Rapport de projet d'analyse de données

Clustering de populations et apprentissage supervisé de diagnostic du cancer du sein

Nicolas SALVAN Alexandre CORRIOU

Spécialité MAIN
Polytech Sorbonne
Année académique 2023-2024
29 Mai 2024

Table des matières

1 Introduction

Le cancer du sein est le cancer le plus commun, il affecte plus de 2 millions de personnes chaque année, nous avons donc choisi d'étudier un dataset qui contient des données sur cette maladie. Ces données comprennent des mesures de caractéristiques de cellules de tumeurs mammaires, ainsi que l'aspect des cancers.

FIGURE 1 – Manifestation pour l'octobre rose à Rennes (2016)

Notre choix s'est porté sur ce jeu de données spécifique en raison de sa qualité vérifiée et de sa fiabilité puisqu'il est recommandé par Kaggle et reconnu par la communauté scientifique. Le dataset est disponible sur kaggle, et il est accessible sur le lien suivant :

`https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset`

Ce jeu de données pourrait être exploité, notamment pour déceler des groupes de cancer à l'aide d'outils de clustering, ou encore pour réaliser de l'apprentissage supervisée pour permettre de diagnostiquer les cancers bénins et malins.

Au cours de ce rapport, nous allons dans un premier temps étudier le jeu de données à l'aide de statistiques descriptives, puis nous allons déterminer l'importance des variables en utilisant de l'ACP et de l'AFD. Dans un second temps, nous essayerons de déceler les différentes classes de tumeurs en faisant du clustering. Enfin, dans un troisième temps, nous appliquerons nos connaissances en apprentissage supervisé pour construire un modèle qui permet de classer les tumeurs selon si elles sont bénignes ou malignes.

2 Présentation du jeu de données

Nous avons rassemblé dans le fichier `Projet-1sur5-Traitement_des_Donnees.Rmd` les différents codes utilisés pour cette partie.

2.1 Description

Comme indiqué précedemment, le jeu de données que nous avons choisi est constitué de **569 individus et 32 variables**. Il présente différentes caractéristiques de tumeurs de cancer du sein. Parmi les variables qualitatives, on retrouve l'identifiant de la patiente ainsi que la classification du cancer parmi Bénin ou Malin. Le reste des 30 variables est qualitatif, parmi elles on retrouve notamment le périmètre de la tumeur, la concavité, etc.

2.2 Pré-traitement des données

Ce jeu de données ne comportait pas de données nulles ou incohérentes, ce qui nous a permis de le garder quasimment intact pour presque toutes les méthodes utilisées ici. Il a fallu supprimer la colonne `id` puisqu'elle n'est pas intéressante pour notre analyse. Il a fallu également convertir la colonne `diagnosis` en facteur car elle était de type `str`.

2.3 Données de test et d'entraînement

Nous avons également créé, à partir des données nettoyées, un dataset d'entraînement et un dataset de test. Nous avons choisi de prendre 20% des données pour les tests, et 80% pour l'entraînement car nous avons relativement peu de données.

Nous avons également créé un dataset d'entraînement qui est équilibré, c'est-à-dire qu'il contient autant d'individus Bénin `B` que Malin `M`.

3 Statistiques descriptives

Nous allons ensuite réaliser les statistiques descriptives de notre jeu de données. Cette partie traite des résultats obtenus dans le fichier :

`Projet-2sur5-Statistiques_Descriptives.Rmd`

3.1 Aperçu rapide

Nous avons utlisé la fonction `summary` pour obtenir un aperçu de notre jeu de données. Les variables quantitatives sont vraiement très nombreuses, il nous a fallu les visualiser autrement.

3.2 Distribution des diagnostics

Nous avons fait un histogramme des effectifs de chaque classe de diagnostic.

