cumsum(pourcentage_inertie)
```

a- Eboulis des Composantes Principales

```r
# Créer un graphique avec un seul appel
par(mfrow = c(1, 1)) # Assurez-vous d'avoir une seule fenêtre graphique

# Créer un plot avec des barres pour les pourcentages d'inertie
barplot(pourcentage_inertie, main = "Éboulis des Composantes Principales",
        xlab = "Composante Principale", ylab = "Pourcentage d'Inertie",
        col = "lightblue", border = "blue", ylim = c(0, 100))

# Ajouter la courbe cumulée en superposition sur le même graphique
```

```r
# Recréer le graphique avec la courbe cumulée en utilisant `plot` pour ne pas effacer les ba
par(new = TRUE) # Permet de superposer le nouveau graphique sur le précédent
plot(cumulative_inertia, type = "o", col = "red", pch = 19,
     ylim = c(0, 100), xlab = "", ylab = "", axes = FALSE)

# Ajouter une légende
legend("topright", legend = c("Inertie Cumulée (%)"), col = "red", pch = 19, inset = c(0, 0.
```



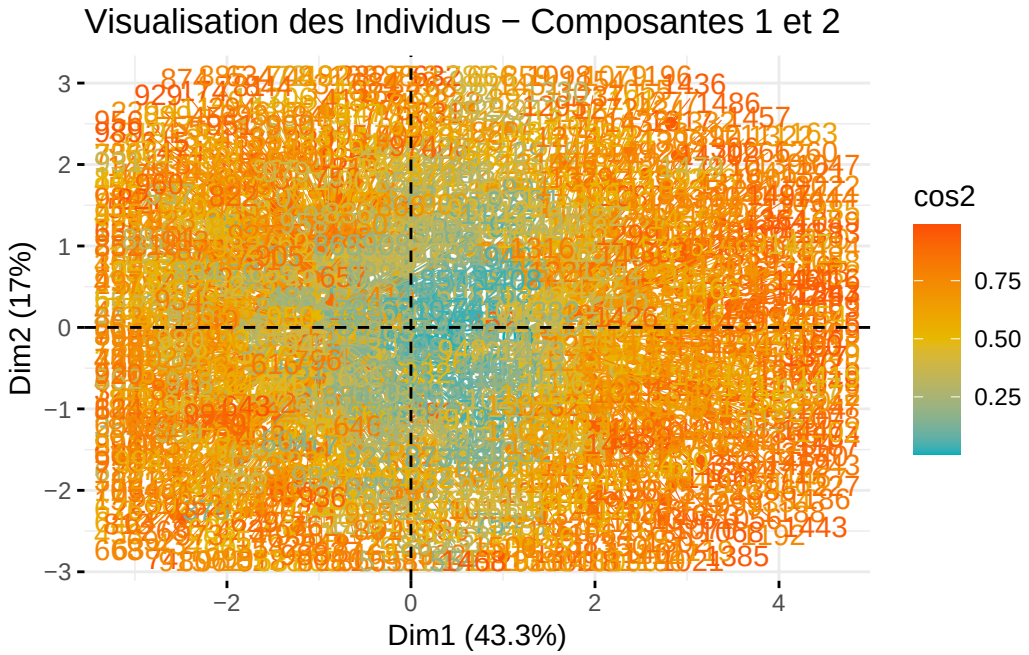
On remarque avec l'ébouli qu'il est nécessaire d'avoir 5 composantes pour expliquer 90% de l'inertie des 6 variables. Il n'est pas très utile de réduire la dimensionalité dans notre si nous perdons en précision pour expliquer la variance.

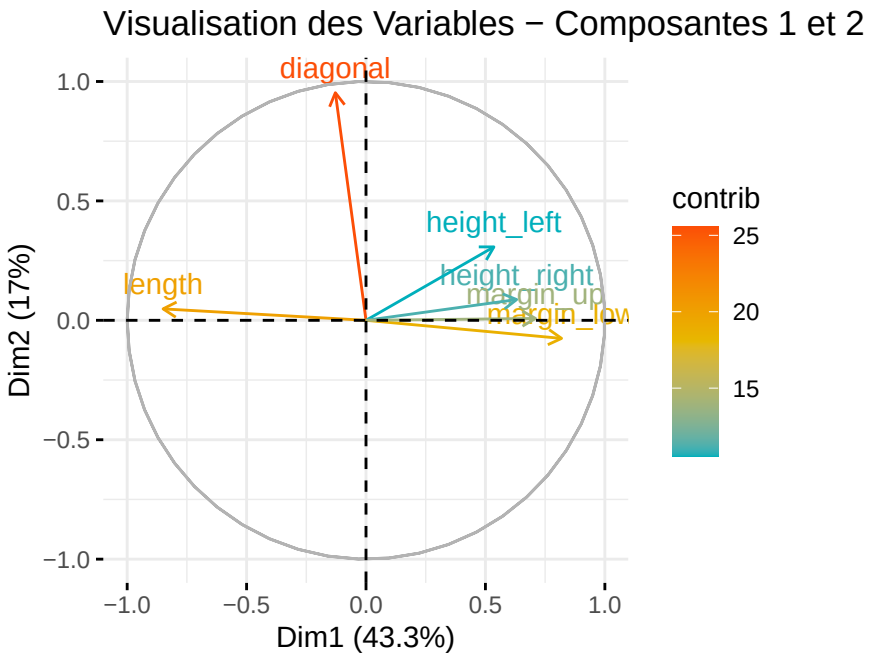
Nous pouvons cependant réaliser l'ACP pour afficher le cercle des corrélations, notamment pour les 2 premières dimensions qui expliquent 60% de la variance totale.

```r
# on choisit 2 composantes principales qui représentent 60% de la variance
resultat_acp_optimal <- PCA(datanum_without_ouliers_z, scale.unit = TRUE, ncp = 2, graph = FA
```

```r
# Visualiser les individus et les variables pour les premières et dernières combinaisons de c
pairs_to_plot <- list(c(1, 2))  # Choisir les paires de composantes à afficher

# Visualiser les individus et les variables pour les différentes combinaisons de composantes
for (pair in pairs_to_plot) {
```

```r
# Visualiser les individus
  print(fviz_pca_ind(resultat_acp_optimal,
                     axes = pair,
                     col.ind = "cos2",
                     gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
                     repel = TRUE) +
          ggtitle(paste("Visualisation des Individus - Composantes", pair[1], "et", pair[2])))

  # Visualiser les variables
  print(fviz_pca_var(resultat_acp_optimal,
                     axes = pair,
                     col.var = "contrib",
                     gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07")) +
          ggtitle(paste("Visualisation des Variables - Composantes", pair[1], "et", pair[2])))
}
```





Axes principaux :
Dim1 (43.3%) : Cet axe explique 43.3% de la variance totale des données.
Dim2 (17%) : Cet axe explique 17% de la variance totale des données.

Dim1 On observe :
- une forte contribution positibe de la marge basse (margin_low)
- une forte contribution négative de la longueur (length)
une forte valeur de Dim1 indiquera un longueur de billet faible avec une marge basse élevée

Dim2 On observe :
- une forte contribution positibe de la diagonale Une forte valeur de Dim2 indiquera une valeur de diagonale élevée

```
# Extraire les scores des individus
scores <- resultat_acp_optimal$ind$coord
```

```
head(scores)
```

```
       Dim.1      Dim.2
2 -2.0300617 -2.2233690
3 -0.9417121  2.6761752
4 -1.3893330 -1.8605904
5  0.1704404 -0.5343113
6 -0.6704944  0.4185489
```

```
7   0.2156576   1.3324354
```

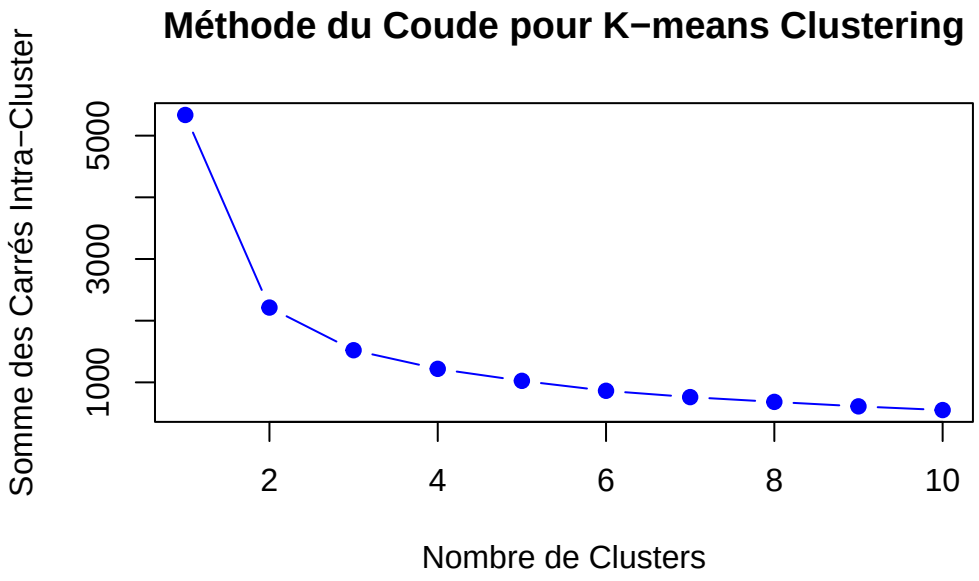
4- Clustering avec K-means

a- Méthode du coude

```r
# Définir une plage de nombres de clusters à tester
k_values <- 1:10  # Tester de 1 à 10 clusters
wss <- numeric(length(k_values))  # Pour stocker la somme des carrés intra-cluster

# Calculer l'inertie pour chaque nombre de clusters
for (k in k_values) {
  kmeans_result <- kmeans(scores, centers = k, nstart = 25)  # nstart pour la reproductibili
  wss[k] <- kmeans_result$tot.withinss
}
```

```
Warning: pas de convergence en 10 itérations
```

```r
# Tracer la méthode du coude
plot(k_values, wss, type = "b", pch = 19, col = "blue",
     xlab = "Nombre de Clusters", ylab = "Somme des Carrés Intra-Cluster",
     main = "Méthode du Coude pour K-means Clustering")
```

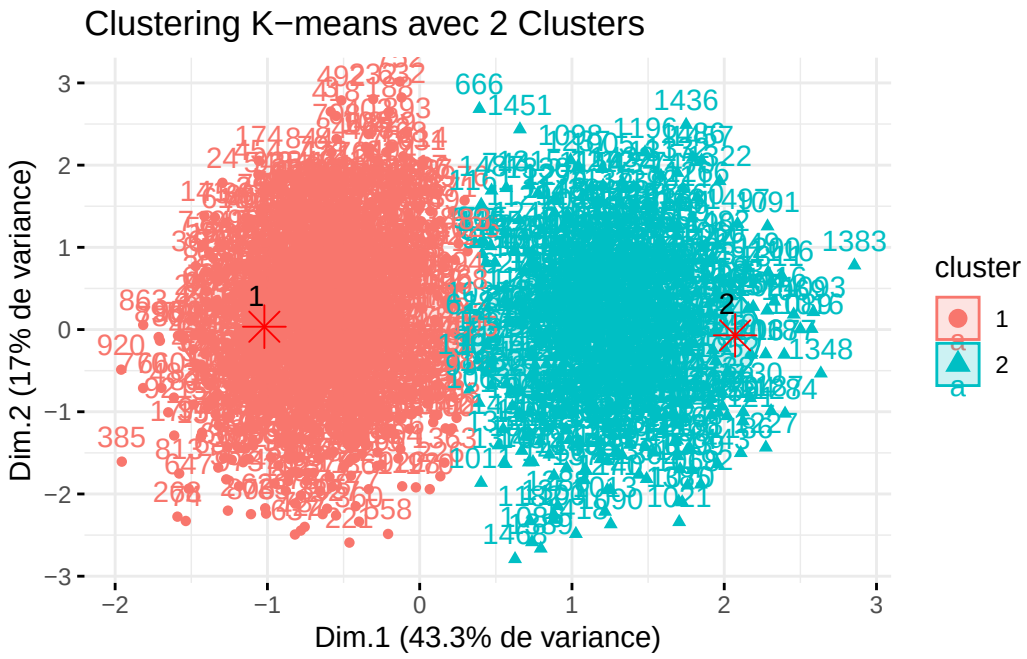


La méthode du coude indique que l'ajout de clusters au-delà de 2 n'améliore plus significativement la dispersion intra-cluster, suggérant que 2 clusters est le nombre optimal.
En utilisant 2 clusters avec K-means, on obtient une séparation efficace des données tout en évitant une complexité inutile.

b- Test avec 2 Clusters

```
# Appliquer K-means clustering avec 2 clusters
kmeans_result <- kmeans(scores, centers = 2, nstart = 25)  # nstart pour la reproductibilité
```

```
# Installer et charger le package factoextra si ce n'est pas déjà fait
if (!require(factoextra)) install.packages("factoextra")
library(factoextra)

# Installer et charger le package ggplot2 si ce n'est pas déjà fait
if (!require(ggplot2)) install.packages("ggplot2")
library(ggplot2)

# Visualiser les clusters dans les deux premières composantes principales
plot_clusters <- fviz_cluster(kmeans_result, data = scores,
                              ellipse.type = "euclid",  # Ajouter des ellipses autour des clu
                              ggtheme = theme_minimal())

# Modifier les étiquettes des axes pour refléter la variance expliquée
plot_clusters <- plot_clusters +
  labs(x = "Dim.1 (43.3% de variance)", y = "Dim.2 (17% de variance)") +
  ggtitle("Clustering K-means avec 2 Clusters")

# Ajouter les centres des clusters au graphique
centroids <- kmeans_result$centers

plot_clusters <- plot_clusters +
  geom_point(data = as.data.frame(centroids), aes(x = Dim.1, y = Dim.2),
             color = "red", size = 5, shape = 8) +  # Points rouges pour les centres
  geom_text(data = as.data.frame(centroids), aes(x = Dim.1, y = Dim.2, label = rownames(centi
            color = "black", vjust = -1, hjust = 1)  # Étiquettes pour les centres

# Afficher le graphique final
print(plot_clusters)
```



Centroids :

```
# Centres des clusters
print(kmeans_result$centers)
```

```
      Dim.1       Dim.2
1 -1.020025  0.03457443
2  2.071468 -0.07021379
```

```
# Taille de chaque cluster
print(kmeans_result$size)
```

```
[1] 989 487
```

```
# Ajouter les résultats du clustering à un DataFrame
data_with_clusters <- data.frame(data_without_ouliers_z, Cluster = kmeans_result$cluster)

# Afficher les premières lignes du DataFrame avec les clusters
head(data_with_clusters)
```

```
  is_genuine diagonal height_left height_right margin_low margin_up length
```

```
2       True   171.46      103.36       103.66       3.77      2.99 113.09
3       True   172.69      104.48       103.50       4.40      2.94 113.16
4       True   171.36      103.91       103.94       3.62      3.01 113.51
5       True   171.73      104.28       103.46       4.04      3.48 112.54
6       True   172.17      103.74       104.08       4.42      2.95 112.81
7       True   172.34      104.18       103.85       4.58      3.26 112.81
  Cluster
2       1
3       1
4       1
5       1
6       1
7       1
```

```r
count_values  <- data_with_clusters %>%
  group_by(Cluster, is_genuine) %>%
  count()

print(count_values)
```

```
# A tibble: 4 x 3
# Groups:   Cluster, is_genuine [4]
  Cluster is_genuine     n
    <int> <chr>      <int>
1       1 False         12
2       1 True         977
3       2 False        471
4       2 True          16
```

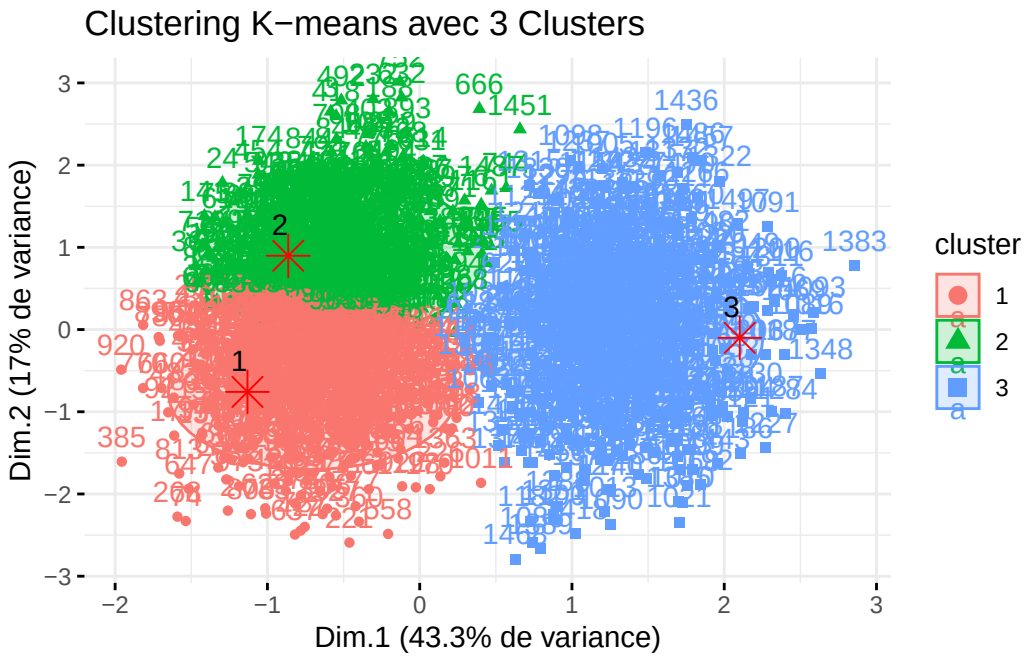
La répartition des “False” et “True” de chaque cluster nous indique que : - Le cluster 1 correspond aux faux billets avec 96.7 % de faux billets - Le cluster 2 correspond aux vrais billets avec 98.8 % de vrais billets

Naturellement la méthode des K-means a séparé notre jeu de donnée en 2 groupes avec une efficacité globale ou accuracy de 98.10 %.

c- Test avec 3 Clusters

```r
# Appliquer K-means clustering avec 3 clusters
kmeans_result_2 <- kmeans(scores, centers = 3, nstart = 36)  # nstart pour la reproductibili
```

```r
# Installer et charger le package factoextra si ce n'est pas déjà fait
if (!require(factoextra)) install.packages("factoextra")
library(factoextra)

# Installer et charger le package ggplot2 si ce n'est pas déjà fait
if (!require(ggplot2)) install.packages("ggplot2")
library(ggplot2)

# Visualiser les clusters dans les deux premières composantes principales
plot_clusters_2 <- fviz_cluster(kmeans_result_2, data = scores,
                                ellipse.type = "euclid",  # Ajouter des ellipses autour des clu
                                ggtheme = theme_minimal())

# Modifier les étiquettes des axes pour refléter la variance expliquée
plot_clusters_2 <- plot_clusters_2 +
  labs(x = "Dim.1 (43.3% de variance)", y = "Dim.2 (17% de variance)") +
  ggtitle("Clustering K-means avec 3 Clusters")

# Ajouter les centres des clusters au graphique
centroids_2 <- kmeans_result_2$centers

plot_clusters_2 <- plot_clusters_2 +
  geom_point(data = as.data.frame(centroids_2), aes(x = Dim.1, y = Dim.2),
             color = "red", size = 5, shape = 8) +  # Points rouges pour les centres
  geom_text(data = as.data.frame(centroids_2), aes(x = Dim.1, y = Dim.2, label = rownames(cen
            color = "black", vjust = -1, hjust = 1)  # Étiquettes pour les centres

# Afficher le graphique final
print(plot_clusters_2)
```




```
# Centres des clusters
print(kmeans_result_2$centers)
```

```
       Dim.1      Dim.2
1 -1.1308693 -0.7583317
2 -0.8632331  0.8974863
3  2.1019548 -0.1009489
```

```
# Taille de chaque cluster
print(kmeans_result_2$size)
```

```
[1] 513 487 476
```

```
# Ajouter les résultats du clustering à un DataFrame
data_with_clusters_2 <- data.frame(data_without_ouliers_z, Cluster = kmeans_result_2$cluster)

# Afficher les premières lignes du DataFrame avec les clusters
head(data_with_clusters_2)
```

```
  is_genuine diagonal height_left height_right margin_low margin_up length
2       True   171.46      103.36       103.66       3.77      2.99 113.09
```

```
3       True   172.69      104.48       103.50       4.40      2.94 113.16
4       True   171.36      103.91       103.94       3.62      3.01 113.51
5       True   171.73      104.28       103.46       4.04      3.48 112.54
6       True   172.17      103.74       104.08       4.42      2.95 112.81
7       True   172.34      104.18       103.85       4.58      3.26 112.81
  Cluster
2       1
3       2
4       1
5       1
6       2
7       2
```

```r
count_values  <- data_with_clusters_2 %>%
  group_by(Cluster, is_genuine) %>%
  count()

print(count_values)
```

```
# A tibble: 6 x 3
# Groups:   Cluster, is_genuine [6]
  Cluster is_genuine     n
    <int> <chr>      <int>
1       1 False          5
2       1 True         508
3       2 False         11
4       2 True         476
5       3 False        467
6       3 True           9
```

Ici nous avons : - cluster 1 avec 98.1 % de faux billets - cluster 2 avec 97.7 % de vrais billets - cluster 3 avec 99 % de vrais billets pour une efficacité globale de 98.3 %

Il y a une meilleure efficacité au global, le cluster 3 atteint jusqu'à 99% de vrais billets mais cela ne nous aide pas énormément dans notre objectif de détecter les faux billets.

Le but était de vérifier comment se comportait le modèle avec 3 clusters. L'idéal aurait été un clustering avec 100 % de vrai dans un cluster 100 % de faux dans un autre cluster et les douteux dans un 3ème cluster

VII- Régression Logistique

1- Remplacement des valeurs de “is_genuine” par “0” et “1”

Nous allons commencer par remplacer “False” par “0” et “True” par “1” afin que “is_genuine” contienne des valeurs numériques binaires.