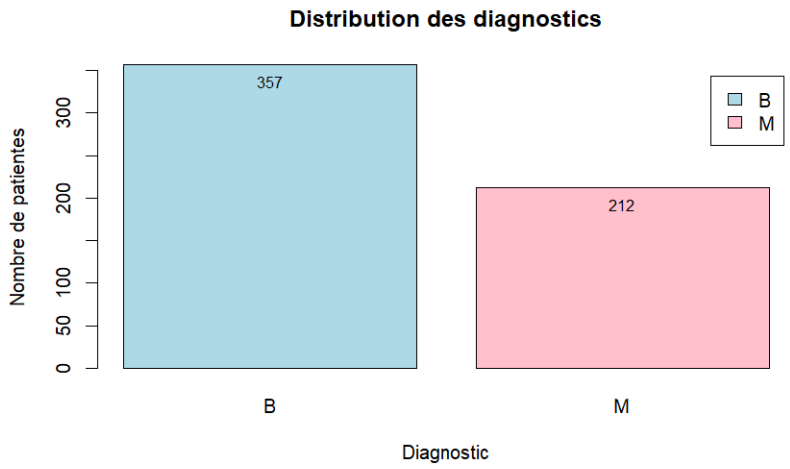


FIGURE 2 – Histogramme des effectifs des diagnostics

Nous avons observé la distribution de l'unique variable qualitative, à savoir `diagnosis`. Notre jeu de données est ainsi constitué de 62.7% d'individus `B` et de 37.3% d'individus `M`. On a un léger déséquilibre quant à la proportion d'individu des deux classes, ainsi, on peut s'attendre à des différences de performance et à prédire mieux la classe `B` que la classe `M`.

3.3 Distribution des variables qualitatives

Nous avons utilisé un corrélogramme, qui est une représentation d'une matrice de corrélation de nos données, pour visualiser les différents liens de corrélation entre elles.

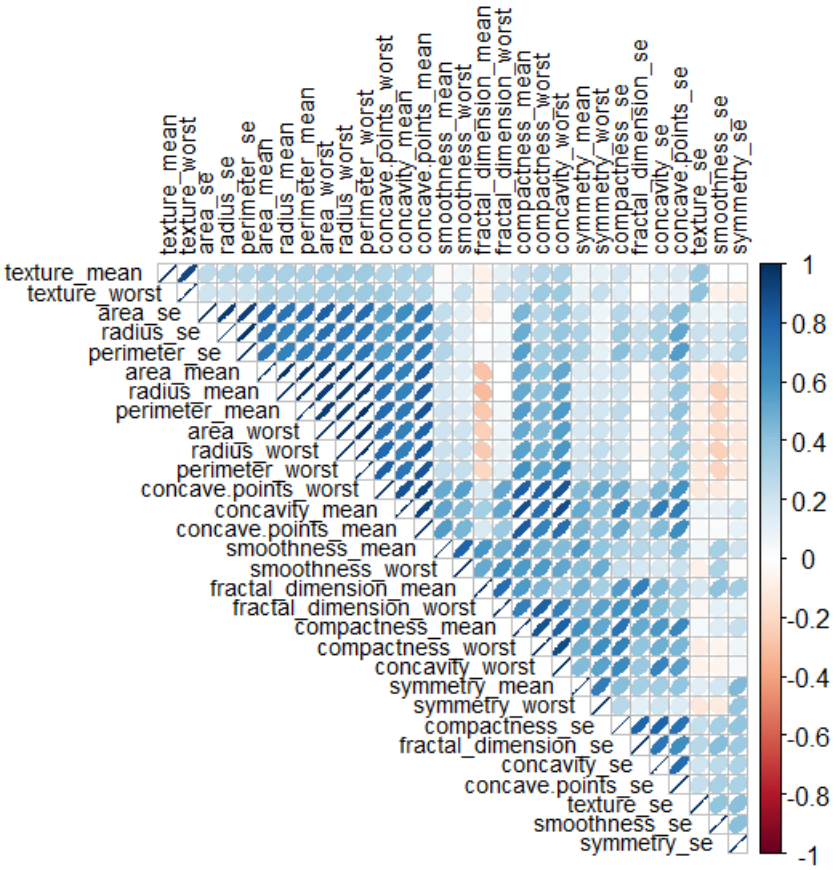


FIGURE 3 – Corrélogramme des variables qualitatives

Nous pouvons observer des liens de corrélations forts entre certaines variables. Nous avons alors cherché quelles variables étaient le plus corrélées, en valeur absolue et nous les avons affichés.

```
## radius_mean with perimeter_mean
## radius_mean with area_mean
## perimeter_mean with texture_mean
## radius_worst with perimeter_mean
## radius_worst with area_mean
```