```
# Convertir 'is_genuine' en binaire : True -> 1 et False -> 0
data$is_genuine <- ifelse(data$is_genuine == "True", 1, 0)

# Vérifiez les premiers enregistrements pour confirmer la transformation
head(data$is_genuine)
```

```
[1] 1 1 1 1 1 1
```

2- Régréssion logistique complète (toutes les variables)

```
rl_model <- glm(is_genuine~diagonal+height_left+height_right+margin_low+margin_up+length,
family="binomial",data=data)
summary(rl_model)
```

```
Call:
glm(formula = is_genuine ~ diagonal + height_left + height_right +
    margin_low + margin_up + length, family = "binomial", data = data)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  -213.52163  244.09440  -0.875   0.3817
diagonal        0.09636    1.09649   0.088   0.9300
height_left    -1.66297    1.10714  -1.502   0.1331
height_right   -2.33619    1.07197  -2.179   0.0293 *
margin_low     -5.88384    0.96131  -6.121 9.32e-10 ***
margin_up     -10.18443    2.13235  -4.776 1.79e-06 ***
length          5.97673    0.86444   6.914 4.71e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)
```

```
    Null deviance: 1909.543  on 1499  degrees of freedom
Residual deviance:   83.222  on 1493  degrees of freedom
AIC: 97.222

Number of Fisher Scoring iterations: 10
```

Certaines des variables obtenues ont des p-valeurs qui sont inférieures au niveau de test de 5 %, ce qui nous indique qu'elles sont bien significatives. Certaines autres ne sont pas en dessous de ce seuil.

On peut donc passer sur une procédure de sélection en retirant les variables non significatives au fur et à mesure, mais nous pouvons aussi sélectionner automatiquement un modèle avec une commande telle que stepAIC , qui sélectionne de manière automatique un modèle en se basant sur le critère AIC.

3- Sélection du modèle (méthode AIC)

```
library(MASS)
```

```
Attachement du package : 'MASS'

L'objet suivant est masqué depuis 'package:dplyr':

    select
```

```
stepAIC(rl_model, direction = "both")
```

```
Start:  AIC=97.22
is_genuine ~ diagonal + height_left + height_right + margin_low +
    margin_up + length

               Df Deviance    AIC
- diagonal      1   83.230  95.23
<none>              83.222  97.22
- height_left   1   85.533  97.53
- height_right  1   88.703 100.70
- margin_up     1  125.430 137.43
- margin_low    1  205.785 217.79
```

```
- length        1  308.071 320.07

Step:  AIC=95.23
is_genuine ~ height_left + height_right + margin_low + margin_up +
    length

               Df Deviance    AIC
<none>              83.230  95.23
- height_left   1   85.544  95.54
+ diagonal      1   83.222  97.22
- height_right  1   88.869  98.87
- margin_up     1  126.364 136.36
- margin_low    1  211.175 221.17
- length        1  308.956 318.96



Call:  glm(formula = is_genuine ~ height_left + height_right + margin_low +
    margin_up + length, family = "binomial", data = data)

Coefficients:
 (Intercept)   height_left  height_right    margin_low     margin_up
    -197.260        -1.663        -2.344        -5.910       -10.203
      length
       5.988

Degrees of Freedom: 1499 Total (i.e. Null);  1494 Residual
Null Deviance:      1910
Residual Deviance: 83.23    AIC: 95.23
```

Critère AIC initial 97.22 Après avoir retiré “diagonal”, le modèle à le meilleur AIC avec 95.23. Les autres étapes de suppression de variable augmentent l’AIC ce qui indique qu’elles sont importantes pour le modèle.

La procédure de sélection de modèle nous a permis de déterminer que la variable “diagonal” n’était pas nécessaire pour le modèle final. Les variables restantes (“height_right”, “height_right”, “margin_low”, “margin_up”, et “length”) sont significatives et ont un impact important sur le modèle.
Nous avons réussi à réduire l’AIC, indiquant que le modèle final est plus parcimonieux tout en conservant une bonne qualité d’ajustement.

a- Analyse des coefficients

3 variables ont une valeur absolue élevée ce qui indique qu'elles ont plus d'importance et d'influence dans la prédiction.

1 correspondant à la valeur "vrai" de is_genuine, un coefficient positif indique que si la valeur augmente, la probabilité de "vrai" billet augmente également inversement un coefficient négatif indique que plus la valeur de la variable augmente plus la probabilité d'un "faux" billet augmente également.

- "margin_up" a un coeff négatif de -10.203 ce qui indique que plus la marge haute augmente, plus il y a de chance qu'il s'agisse d'un faux billet.

- "margin_low" avec un coeff négatif également de -5.910 a également une influence importante et indique que l'augmentation de la marge basse augmente les probabilités qu'il s'agisse d'un faux billet.

- "length" a un coeff positif de 5.988 qui indique à l'inverse que l'augmentation de la longueur du billet augmente la probabilité qu'il s'agisse d'un vrai billet.

b- Odds Ratios

Calculons les Odds Ratios pour indiquer comment les changements dans les variables indépendantes affectent la probabilité de vrais ou faux billets :

```
rl_model_1 <- glm(is_genuine ~ height_left + height_right + margin_low +
    margin_up + length,
family="binomial",data=data)
summary(rl_model_1)
```

```
Call:
glm(formula = is_genuine ~ height_left + height_right + margin_low +
    margin_up + length, family = "binomial", data = data)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -197.2599   158.9321  -1.241   0.2145
height_left   -1.6626     1.1062  -1.503   0.1328
height_right  -2.3440     1.0661  -2.199   0.0279 *
margin_low    -5.9103     0.9168  -6.447 1.14e-10 ***
margin_up    -10.2028     2.1251  -4.801 1.58e-06 ***
length         5.9879     0.8571   6.986 2.83e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1909.54  on 1499  degrees of freedom
Residual deviance:   83.23  on 1494  degrees of freedom
AIC: 95.23

Number of Fisher Scoring iterations: 10
```

```r
# Récupérer les coefficients du modèle final
coefficients <- coef(rl_model_1)

# Calculer les odds ratios en exponentiant les coefficients
odds_ratios <- exp(coefficients)

# Afficher les odds ratios sans exposant
formatted_odds_ratios <- formatC(odds_ratios, format = "f", digits = 5)


# Imprimer les odds ratios formatés
formatted_odds_ratios
```

```
 (Intercept)  height_left height_right   margin_low    margin_up       length
   "0.00000"    "0.18964"    "0.09595"    "0.00271"    "0.00004"  "398.56936"
```

height_left: 0.18964 Une augmentation d'une unité de height_left diminue les chances que le billet soit authentique d'environ 81% (puisque 0.18964<1)

height_right: 0.09595 Une augmentation d'une unité de height_right diminue les chances que le billet soit authentique d'environ 90.5%.

margin_low: 0.00271 Une augmentation d'une unité de margin_low diminue les chances que le billet soit authentique d'environ 99.7%.

margin_up: 0.00004 Une augmentation d'une unité de margin_up diminue fortement les chances que le billet soit authentique.

length: 398.56936 Une augmentation d'une unité de length augmente de manière significative (par 398 fois) les chances que le billet soit authentique.

4- Evaluation des performances du modèle

a- Métrique de performance

Matrice de confusion

La matrice de confusion permet de comparer les prédictions du modèle avec les valeurs réelles.

```
# Prédictions du modèle
predictions <- predict(rl_model_1, type = "response")

# Convertir les probabilités en classes (0 ou 1) en utilisant un seuil de 0.5
classes <- ifelse(predictions > 0.5, "True", "False")

# Créer la matrice de confusion
conf_matrix <- table(Predicted = classes, Actual = data$is_genuine)
print(conf_matrix)
```

```
         Actual
Predicted   0   1
    False 491   4
    True    9 996
```

Vrais Négatifs (VN) : 491
Faux Négatifs (FN) : 4
Faux Positifs (FP) : 9
Vrais Positifs (VP) : 996

Pour une probabilité supérieur à 0.5, le billet est prédit comme bon et en dessous de 0.5 le billet est prédit comme faux. La matrice de confusion montre que le modèle a une très bonne performance en classant les cas True et False correctement avec peu d'erreurs avec un seuil à 0.5.

Mesures de performance

Précision : Proportion des prédictions de vrais positifs parmi toutes les prédictions de positifs. VP/(VP+FP)
Sensibilité (Recall) : Proportion des vrais positifs parmi tous les cas positifs réels. VP/(VP+FN)
Spécificité : Proportion des vrais négatifs parmi tous les cas négatifs réels. VN/(VN+FP)
F1-Score : Moyenne harmonique de la précision et de la sensibilité. 2x((precision x recall)/(precision + recall))

```r
# Calculer les mesures de performance
precision_rl_model_1 <- conf_matrix[2, 2] / (conf_matrix[2, 2] + conf_matrix[2, 1])
recall_rl_model_1 <- conf_matrix[2, 2] / (conf_matrix[2, 2] + conf_matrix[1, 2])
specificity_rl_model_1 <- conf_matrix[1, 1] / (conf_matrix[1, 1] + conf_matrix[2, 1])
f1_score_rl_model_1 <- 2 * (precision_rl_model_1 * recall_rl_model_1) / (precision_rl_model_1
# Calculer Accuracy
accuracy_rl_model_1 <- (conf_matrix[1, 1] + conf_matrix[2, 2]) / sum(conf_matrix)      # (TP

# Afficher les résultats
cat("Précision: ", precision_rl_model_1, "\n")
```

```
Précision:  0.9910448
```

```r
cat("Sensibilité (Recall): ", recall_rl_model_1, "\n")
```

```
Sensibilité (Recall):  0.996
```

```r
cat("Spécificité: ", specificity_rl_model_1, "\n")
```

```
Spécificité:  0.982
```

```r
cat("F1-Score: ", f1_score_rl_model_1, "\n")
```

```
F1-Score:  0.9935162
```

```r
cat("Accuracy: ", accuracy_rl_model_1, "\n")
```

```
Accuracy:  0.9913333
```

Précision (Precision) : 0.991 Cela signifie que parmi toutes les instances que le modèle a classées comme True, 99.1% sont effectivement True. C'est un excellent résultat, indiquant que le modèle est très précis dans ses prédictions positives.

Sensibilité (Recall) : 0.996 La sensibilité mesure la proportion des véritables True correctement identifiés par le modèle. Avec une sensibilité de 99.6%, votre modèle détecte presque tous les cas positifs.

Spécificité : 0.982 La spécificité mesure la proportion des véritables False correctement identifiés par le modèle. Avec une spécificité de 98.2%, le modèle est aussi très efficace pour identifier les cas négatifs.

F1-Score : 0.993 L'F1-Score est la moyenne harmonique de la précision et du rappel. Un F1-Score de 99.3% indique un excellent équilibre entre précision et rappel.

Mon objectif est d'obtenir un modèle avec une spécificité proche de 100% pour minimiser au maximum le risque de faux positif, il est donc important que tous les vrais négatifs soient identifier même si cela déteriore un peu la précision.

Courbe ROC et AUC

La courbe ROC (Receiver Operating Characteristic) est utile pour visualiser la performance du modèle à différents seuils de classification. L'AUC (Area Under the Curve) mesure la capacité globale du modèle à discriminer entre les classes.

```
# Installer et charger le package pROC si ce n'est pas déjà fait
if (!require(pROC)) install.packages("pROC")
```

```
Le chargement a nécessité le package : pROC
```

```
Type 'citation("pROC")' for a citation.
```

```

Attachement du package : 'pROC'
```

```
Les objets suivants sont masqués depuis 'package:stats':
```

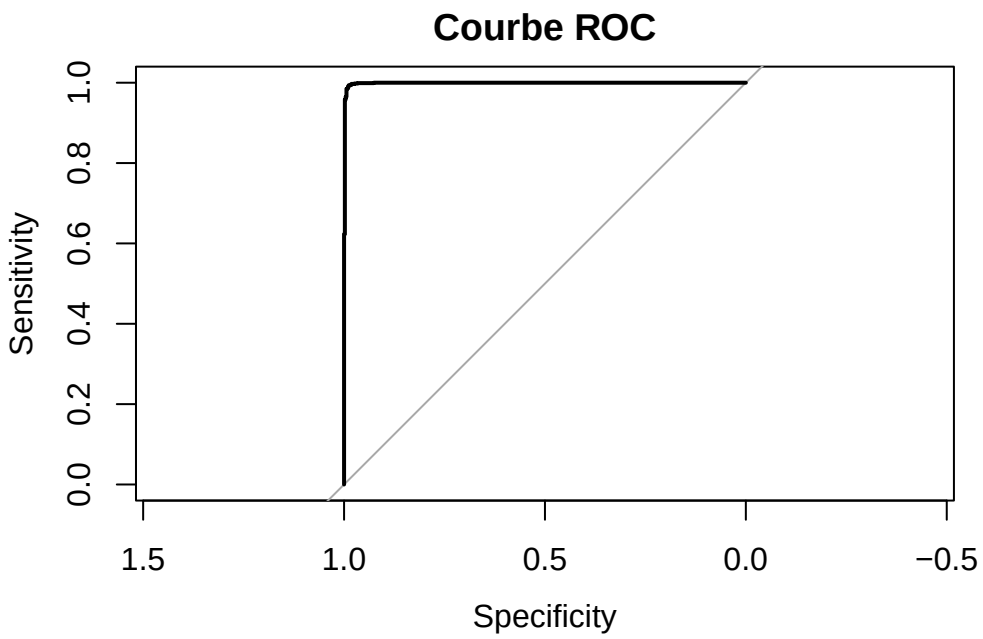
```
    cov, smooth, var
```

```
library(pROC)

# Calculer la courbe ROC
roc_curve <- roc(data$is_genuine, predictions)
```

```
Setting levels: control = 0, case = 1
```

```
Setting direction: controls < cases
```

```
# Tracer la courbe ROC
plot(roc_curve, main = "Courbe ROC")
```




```
# Calculer l'AUC
auc_value <- auc(roc_curve)
cat("AUC: ", auc_value, "\n")
```

```
AUC:  0.9989
```

AUC : 0.998894 L'AUC mesure la capacité globale du modèle à discriminer entre les classes. Une AUC proche de 1 indique que le modèle a une très bonne performance de classification, capable de distinguer presque parfaitement entre les classes True et False.

b- Validation croisée K-Fold :

La validation croisée K-Fold permet de tester la robustesse du modèle en le validant sur différentes sous-parties du dataset.

```
# Installer et charger le package cvms si ce n'est pas déjà fait
if (!require(cvms)) install.packages("cvms")
```

```
Le chargement a nécessité le package : cvms
```

```r
if (!require(caret)) install.packages("caret")
```

```
Le chargement a nécessité le package : caret
```

```
Le chargement a nécessité le package : lattice
```

```

Attachement du package : 'caret'
```

```
L'objet suivant est masqué depuis 'package:purrr':
```

```
    lift
```

```r
library(cvms)
library(caret)

# Appliquer la validation croisée K-Fold
set.seed(123)  # Pour la reproductibilité
folds <- createFolds(data$is_genuine, k = 10)

# Calculer les performances pour chaque fold
cv_results <- lapply(folds, function(fold) {
  train_data <- data[-fold, ]
  test_data <- data[fold, ]

  # Ajuster le modèle sur les données d'entraînement
  model <- glm(is_genuine ~ height_left + height_right + margin_low + margin_up + length,
               family = "binomial", data = train_data)

  # Prédictions sur les données de test
  test_predictions <- predict(model, newdata = test_data, type = "response")
  test_classes <- ifelse(test_predictions > 0.5, 1, 0)

  # Calculer la matrice de confusion et les mesures de performance
  confusion_test <- table(Predicted = test_classes, Actual = test_data$is_genuine)
  precision_test <- confusion_test[2, 2] / (confusion_test[2, 2] + confusion_test[1, 2])
  recall_test <- confusion_test[2, 2] / (confusion_test[2, 2] + confusion_test[2, 1])
  specificity_test <- confusion_test[1, 1] / (confusion_test[1, 1] + confusion_test[1, 2])
  f1_score_test <- 2 * (precision_test * recall_test) / (precision_test + recall_test)
```

```
  return(c(precision = precision_test, recall = recall_test, specificity = specificity_test,
})
```

```
Warning: glm.fit: des probabilités ont été ajustées numériquement à 0 ou 1
```

```
# Moyenne des résultats de validation croisée
cv_results <- do.call(rbind, cv_results)
mean_cv_results <- colMeans(cv_results)
print(mean_cv_results)
```

```
  precision      recall specificity    f1_score
  0.9939991   0.9910379   0.9877148   0.9925019
```

Le lift est une mesure souvent utilisée pour évaluer l'amélioration de la prédiction par rapport à un modèle de base (par exemple, prédire la classe majoritaire). Les valeurs élevées du lift indiquent que le modèle offre des améliorations significatives par rapport à des méthodes naïves.

Conclusion Ces résultats montrent que le modèle de régression logistique est très performant avec des scores de précision, de rappel, de spécificité et de F1-Score très élevés, ainsi qu'une AUC presque parfaite. Cela suggère que le modèle est bien ajusté aux données et est capable de faire des prédictions avec une grande précision.

Prochaines Étapes Validation Externe : Si possible, testez le modèle sur un jeu de données complètement indépendant pour vérifier sa généralisation. Analyse des Résidus : Examinez les résidus pour identifier tout modèle non détecté ou problème potentiel. Exploration de Modèles Alternatifs : Bien que le modèle de régression logistique semble excellent, vous pouvez explorer d'autres modèles pour comparer les performances. Implémentation et Déploiement : Si le modèle répond à vos attentes, vous pouvez envisager de l'implémenter dans un environnement de production. Ces étapes vous aideront à assurer que votre modèle est non seulement performant mais aussi robuste et fiable dans des conditions réelles.

5- Amélioration du modèle

Nous allons ajuster le seuil pour réduire la probabilité de classer un faux billet parmis les vrais, c'est à dire minimiser le risque de faux positifs.

```r
head(predictions)
```

```
        1         2         3         4         5         6
0.8595325 0.9999954 0.9996703 0.9999991 0.7939694 0.9962111
```

a- Ajustement du seuil pour réduire le taux de faux positifs

La mesure de performance concernant la détection des faux billets est la spécificité dont le calcul est le suivant : VN/(VN+FP) Cela nous donne la proportion de vrais négatifs détectés parmi les négatifs.

Création d'une fonction pour calculer la spécificité

```r
# Fonction pour calculer la spécificité
calculate_specificity <- function(pred_probs, actual_classes, threshold) {
  predicted_classes <- ifelse(pred_probs > threshold, 1, 0)
  confusion <- table(predicted_classes, actual_classes)
  # Confusion peut avoir des noms différents selon les résultats, ajustez si nécessaire
  VN <- confusion["0", "0"]
  FP <- confusion["1", "0"]
  specificity <- VN / (VN + FP)
  return(specificity)
}
```

Plage de seuils à tester

```r
thresholds <- seq(0.5, 0.99, by = 0.01)
specificities <- sapply(thresholds, function(t) {
  calculate_specificity(predictions, data$is_genuine, t)
})
```

Recherche du seuil qui offre le maximum de spécificité

```r
best_threshold <- thresholds[which.max(specificities)]
best_specificity <- max(specificities)

cat("Meilleur seuil:", best_threshold, "\n")
```

```
Meilleur seuil: 0.97
```

```
cat("Spécificité à ce seuil:", best_specificity, "\n")
```

```
Spécificité à ce seuil: 0.998
```

b- Métrique de performance

Matrice de confusion

On applique ce seuil optimal à notre modèle pour obtenir les prédictions finales

```
final_predictions <- ifelse(predictions > best_threshold, "True", "False")
final_conf_matrix <- table(final_predictions, Actual = data$is_genuine)
print(final_conf_matrix)
```

```
                Actual
final_predictions   0   1
            False 499  48
            True    1 952
```

Avec ce modèle et ce seuil de sensibilité, on obtient un résultat intéressant pour le nombre de faux positif. Il n'y a qu'un seul faux billet détecté comme positif sur les 1500 billets de notre jeu de donnée.