FIGURE 4 – Extrait de sortie R montrant les variables les plus corrélées entre elles

Les liens de corrélations que la matrice de corrélation a permis de voir sont notamment les variables qui décrivent le périmètre, l'aire et le rayon de la tumeur, et cela s'explique simplement par le lien géométrique entre ces trois variables qui sont liées par des formules. Le lien entre `perimeter_mean` et `texture_mean` est plus difficile à expliquer néanmoins. Ces relations de corrélation pourront être simplifiées lors de nos modélisations.

4 Importance des variables

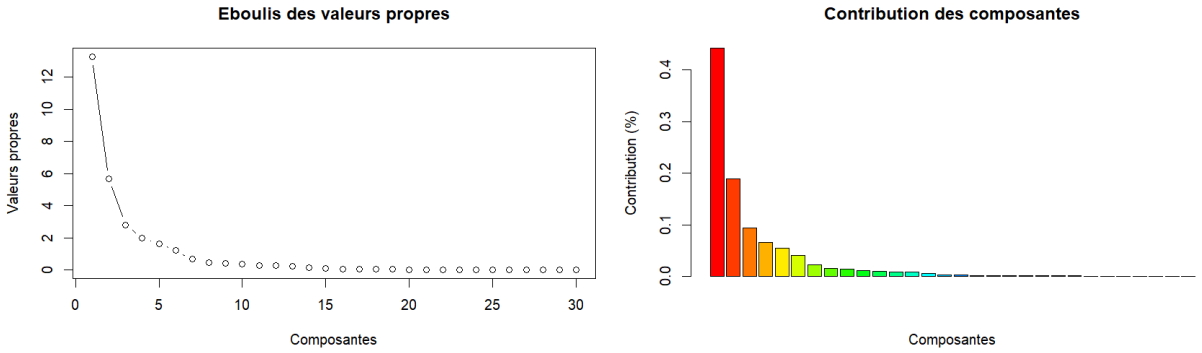
Nous avons réalisé différentes méthodes pour pouvoir déceler les variables qui décrivent le plus nos donneés. Cette partie se réfère au code du fichier `Projet-4sur5-ACP_et_AFD.Rmd`.

4.1 Analyse en Composantes Principales

Nous avons réalisé une ACP sur notre jeu de données pour pouvoir expliquer la dispersion de celles-ci.

4.1.1 Nombre de variables explicatives

En affichant les valeurs propres de la matrice de corrélation, nous avons obtenir les composantes qui expliquent le plus la dispersion.



(a) Valeurs propres

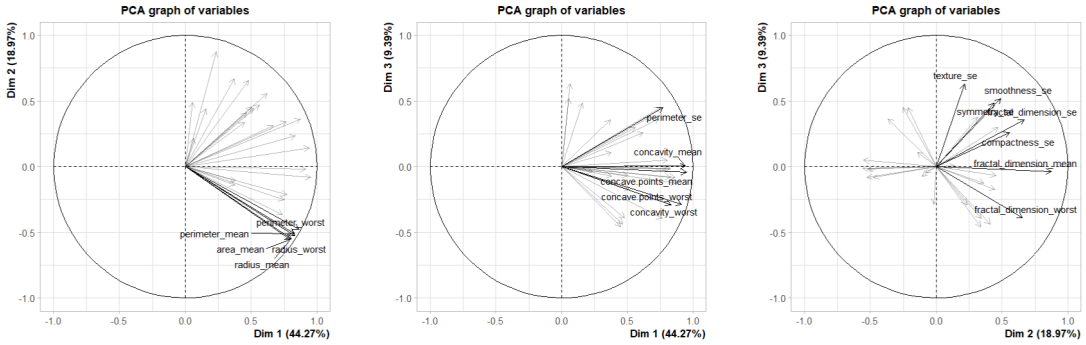
(b) Pourcentage de contribution

FIGURE 5 – ACP : Graphiques montrant les valeurs propres en fonction des composantes

Nous avons remarqué que près de 71% des données étaient expliquées par les **trois composantes principales**.