```
taux_erreur <- 1/1500 * 100
print(paste("le taux de faux positif est de", format(taux_erreur, digits = 1, nsmall = 1), "%
```

```
[1] "le taux de faux positif est de 0.07 % avec ce modèle et ce seuil"
```

Mesures de performance

```
# Calculer les mesures de performance
precision_rl_model_2 <- final_conf_matrix[2, 2] / (final_conf_matrix[2, 2] + final_conf_matri
recall_rl_model_2 <- final_conf_matrix[2, 2] / (final_conf_matrix[2, 2] + final_conf_matrix[
specificity_rl_model_2 <- final_conf_matrix[1, 1] / (final_conf_matrix[1, 1] + final_conf_mat
f1_score_rl_model_2 <- 2 * (precision_rl_model_2 * recall_rl_model_2) / (precision_rl_model_2
# Calculer Accuracy
accuracy_rl_model_2 <- (final_conf_matrix[1, 1] + final_conf_matrix[2, 2]) / sum(final_conf_m
# Afficher les résultats
cat("Précision: ", precision_rl_model_2, "\n")
```

```
Précision:  0.9989507
```

```
cat("Sensibilité (Recall): ", recall_rl_model_2, "\n")
```

```
Sensibilité (Recall):  0.952
```

```
cat("Spécificité: ", specificity_rl_model_2, "\n")
```

```
Spécificité:  0.998
```

```
cat("F1-Score: ", f1_score_rl_model_2, "\n")
```

```
F1-Score:  0.9749104
```

```
cat("Accuracy: ", accuracy_rl_model_2, "\n")
```

```
Accuracy:  0.9673333
```

```
print(paste("L'éfficacité golobale (accuracy) du modèle avec un seuil de 0.5 était de", roun
```

```
[1] "L'éfficacité golobale (accuracy) du modèle avec un seuil de 0.5 était de 99.13 % tandis
```

En regardant les performances globales, on voit que nous avons effectivement amélioré la Spécificité, c'est à dire réduire le risque de faux positifs, passant d'une spécificité de 98.2% à 99.8%.
99.8% des faux billets sont donc détectés dans notre jeu de données initial.
Nous avons également augmenté la précision passant de 0.991 à 0.998, cela signifie que 99.8% des billets classés dans "True" sont réellement vrais.

Cependant nous avons une baisse de sensibilité passant de 0.996 à 0.952 ce qui signifie que seulement 95.2 % des vrais billets sont identifiés et classés comme tel, donc 4.8 % de vrais billets sont considérés comme faux par le modèle. et une baisse du F1_score passant de 0.993 à 0.97, c'est à dire un moins bon équilibre entre détection des faux et des vrais billets dans le global.

Nous allons tester d'autres méthodes de classification pour comparer les modèles et leur performance.

6- Test d'autres méthodes

a- Méthode RANDOM FOREST

```
# Charger les packages nécessaires
if (!require(randomForest)) install.packages("randomForest")
```

```
Le chargement a nécessité le package : randomForest
```

```
randomForest 4.7-1.1
```

```
Type rfNews() to see new features/changes/bug fixes.
```

```
Attachement du package : 'randomForest'
```

```
L'objet suivant est masqué depuis 'package:ggplot2':
```

```
    margin
```

```
L'objet suivant est masqué depuis 'package:dplyr':
```

```
    combine
```

```
if (!require(caret)) install.packages("caret")
library(randomForest)
library(caret)

# Préparer les données
features <- data[ , names(data) != "is_genuine"]
target <- as.factor(data$is_genuine)

# Re-niveler la variable cible pour que '1' (ou 'TRUE') soit la classe positive
target <- relevel(target, ref = "1")  # Assurez-vous que '1' est bien la classe positive

# Diviser les données en ensembles d'entraînement et de test
set.seed(123)
trainIndex <- createDataPartition(target, p = 0.7, list = FALSE)
trainData <- data[trainIndex, ]
testData <- data[-trainIndex, ]
```

```r
# Préparer les caractéristiques et la cible pour l'entraînement et le test
trainFeatures <- trainData[ , names(trainData) != "is_genuine"]
trainTarget <- as.factor(trainData$is_genuine)
trainTarget <- relevel(trainTarget, ref = "1")  # Re-niveler la variable cible dans les donne

testFeatures <- testData[ , names(testData) != "is_genuine"]
testTarget <- as.factor(testData$is_genuine)
testTarget <- relevel(testTarget, ref = "1")  # Re-niveler aussi la variable cible dans les d

# Entraîner le modèle Random Forest pour classification
rf_model <- randomForest(x = trainFeatures, y = trainTarget, ntree = 100, importance = TRUE)
print(rf_model)
```

```
Call:
 randomForest(x = trainFeatures, y = trainTarget, ntree = 100,      importance = TRUE)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 2

        OOB estimate of  error rate: 0.95%
Confusion matrix:
    1   0 class.error
1 697   3 0.004285714
0   7 343 0.020000000
```

```r
# Faire des prédictions sur l'ensemble de test
rf_predictions <- predict(rf_model, testFeatures)

# Convertir les prédictions en facteur avec les niveaux de testTarget
rf_predictions <- factor(rf_predictions, levels = levels(testTarget))

# Évaluer le modèle
rf_conf_matrix <- confusionMatrix(rf_predictions, testTarget)
print(rf_conf_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
```

```
         1 298   3
         0   2 147

               Accuracy : 0.9889
                 95% CI : (0.9743, 0.9964)
    No Information Rate : 0.6667
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.975

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9933
            Specificity : 0.9800
         Pos Pred Value : 0.9900
         Neg Pred Value : 0.9866
             Prevalence : 0.6667
         Detection Rate : 0.6622
   Detection Prevalence : 0.6689
      Balanced Accuracy : 0.9867

       'Positive' Class : 1
```

```r
rf_accuracy <- rf_conf_matrix$overall['Accuracy']
rf_specificity <- rf_conf_matrix$byClass['Specificity']
rf_sensitivity <- rf_conf_matrix$byClass['Sensitivity']
rf_precision <- rf_conf_matrix$byClass['Pos Pred Value']
rf_f1_score <- 2 * (rf_precision * rf_sensitivity) / (rf_precision + rf_sensitivity)
```

Utilisation des variables déterminées par StepAIC pour voir si cela améliore le modèle Randomforest

```r
# Préparer les données pour Random Forest en utilisant les variables sélectionnées
selected_vars <- c("height_left", "height_right", "margin_low", "margin_up", "length")

trainData_selected <- trainData[, c(selected_vars, "is_genuine")]
testData_selected <- testData[, c(selected_vars, "is_genuine")]

# Convertir 'is_genuine' en facteur et re-niveler pour que '1' soit la classe positive
trainData_selected$is_genuine <- relevel(as.factor(trainData_selected$is_genuine), ref = "1")
testData_selected$is_genuine <- relevel(as.factor(testData_selected$is_genuine), ref = "1")
```

```r
library(randomForest)

# Construire le modèle Random Forest
rf_model_2 <- randomForest(is_genuine ~ ., data = trainData_selected, ntree = 100, importanc

# Afficher les résultats du modèle
print(rf_model_2)
```

```
Call:
 randomForest(formula = is_genuine ~ ., data = trainData_selected,      ntree = 100, importa
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 2

        OOB estimate of  error rate: 1.14%
Confusion matrix:
    1   0 class.error
1 697   3 0.004285714
0   9 341 0.025714286
```

```r
# Prédictions sur le jeu de test
predictions_2 <- predict(rf_model_2, newdata = testData_selected)

# Calculer la matrice de confusion et d'autres statistiques
library(caret)
rf2_conf_matrix <- confusionMatrix(predictions_2, testData_selected$is_genuine, positive = "
print(rf2_conf_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 297   2
         0   3 148

               Accuracy : 0.9889
                 95% CI : (0.9743, 0.9964)
    No Information Rate : 0.6667
    P-Value [Acc > NIR] : <2e-16
```

```
                 Kappa : 0.975

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9900
            Specificity : 0.9867
         Pos Pred Value : 0.9933
         Neg Pred Value : 0.9801
             Prevalence : 0.6667
         Detection Rate : 0.6600
   Detection Prevalence : 0.6644
      Balanced Accuracy : 0.9883

       'Positive' Class : 1
```

Les statistiques sont assez proche.

```
rf2_accuracy <- rf2_conf_matrix$overall['Accuracy']
rf2_specificity <- rf2_conf_matrix$byClass['Specificity']
rf2_sensitivity <- rf2_conf_matrix$byClass['Sensitivity']
rf2_precision <- rf2_conf_matrix$byClass['Pos Pred Value']
rf2_f1_score <- 2 * (rf2_precision * rf2_sensitivity) / (rf2_precision + rf2_sensitivity)
```

b- Méthode de Réseau Neuronal

Utilisation de la librairie “reticulate” qui intègre python dans R Choix de l’environnement Conda utilisé (contient les installations spécifiques pour l’utilisation de Tensorflow et keras) Chargement des librairie

```
library(reticulate)

# Utilisez le nouvel environnement Conda
use_condaenv("tf-env", required = TRUE)

library(keras)
```

```
Attachement du package : 'keras'
```

```
L'objet suivant est masqué depuis 'package:cvms':

    evaluate
```

```
# Vérifiez la configuration de Python
py_config()
```

```
python:         C:/Users/Utilisateur/Documents/Anaconda/envs/tf-env/python.exe
libpython:      C:/Users/Utilisateur/Documents/Anaconda/envs/tf-env/python39.dll
pythonhome:     C:/Users/Utilisateur/Documents/Anaconda/envs/tf-env
version:        3.9.13 | packaged by conda-forge | (main, May 27 2022, 16:51:29) [MSC v.1929
Architecture:   64bit
numpy:          C:/Users/Utilisateur/Documents/Anaconda/envs/tf-env/Lib/site-packages/numpy
numpy_version:  1.26.4
tensorflow:     C:\Users\UTILIS~1\DOCUME~1\Anaconda\envs\tf-env\lib\site-packages\tensorflow\

NOTE: Python version was forced by use_python() function
```

```
# Normalisation des variables continues
df_normalized <- data %>%
  mutate(across(c(diagonal, height_left, height_right, margin_low, margin_up, length), scale)

# Diviser les données en ensembles d'entraînement et de test
set.seed(52)  # Pour la reproductibilité
train_indices <- sample(1:nrow(df_normalized), 0.8 * nrow(df_normalized))
train_data <- df_normalized[train_indices, ]
test_data <- df_normalized[-train_indices, ]

# Préparer les entrées et sorties
x_train <- as.matrix(train_data %>% dplyr::select(everything(), -is_genuine))
y_train <- as.matrix(train_data$is_genuine)

x_test <- as.matrix(test_data %>% dplyr::select(everything(), -is_genuine))
y_test <- as.matrix(test_data$is_genuine)
```

Modèle 1 Réseau simple sans régularisation

```
library(keras)

# Créer le modèle
model_neuronal_1 <- keras_model_sequential() %>%
```

```
  layer_dense(units = 32, activation = 'relu', input_shape = c(ncol(x_train))) %>%
  layer_dense(units = 16, activation = 'relu') %>%
  layer_dense(units = 1, activation = 'sigmoid')
# Compiler le modèle
model_neuronal_1 %>% compile(
  loss = 'binary_crossentropy',
  optimizer = optimizer_adam(),
  metrics = c('accuracy')
)
# Évaluer le modèle
score1 <- model_neuronal_1 %>% evaluate(x_test, y_test, positive ="1")
```

```
10/10 - 1s - loss: 0.7925 - accuracy: 0.3067 - 920ms/epoch - 92ms/step
```

```
print(score1)
```

```
     loss  accuracy
0.7924566 0.3066667
```

Le modèle 1 est assez simple et n'inclut aucune forme de régularisation. Il pourrait avoir un certain degré de surajustement (overfitting) ou de sous-ajustement (underfitting), mais la précision de 70 % montre qu'il capture une partie des relations dans les données. La perte est relativement élevée, indiquant que le modèle pourrait être amélioré.