4.1.2 Cercles de corrélations sur les plans principaux

Nous avons également étudié la contribution de chaque variable pour les axes principaux à l'aide d'un cercle de corrélation.



(a) Plan (F1,F2) (b) Plan (F1,F3) (c) Plan (F2,F3)

FIGURE 6 – Cercles de corrélation des variables sur les plans principaux

Dans un premier temps, on remarque que certains plans sont plus représentatifs que d'autres : en effet, le plan $(F2, F3)$ n'a aucune variable avec un $cos2$ au dessus de 0.8. Nous avons affiché les variables qui contribuent le plus aux axes dans un histogramme.

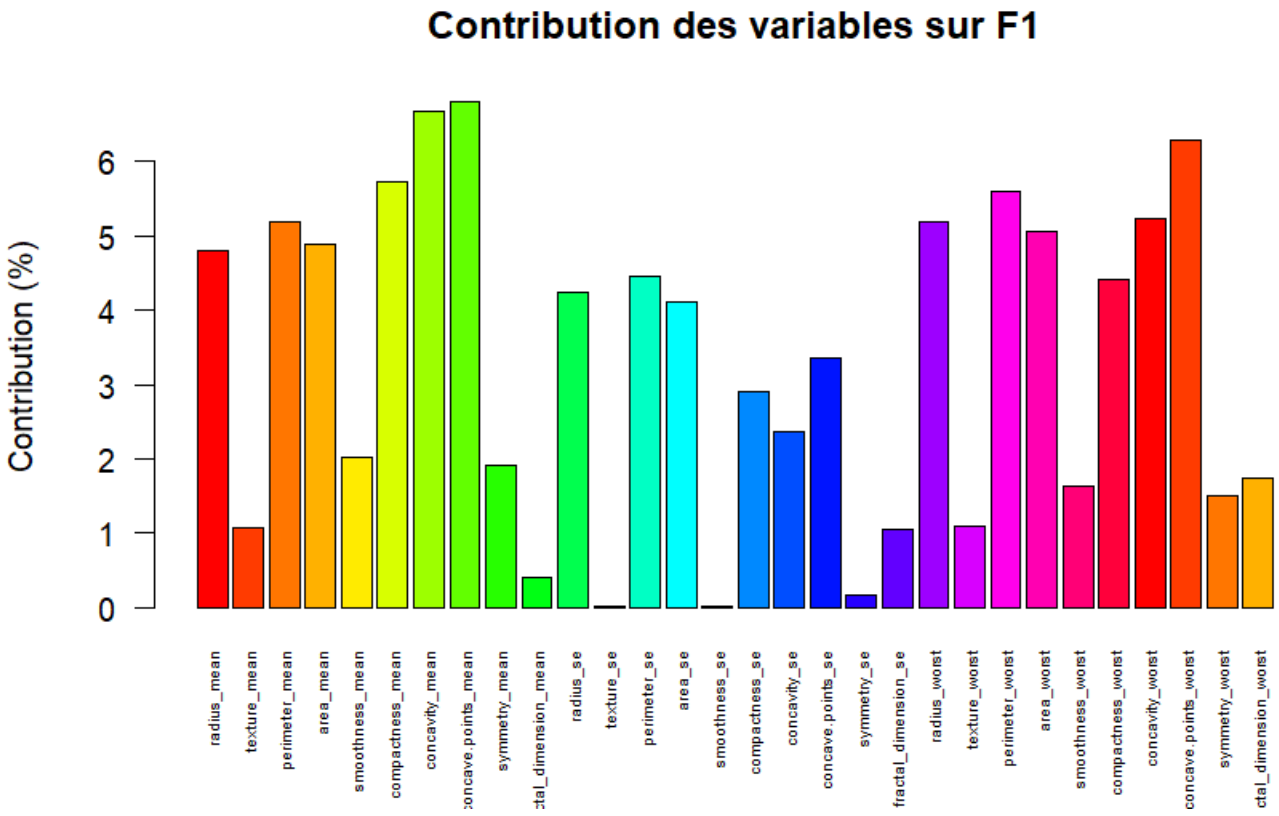


FIGURE 7 – Contributions des variables à l'axe $F1$

L'axe F1 est constitué principalement des variables qui régissent la texture des tumeurs,

comme la `fractal_dimension`, `concavity`, `compactness`, etc. Cela pourrait correspondre aux déformations provoquées par la tumeur. Sur les sorties R en annexe dans la partie 8, nous pouvons voir que les variables qui contribuent le plus à l'axe F2 sont principalement les variantes de `fractal_dimension`. Pour l'axe 3, on n'obtient pas des projections satisfaisantes pour l'exploiter.

4.1.3 Représentation des individus

Nous avons pu observer que certains individus contribuaient un peu plus aux axes principaux.

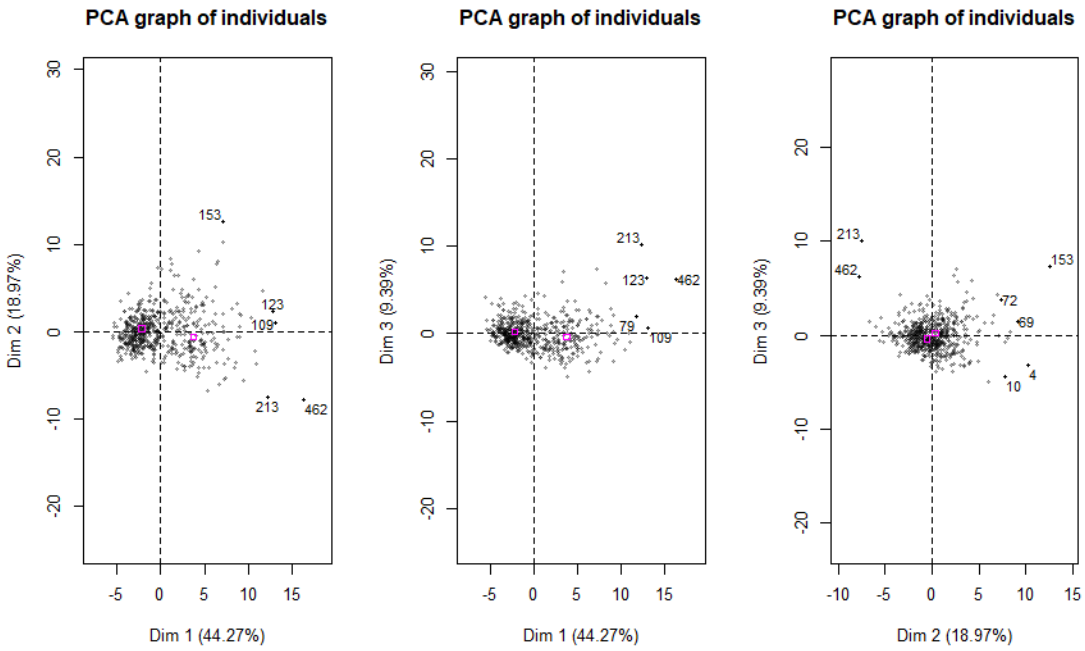


FIGURE 8 – Contribution des individus sur les plans $F12$, $F13$, $F23$

Nous avons repéré que les individus 462, 213 et 123 contribuent fortement à ces trois axes. Néanmoins, sur les plots, on observe qu'avec la distribution en nuage de points en données, cette importance ne semble pas si gênante.

On remarque également que les centres de gravité des classes `B` et `M` sont plutôt proches de l'axe $F1$ et disposées d'un côté et de l'autre des axes 2 et 3. Cela nous pousse à croire que l'axe F1 serait assez discriminant.

4.1.4 Représentation des classes sur les plans principaux

Nous avons affiché les classes des individus pour essayer de voir si l'on a trouvé un axe discriminant avec l'ACP, en sachant que les centres de gravité des classes sont situées proches de $F1$.