```
# Faire des prédictions sur les données de test
predictions <- model_neuronal_1 %>% predict(x_test)
```

```
10/10 - 0s - 80ms/epoch - 8ms/step
```

```
# Convertir les probabilités en classes (0 ou 1) en utilisant un seuil de 0.5
classes <- ifelse(predictions > 0.5, 1, 0)

# Créer la matrice de confusion
conf_matrix <- confusionMatrix(as.factor(classes), as.factor(y_test), positive ="1")

# Afficher la matrice de confusion
print(conf_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  86 200
         1   8   6

               Accuracy : 0.3067
                 95% CI : (0.255, 0.3622)
    No Information Rate : 0.6867
    P-Value [Acc > NIR] : 1

                  Kappa : -0.036

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.02913
            Specificity : 0.91489
         Pos Pred Value : 0.42857
         Neg Pred Value : 0.30070
             Prevalence : 0.68667
         Detection Rate : 0.02000
   Detection Prevalence : 0.04667
      Balanced Accuracy : 0.47201

       'Positive' Class : 1
```

```r
# Extraire les métriques
accuracy_neurone_1 <- conf_matrix$overall['Accuracy']
specificity_neurone_1 <- conf_matrix$byClass['Specificity']
sensitivity_neurone_1 <- conf_matrix$byClass['Sensitivity']
f1_score_neurone_1 <- conf_matrix$byClass['F1']

# Afficher les résultats
cat("Accuracy:", accuracy_neurone_1, "\n")
```

```
Accuracy: 0.3066667
```

```r
cat("Specificity:", specificity_neurone_1, "\n")
```

```
Specificity: 0.9148936
```

```r
cat("Sensitivity:", sensitivity_neurone_1, "\n")
```

```
Sensitivity: 0.02912621
```

```r
cat("F1 Score:", f1_score_neurone_1, "\n")
```

```
F1 Score: 0.05454545
```

Modèle 2 Ajout de dropout pour régularisation et taille de couche plus élevée (désactivation de certains neurones pour éviter le surapprentissage)

```r
# Créer le modèle
model_neuronal_2 <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = 'relu', input_shape = c(ncol(x_train))) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 32, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 16, activation = 'relu') %>%
  layer_dense(units = 1, activation = 'sigmoid')
# Compiler le modèle
model_neuronal_2 %>% compile(
  loss = 'binary_crossentropy',
  optimizer = optimizer_adam(learning_rate = 0.001),
  metrics = c('accuracy')
)
# Évaluer le modèle
score2 <- model_neuronal_2 %>% evaluate(x_test, y_test)
```

```
10/10 - 0s - loss: 0.6955 - accuracy: 0.6033 - 173ms/epoch - 17ms/step
```

```r
print(score2)
```

```
     loss  accuracy
0.6955210 0.6033334
```

Le modèle 2, bien que plus complexe et régularisé avec Dropout, a une performance plus faible que le modèle 1. Cela pourrait être dû à une sur-régularisation, où le modèle perd trop d'information lors du processus de régularisation. De plus, l'architecture plus grande pourrait nécessiter un entraînement plus long ou des ajustements supplémentaires.

```r
# Faire des prédictions sur les données de test
predictions <- model_neuronal_2 %>% predict(x_test)
```

```
10/10 - 0s - 61ms/epoch - 6ms/step
```

```r
# Convertir les probabilités en classes (0 ou 1) en utilisant un seuil de 0.5
classes <- ifelse(predictions > 0.5, 1, 0)

# Créer la matrice de confusion
conf_matrix <- confusionMatrix(as.factor(classes), as.factor(y_test), positive ="1")

# Afficher la matrice de confusion
print(conf_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0   1  26
         1  93 180

               Accuracy : 0.6033
                 95% CI : (0.5455, 0.6591)
    No Information Rate : 0.6867
    P-Value [Acc > NIR] : 0.9991

                  Kappa : -0.1434

 Mcnemar's Test P-Value : 1.447e-09

            Sensitivity : 0.87379
            Specificity : 0.01064
         Pos Pred Value : 0.65934
         Neg Pred Value : 0.03704
             Prevalence : 0.68667
         Detection Rate : 0.60000
   Detection Prevalence : 0.91000
      Balanced Accuracy : 0.44221

       'Positive' Class : 1
```

```r
# Extraire les métriques
accuracy_neurone_2 <- conf_matrix$overall['Accuracy']
specificity_neurone_2 <- conf_matrix$byClass['Specificity']
sensitivity_neurone_2 <- conf_matrix$byClass['Sensitivity']
f1_score_neurone_2 <- conf_matrix$byClass['F1']

# Afficher les résultats
cat("Accuracy:", accuracy_neurone_2, "\n")
```

```
Accuracy: 0.6033333
```

```r
cat("Specificity:", specificity_neurone_2, "\n")
```

```
Specificity: 0.0106383
```

```r
cat("Sensitivity:", sensitivity_neurone_2, "\n")
```

```
Sensitivity: 0.8737864
```

```r
cat("F1 Score:", f1_score_neurone_2, "\n")
```

```
F1 Score: 0.7515658
```

Modèle 3 simplifié sans dropout mais entrainé sur 20 époques

```r
# Créer un modèle plus simple
model_neuronal_simplified <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = 'relu', input_shape = c(ncol(x_train))) %>%
  layer_dense(units = 32, activation = 'relu') %>%
  layer_dense(units = 16, activation = 'relu') %>%
  layer_dense(units = 1, activation = 'sigmoid')

# Compiler le modèle
model_neuronal_simplified %>% compile(
  loss = 'binary_crossentropy',
  optimizer = optimizer_adam(learning_rate = 0.001),
  metrics = c('accuracy')
)
```

```r
# Entraîner le modèle
history <- model_neuronal_simplified %>% fit(
  x_train, y_train,
  epochs = 20,  # Augmenter le nombre d'époques si nécessaire
  batch_size = 32,
  validation_split = 0.2
)
```

```
Epoch 1/20
30/30 - 1s - loss: 0.5326 - accuracy: 0.8208 - val_loss: 0.3437 - val_accuracy: 0.9792 - 564m
Epoch 2/20
30/30 - 0s - loss: 0.2161 - accuracy: 0.9917 - val_loss: 0.1138 - val_accuracy: 0.9875 - 129m
Epoch 3/20
30/30 - 0s - loss: 0.0691 - accuracy: 0.9937 - val_loss: 0.0593 - val_accuracy: 0.9875 - 113m
Epoch 4/20
30/30 - 0s - loss: 0.0375 - accuracy: 0.9948 - val_loss: 0.0502 - val_accuracy: 0.9875 - 98ms
Epoch 5/20
30/30 - 0s - loss: 0.0302 - accuracy: 0.9927 - val_loss: 0.0477 - val_accuracy: 0.9875 - 97ms
Epoch 6/20
30/30 - 0s - loss: 0.0270 - accuracy: 0.9937 - val_loss: 0.0489 - val_accuracy: 0.9875 - 104m
Epoch 7/20
30/30 - 0s - loss: 0.0254 - accuracy: 0.9937 - val_loss: 0.0478 - val_accuracy: 0.9875 - 105m
Epoch 8/20
30/30 - 0s - loss: 0.0245 - accuracy: 0.9937 - val_loss: 0.0473 - val_accuracy: 0.9875 - 112m
Epoch 9/20
30/30 - 0s - loss: 0.0230 - accuracy: 0.9937 - val_loss: 0.0481 - val_accuracy: 0.9875 - 109m
Epoch 10/20
30/30 - 0s - loss: 0.0230 - accuracy: 0.9948 - val_loss: 0.0495 - val_accuracy: 0.9875 - 105m
Epoch 11/20
30/30 - 0s - loss: 0.0218 - accuracy: 0.9937 - val_loss: 0.0496 - val_accuracy: 0.9875 - 113m
Epoch 12/20
30/30 - 0s - loss: 0.0210 - accuracy: 0.9948 - val_loss: 0.0506 - val_accuracy: 0.9875 - 104m
Epoch 13/20
30/30 - 0s - loss: 0.0205 - accuracy: 0.9958 - val_loss: 0.0495 - val_accuracy: 0.9875 - 111m
Epoch 14/20
30/30 - 0s - loss: 0.0202 - accuracy: 0.9958 - val_loss: 0.0507 - val_accuracy: 0.9875 - 107m
Epoch 15/20
30/30 - 0s - loss: 0.0195 - accuracy: 0.9948 - val_loss: 0.0500 - val_accuracy: 0.9875 - 98ms
Epoch 16/20
30/30 - 0s - loss: 0.0187 - accuracy: 0.9958 - val_loss: 0.0514 - val_accuracy: 0.9875 - 101m
Epoch 17/20
```

```
30/30 - 0s - loss: 0.0182 - accuracy: 0.9958 - val_loss: 0.0515 - val_accuracy: 0.9875 - 105r
Epoch 18/20
30/30 - 0s - loss: 0.0178 - accuracy: 0.9958 - val_loss: 0.0513 - val_accuracy: 0.9875 - 111r
Epoch 19/20
30/30 - 0s - loss: 0.0176 - accuracy: 0.9958 - val_loss: 0.0515 - val_accuracy: 0.9875 - 115r
Epoch 20/20
30/30 - 0s - loss: 0.0167 - accuracy: 0.9958 - val_loss: 0.0519 - val_accuracy: 0.9875 - 104r
```

```
# Évaluer le modèle
score_simplified <- model_neuronal_simplified %>% evaluate(x_test, y_test)
```

```
10/10 - 0s - loss: 0.0354 - accuracy: 0.9867 - 34ms/epoch - 3ms/step
```

```
print(score_simplified)
```

```
      loss   accuracy
0.03535791 0.98666668
```

Le modèle 3 est de loin le plus performant parmi les trois. La faible perte et la haute précision indiquent qu'il est très bien ajusté aux données. Cependant, il pourrait aussi y avoir un risque de surajustement (overfitting), car la performance sur l'ensemble de test est très proche de celle sur l'ensemble d'entraînement. L'absence de Dropout pourrait expliquer pourquoi le modèle performe aussi bien, mais cela signifie aussi que la généralisation à des données non vues pourrait poser problème si la complexité des données augmente.