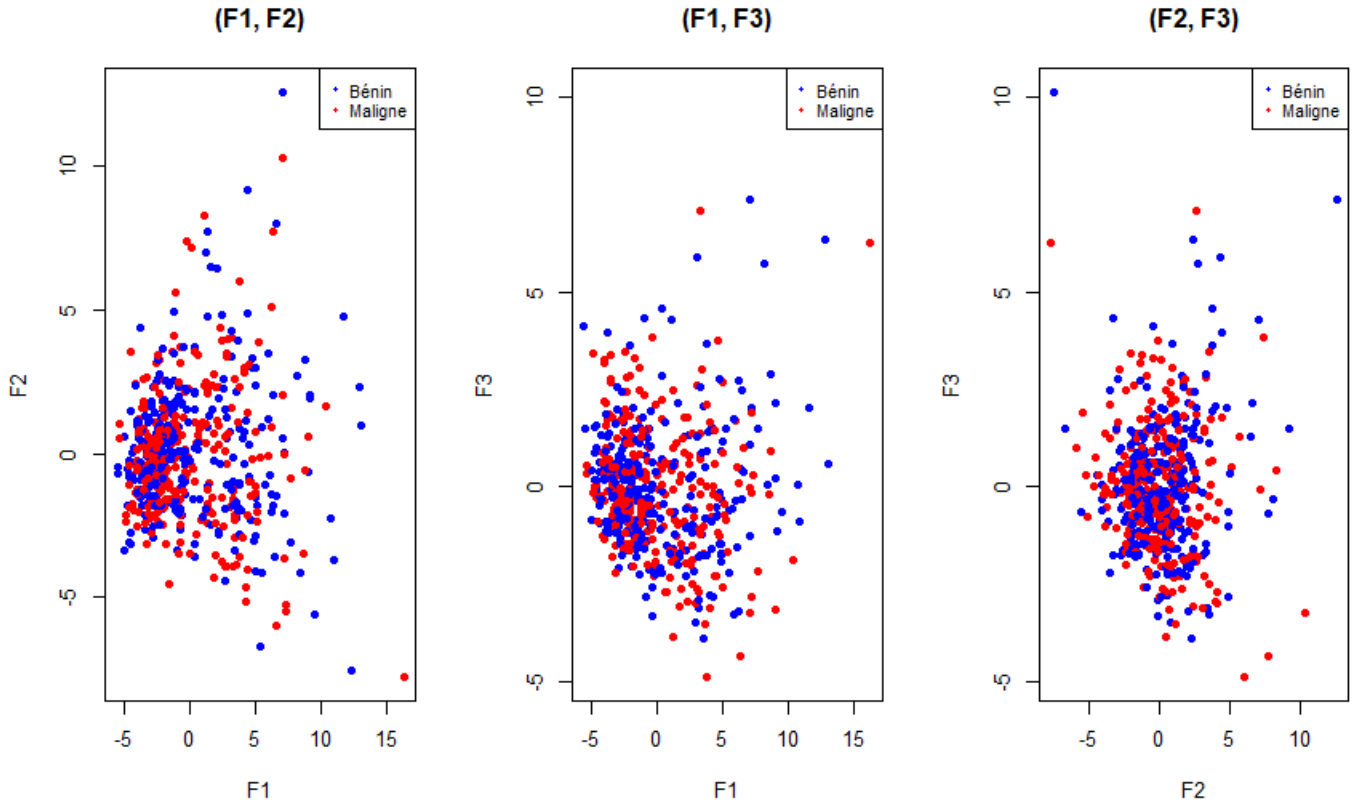


FIGURE 9 – Caption

On observe sur ces graphiques des individus qu'il est impossible de distinguer les diagnostics dans ces plans principaux, ce qui nous pousse à faire une AFD.

4.2 Analyse Factorielle discriminante

Nous avons réalisé une AFD sur notre jeu de données pour essayer de mieux les visualiser. Comme nous avions deux classes `B` et `M`, nous avons obtenu un unique axe discriminant.

4.2.1 Affichage des individus dans le plan discriminant

Nous pouvons affichier les données sur le plan discriminant.