```
# Faire des prédictions sur les données de test
predictions <- model_neuronal_simplified %>% predict(x_test)
```

```
10/10 - 0s - 57ms/epoch - 6ms/step
```

```
# Convertir les probabilités en classes (0 ou 1) en utilisant un seuil de 0.5
classes <- ifelse(predictions > 0.5, 1, 0)

# Créer la matrice de confusion
conf_matrix <- confusionMatrix(as.factor(classes), as.factor(y_test), positive ="1")

# Afficher la matrice de confusion
print(conf_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  92   2
         1   2 204

               Accuracy : 0.9867
                 95% CI : (0.9662, 0.9964)
    No Information Rate : 0.6867
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.969

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9903
            Specificity : 0.9787
         Pos Pred Value : 0.9903
         Neg Pred Value : 0.9787
             Prevalence : 0.6867
         Detection Rate : 0.6800
   Detection Prevalence : 0.6867
      Balanced Accuracy : 0.9845

       'Positive' Class : 1
```

```r
# Extraire les métriques
accuracy_neurone_3 <- conf_matrix$overall['Accuracy']
specificity_neurone_3 <- conf_matrix$byClass['Specificity']
sensitivity_neurone_3 <- conf_matrix$byClass['Sensitivity']
f1_score_neurone_3 <- conf_matrix$byClass['F1']

# Afficher les résultats
cat("Accuracy:", accuracy_neurone_3, "\n")
```

```
Accuracy: 0.9866667
```

```r
cat("Specificity:", specificity_neurone_3, "\n")
```

```
Specificity: 0.9787234
```

```r
cat("Sensitivity:", sensitivity_neurone_3, "\n")
```

```
Sensitivity: 0.9902913
```

```r
cat("F1 Score:", f1_score_neurone_3, "\n")
```

```
F1 Score: 0.9902913
```

Modèle 4 Dropout : Ajout d'un taux de 30 % de Dropout après chaque couche dense pour éviter que le modèle ne s'ajuste trop fortement aux données d'entraînement.

Régularisation L2 : Ajout d'une pénalisation L2 avec un coefficient de régularisation de 0.001 sur chaque couche dense pour limiter l'importance des poids.

Ajustement du taux d'apprentissage : Utilisation d'un plan de décroissance exponentielle du taux d'apprentissage pour affiner l'apprentissage avec le temps.

Early Stopping : Le modèle s'arrêtera automatiquement si la performance de validation ne s'améliore plus après 10 époques, et restaurera les meilleurs poids enregistrés.

Avantages :
Meilleure généralisation : Grâce à la régularisation et au Dropout.
Convergence optimale : Ajustement fin du taux d'apprentissage et contrôle du surapprentissage avec l'Early Stopping.

```r
# Stratégie d'ajustement du taux d'apprentissage (exponential decay)
learning_rate_schedule <- learning_rate_schedule_exponential_decay(
  initial_learning_rate = 0.001,
  decay_steps = 10000,
  decay_rate = 0.96,
  staircase = TRUE
)

# Optimizer avec le learning rate schedule
optimizer <- optimizer_adam(learning_rate = learning_rate_schedule)

# Créer le modèle amélioré avec Dropout et régularisation L2
model_neuronal_ameliore <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = 'relu',
              kernel_regularizer = regularizer_l2(0.001), input_shape = c(ncol(x_train))) %>%
  layer_dropout(rate = 0.3) %>%  # Dropout de 30% après la première couche
  layer_dense(units = 32, activation = 'relu',
```

```
              kernel_regularizer = regularizer_l2(0.001)) %>%
  layer_dropout(rate = 0.3) %>%  # Dropout de 30% après la deuxième couche
  layer_dense(units = 16, activation = 'relu',
              kernel_regularizer = regularizer_l2(0.001)) %>%
  layer_dense(units = 1, activation = 'sigmoid')  # Couche de sortie pour la classification l

# Compiler le modèle
model_neuronal_ameliore %>% compile(
  loss = 'binary_crossentropy',
  optimizer = optimizer,
  metrics = c('accuracy')
)

# Callback d'arrêt anticipé (Early Stopping)
early_stopping <- callback_early_stopping(
  monitor = "val_loss",  # Surveiller la perte sur l'ensemble de validation
  patience = 10,         # Nombre d'époques sans amélioration avant d'arrêter
  restore_best_weights = TRUE  # Restaurer les meilleurs poids
)

# Entraîner le modèle avec validation_split et arrêt anticipé
history <- model_neuronal_ameliore %>% fit(
  x_train, y_train,
  epochs = 50,  # Augmenter le nombre d'époques
  batch_size = 32,
  validation_split = 0.2,  # Utiliser 20% des données pour la validation
  callbacks = list(early_stopping)
)
```

```
Epoch 1/50
30/30 - 1s - loss: 0.6425 - accuracy: 0.8052 - val_loss: 0.4647 - val_accuracy: 0.9792 - 638r
Epoch 2/50
30/30 - 0s - loss: 0.3761 - accuracy: 0.9688 - val_loss: 0.2286 - val_accuracy: 0.9875 - 122r
Epoch 3/50
30/30 - 0s - loss: 0.2108 - accuracy: 0.9729 - val_loss: 0.1404 - val_accuracy: 0.9833 - 114r
Epoch 4/50
30/30 - 0s - loss: 0.1421 - accuracy: 0.9854 - val_loss: 0.1238 - val_accuracy: 0.9875 - 114r
Epoch 5/50
30/30 - 0s - loss: 0.1248 - accuracy: 0.9833 - val_loss: 0.1196 - val_accuracy: 0.9875 - 111r
Epoch 6/50
30/30 - 0s - loss: 0.1116 - accuracy: 0.9865 - val_loss: 0.1175 - val_accuracy: 0.9875 - 133r
Epoch 7/50
```

```
30/30 - 0s - loss: 0.1062 - accuracy: 0.9896 - val_loss: 0.1163 - val_accuracy: 0.9875 - 117
Epoch 8/50
30/30 - 0s - loss: 0.0976 - accuracy: 0.9896 - val_loss: 0.1153 - val_accuracy: 0.9875 - 115
Epoch 9/50
30/30 - 0s - loss: 0.0947 - accuracy: 0.9896 - val_loss: 0.1133 - val_accuracy: 0.9875 - 119
Epoch 10/50
30/30 - 0s - loss: 0.0897 - accuracy: 0.9927 - val_loss: 0.1120 - val_accuracy: 0.9875 - 115
Epoch 11/50
30/30 - 0s - loss: 0.0922 - accuracy: 0.9906 - val_loss: 0.1092 - val_accuracy: 0.9875 - 131
Epoch 12/50
30/30 - 0s - loss: 0.0808 - accuracy: 0.9917 - val_loss: 0.1087 - val_accuracy: 0.9875 - 115
Epoch 13/50
30/30 - 0s - loss: 0.0822 - accuracy: 0.9927 - val_loss: 0.1053 - val_accuracy: 0.9875 - 116
Epoch 14/50
30/30 - 0s - loss: 0.0802 - accuracy: 0.9906 - val_loss: 0.1039 - val_accuracy: 0.9875 - 114
Epoch 15/50
30/30 - 0s - loss: 0.0835 - accuracy: 0.9906 - val_loss: 0.1019 - val_accuracy: 0.9875 - 131
Epoch 16/50
30/30 - 0s - loss: 0.0783 - accuracy: 0.9896 - val_loss: 0.1007 - val_accuracy: 0.9875 - 127
Epoch 17/50
30/30 - 0s - loss: 0.0738 - accuracy: 0.9927 - val_loss: 0.1012 - val_accuracy: 0.9875 - 115
Epoch 18/50
30/30 - 0s - loss: 0.0697 - accuracy: 0.9927 - val_loss: 0.1002 - val_accuracy: 0.9875 - 117
Epoch 19/50
30/30 - 0s - loss: 0.0717 - accuracy: 0.9937 - val_loss: 0.0989 - val_accuracy: 0.9875 - 110
Epoch 20/50
30/30 - 0s - loss: 0.0736 - accuracy: 0.9917 - val_loss: 0.0974 - val_accuracy: 0.9875 - 119
Epoch 21/50
30/30 - 0s - loss: 0.0693 - accuracy: 0.9927 - val_loss: 0.0949 - val_accuracy: 0.9875 - 114
Epoch 22/50
30/30 - 0s - loss: 0.0681 - accuracy: 0.9948 - val_loss: 0.0944 - val_accuracy: 0.9875 - 109
Epoch 23/50
30/30 - 0s - loss: 0.0655 - accuracy: 0.9948 - val_loss: 0.0934 - val_accuracy: 0.9875 - 108
Epoch 24/50
30/30 - 0s - loss: 0.0646 - accuracy: 0.9917 - val_loss: 0.0934 - val_accuracy: 0.9875 - 135
Epoch 25/50
30/30 - 0s - loss: 0.0623 - accuracy: 0.9927 - val_loss: 0.0920 - val_accuracy: 0.9875 - 116
Epoch 26/50
30/30 - 0s - loss: 0.0638 - accuracy: 0.9937 - val_loss: 0.0920 - val_accuracy: 0.9875 - 130
Epoch 27/50
30/30 - 0s - loss: 0.0620 - accuracy: 0.9937 - val_loss: 0.0903 - val_accuracy: 0.9875 - 120
Epoch 28/50
30/30 - 0s - loss: 0.0621 - accuracy: 0.9917 - val_loss: 0.0876 - val_accuracy: 0.9875 - 124
```