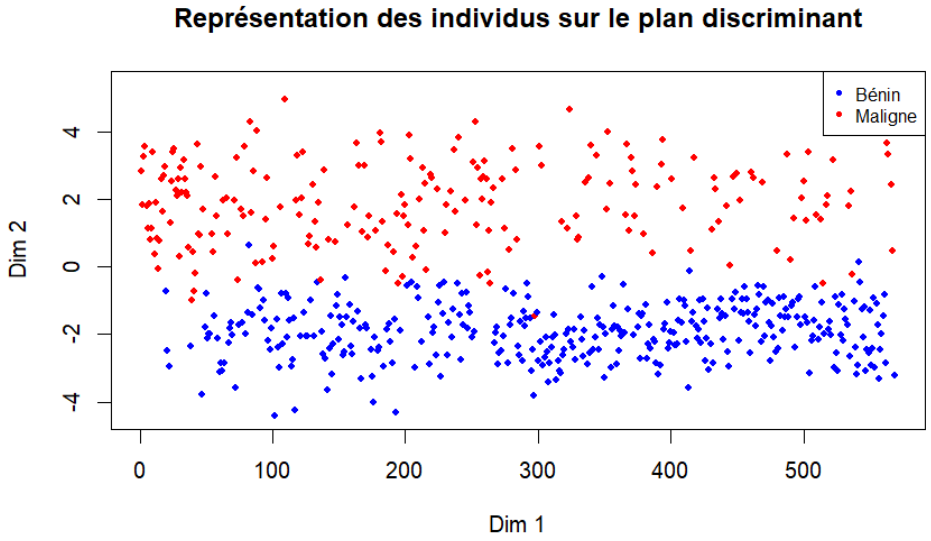


FIGURE 10 – Individus dans le plan discriminant

Nous observons bien plus facilement les deux classes du jeu de données que pour l'ACP.

4.2.2 Variables discriminantes

Comme il n'y a qu'un seul axe discriminant, nous pouvons obtenir le "cercle de corrélation" en projetant les variables sur celui-ci. Les variables les plus discriminantes sont celles liées à la concavité et à la taille de la tumeur.

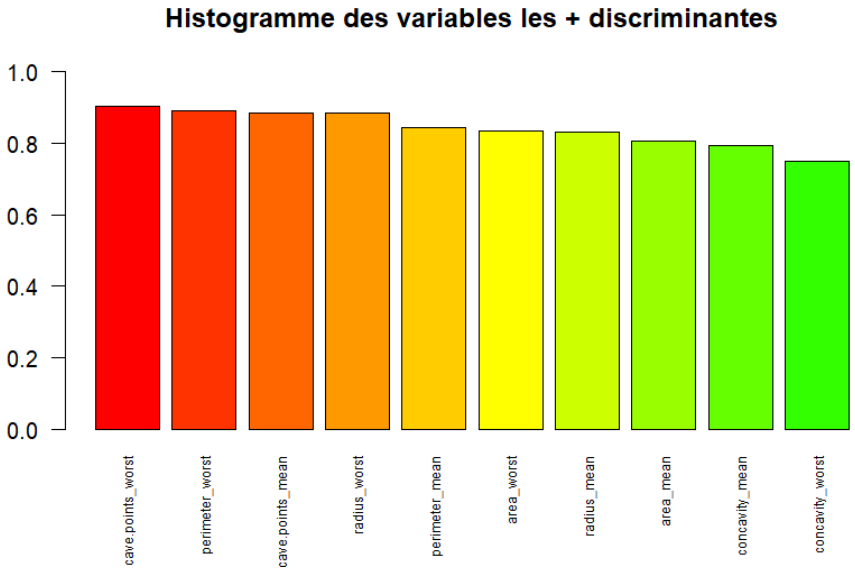


FIGURE 11 – Histogramme montrant les variables les plus discriminantes

5 Clustering

Même si nous avons déjà des classes, il peut être intéressant d'effectuer de la classification non-supervisée afin de voir si cette dernière obtient les mêmes résultats que la classification supervisée. Pour pouvoir effectuer cette classification, nous avons tout d'abord dû retirer la colonne contenant les classes. Une rapide étude du dendrogramme de ces données ainsi que de la hauteur de ce dernier nous permet de faire l'hypothèse qu'il existe deux ou trois groupes de cancers du sein. Nous effectuerons donc du clustering dans ces deux cas afin de voir quel est le meilleur modèle.