```
Epoch 29/50
30/30 - 0s - loss: 0.0596 - accuracy: 0.9948 - val_loss: 0.0895 - val_accuracy: 0.9875 - 124
Epoch 30/50
30/30 - 0s - loss: 0.0541 - accuracy: 0.9937 - val_loss: 0.0889 - val_accuracy: 0.9875 - 116
Epoch 31/50
30/30 - 0s - loss: 0.0589 - accuracy: 0.9937 - val_loss: 0.0869 - val_accuracy: 0.9875 - 114
Epoch 32/50
30/30 - 0s - loss: 0.0534 - accuracy: 0.9937 - val_loss: 0.0842 - val_accuracy: 0.9875 - 114
Epoch 33/50
30/30 - 0s - loss: 0.0522 - accuracy: 0.9937 - val_loss: 0.0819 - val_accuracy: 0.9875 - 116
Epoch 34/50
30/30 - 0s - loss: 0.0509 - accuracy: 0.9948 - val_loss: 0.0825 - val_accuracy: 0.9875 - 106
Epoch 35/50
30/30 - 0s - loss: 0.0570 - accuracy: 0.9937 - val_loss: 0.0799 - val_accuracy: 0.9875 - 113
Epoch 36/50
30/30 - 0s - loss: 0.0526 - accuracy: 0.9937 - val_loss: 0.0793 - val_accuracy: 0.9875 - 126
Epoch 37/50
30/30 - 0s - loss: 0.0488 - accuracy: 0.9948 - val_loss: 0.0819 - val_accuracy: 0.9875 - 110
Epoch 38/50
30/30 - 0s - loss: 0.0483 - accuracy: 0.9937 - val_loss: 0.0817 - val_accuracy: 0.9875 - 105
Epoch 39/50
30/30 - 0s - loss: 0.0509 - accuracy: 0.9927 - val_loss: 0.0828 - val_accuracy: 0.9875 - 116
Epoch 40/50
30/30 - 0s - loss: 0.0458 - accuracy: 0.9969 - val_loss: 0.0831 - val_accuracy: 0.9875 - 121
Epoch 41/50
30/30 - 0s - loss: 0.0472 - accuracy: 0.9937 - val_loss: 0.0819 - val_accuracy: 0.9875 - 125
Epoch 42/50
30/30 - 0s - loss: 0.0481 - accuracy: 0.9937 - val_loss: 0.0808 - val_accuracy: 0.9875 - 121
Epoch 43/50
30/30 - 0s - loss: 0.0506 - accuracy: 0.9948 - val_loss: 0.0783 - val_accuracy: 0.9875 - 118
Epoch 44/50
30/30 - 0s - loss: 0.0426 - accuracy: 0.9958 - val_loss: 0.0799 - val_accuracy: 0.9875 - 114
Epoch 45/50
30/30 - 0s - loss: 0.0435 - accuracy: 0.9937 - val_loss: 0.0796 - val_accuracy: 0.9875 - 120
Epoch 46/50
30/30 - 0s - loss: 0.0432 - accuracy: 0.9927 - val_loss: 0.0798 - val_accuracy: 0.9875 - 108
Epoch 47/50
30/30 - 0s - loss: 0.0440 - accuracy: 0.9958 - val_loss: 0.0785 - val_accuracy: 0.9875 - 117
Epoch 48/50
30/30 - 0s - loss: 0.0455 - accuracy: 0.9937 - val_loss: 0.0786 - val_accuracy: 0.9875 - 111
Epoch 49/50
30/30 - 0s - loss: 0.0438 - accuracy: 0.9937 - val_loss: 0.0783 - val_accuracy: 0.9875 - 129
Epoch 50/50
```

```
30/30 - 0s - loss: 0.0444 - accuracy: 0.9948 - val_loss: 0.0776 - val_accuracy: 0.9875 - 125
```

```r
# Évaluer le modèle sur les données de test
score_ameliore <- model_neuronal_ameliore %>% evaluate(x_test, y_test)
```

```
10/10 - 0s - loss: 0.0551 - accuracy: 0.9867 - 32ms/epoch - 3ms/step
```

```r
print(score_ameliore)
```

```
      loss   accuracy
0.05512204 0.98666668
```

```r
# Faire des prédictions sur les données de test
predictions <- model_neuronal_ameliore %>% predict(x_test)
```

```
10/10 - 0s - 61ms/epoch - 6ms/step
```

```r
# Convertir les probabilités en classes (0 ou 1) en utilisant un seuil de 0.5
classes <- ifelse(predictions > 0.5, 1, 0)

# Créer la matrice de confusion
conf_matrix <- confusionMatrix(as.factor(classes), as.factor(y_test), positive ="1")

# Afficher la matrice de confusion
print(conf_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  92   2
         1   2 204

               Accuracy : 0.9867
                 95% CI : (0.9662, 0.9964)
    No Information Rate : 0.6867
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.969
```

```
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9903
            Specificity : 0.9787
         Pos Pred Value : 0.9903
         Neg Pred Value : 0.9787
             Prevalence : 0.6867
         Detection Rate : 0.6800
   Detection Prevalence : 0.6867
      Balanced Accuracy : 0.9845

       'Positive' Class : 1
```

```r
# Extraire les métriques
accuracy_neurone_4 <- conf_matrix$overall['Accuracy']
specificity_neurone_4 <- conf_matrix$byClass['Specificity']
sensitivity_neurone_4 <- conf_matrix$byClass['Sensitivity']
f1_score_neurone_4 <- conf_matrix$byClass['F1']

# Afficher les résultats
cat("Accuracy:", accuracy_neurone_4, "\n")
```

```
Accuracy: 0.9866667
```

```r
cat("Specificity:", specificity_neurone_4, "\n")
```

```
Specificity: 0.9787234
```

```r
cat("Sensitivity:", sensitivity_neurone_4, "\n")
```

```
Sensitivity: 0.9902913
```

```r
cat("F1 Score:", f1_score_neurone_4, "\n")
```

```
F1 Score: 0.9902913
```

COMPARAISON DES PERFORMANCES ENTRE LES MODELES

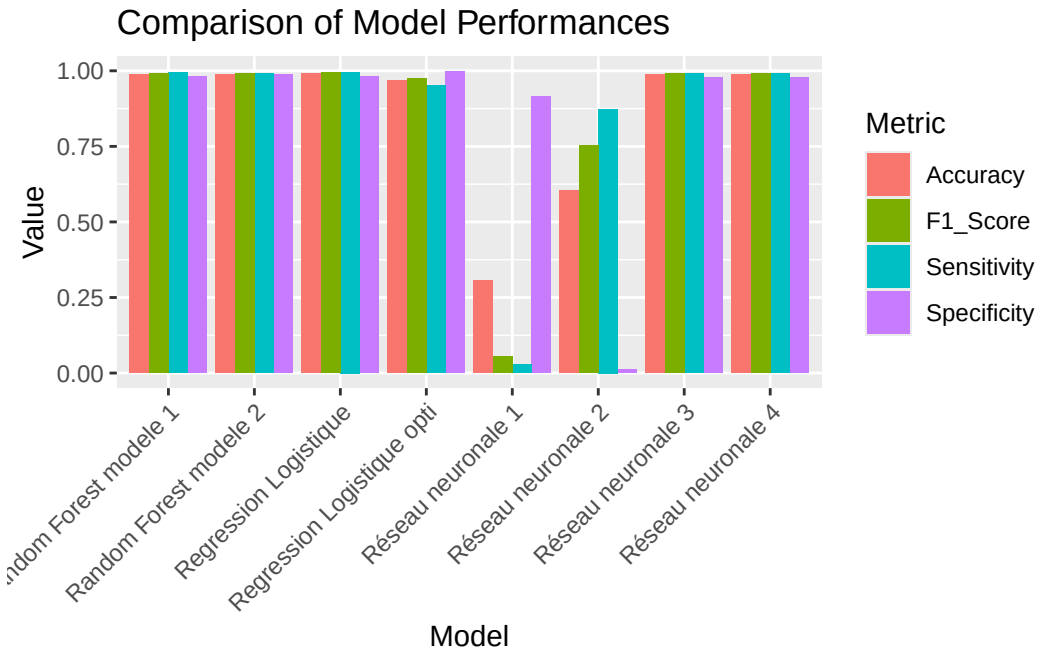
```r
# Installer et charger le package tibble pour une meilleure gestion des tableaux (si ce n'est
if (!require(tibble)) install.packages("tibble")
library(tibble)

# Créer un tableau de comparaison
results <- tibble(
  Model = c("Regression Logistique", "Regression Logistique opti", "Random Forest modele 1",
            "Réseau neuronale 1", "Réseau neuronale 2", "Réseau neuronale 3", "Réseau neurona
  Accuracy = c(accuracy_rl_model_1, accuracy_rl_model_2, rf_accuracy, rf2_accuracy,
               accuracy_neurone_1, accuracy_neurone_2, accuracy_neurone_3, accuracy_neurone_
  Specificity = c(specificity_rl_model_1, specificity_rl_model_2, rf_specificity, rf2_specifi
                  specificity_neurone_1, specificity_neurone_2, specificity_neurone_3, speci
  Sensitivity = c(recall_rl_model_1, recall_rl_model_2, rf_sensitivity, rf2_sensitivity,
                  sensitivity_neurone_1, sensitivity_neurone_2, sensitivity_neurone_3, sensi
  F1_Score = c(f1_score_rl_model_1, f1_score_rl_model_2, rf_f1_score, rf2_f1_score,
               f1_score_neurone_1, f1_score_neurone_2, f1_score_neurone_3, f1_score_neurone_4
)

# Afficher le tableau
print(results)
```

```
# A tibble: 8 x 5
  Model                      Accuracy Specificity Sensitivity F1_Score
  <chr>                         <dbl>       <dbl>       <dbl>    <dbl>
1 Regression Logistique         0.991       0.982       0.996    0.994
2 Regression Logistique opti    0.967       0.998       0.952    0.975
3 Random Forest modele 1        0.989       0.98        0.993    0.992
4 Random Forest modele 2        0.989       0.987       0.99     0.992
5 Réseau neuronale 1            0.307       0.915       0.0291   0.0545
6 Réseau neuronale 2            0.603       0.0106      0.874    0.752
7 Réseau neuronale 3            0.987       0.979       0.990    0.990
8 Réseau neuronale 4            0.987       0.979       0.990    0.990
```

Accuracy : Taux global de prédiction correcte. Specificity : Capacité à identifier les véritables négatifs. Sensitivity : Capacité à identifier les véritables positifs. F1 Score : Moyenne harmonique de la précision et du rappel, utile pour évaluer la performance dans des contextes déséquilibrés.

```r
library(ggplot2)

# Réorganiser les données pour le graphique
```

```r
results_long <- results %>%
  pivot_longer(cols = c(Accuracy, Specificity, Sensitivity, F1_Score), names_to = "Metric", v

# Créer un graphique comparatif
ggplot(results_long, aes(x = Model, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Comparison of Model Performances", x = "Model", y = "Value")
```



Sauvegarde du modèle à utiliser Modèle réseau neuronal 4

```r
# Sauvegarder le modèle neuronales (librairie keras)
model_neuronal_ameliore %>% save_model_hdf5("model_faux_billets.h5")
```

Modèle régréssion logistique opti il correspond à rl_model_1 avec application du seuil optimal

```r
# Sauvegarder le modèle et le seuil
saveRDS(rl_model_1, "modele_regression_logistique.rds")
saveRDS(best_threshold, "seuil_optimal.rds")
```