5.1 Clustering à deux classes

En effectuant le clustering avec deux classes, on trouve la matrice de confusion suivante :

	1	2
B	14	343
M	175	37

TABLE 1 – Matrice de confusion pour du clustering à 2 classes

La première classe correspond aux cancers bénins et la deuxième aux cancers malins. Nous obtenons une précision de 91% avec cette méthode.

5.2 Clustering à trois classes

Le clustering à trois classes nous offre une autre vision des choses. On découvre maintenant trois types de cancer du sein :

— Les cancers bénins
— Les cancers pour lesquels nous ne sommes pas vraiment sûrs
— Les cancers dont on est absolument sûrs qu'ils sont malins

	B	M
1	26	64
2	0	110
3	321	38

TABLE 2 – Matrice de confusion pour du clustering à 3 classes

6 Apprentissage supervisé pour le diagnostic de cancer

Nous avons pu tester différents modèles d'apprentissage supervisé pour pouvoir prédire les diagnostics. Le code associé à cette partie est dans le fichier suivant :

`Projet-3sur5-Classification_Supervisee.Rmd`

Les principaux enjeux étaient de réussir à obtenir des performances similaires sur les classifications des tumeurs bénignes et malignes, en sachant qu'il y avait quand même un déséquilibre dans la représentation de ces classes. Il nous a fallu obtenir une bonne accuracy sur les modèles, mais également obtenir une courbe ROC satisfaisante. Enfin, nous avons essayé de chosir des modèles qui étaient le plus simplifié possible.

Il faut noter que, comme la variable à prédire était qualitative, nous n'avons pas fait de régression linéaire ni d'ANOVA.

6.1 Modèles testés

6.1.1 AFD, LDA, QDA

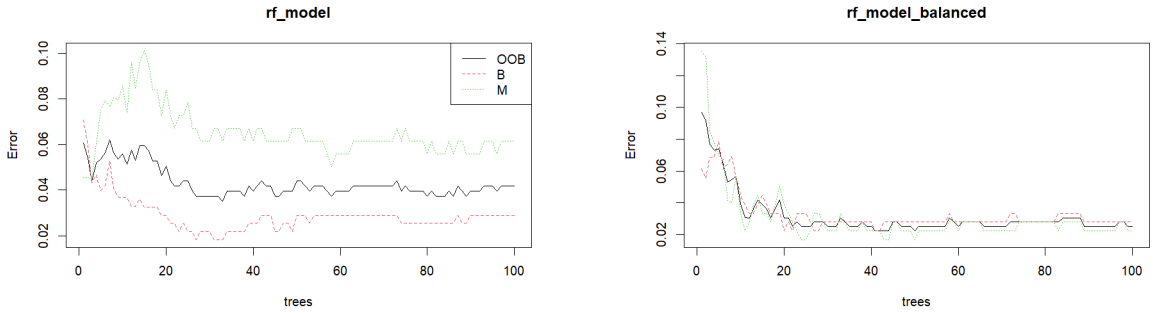
Nous avons dans un premier temps testé les modèles d'AFD, de LDA (Analyse discriminante Linéaire) et de QDA (Analyse discriminante quadratique). Puis dans un deuxième temps, nous avons simplifié les modèles de LDA et QDA pour supprimer le plus de variables inutiles, par exemple pour le QDA, le modèle ne dépendait plus de 27 variables sur les 30 tout en ayant des performances similaires.

6.1.2 CART

Nous avons testé trois modèles de classification CART sur nos données. Le premier, `cart_naive_model` était avec les paramètres par défaut. Le deuxième, `cart_model`, a été obtenu en simplifiant l'arbre pour éviter le sur-apprentissage. Le dernier, `cart_opti`, a été obtenu en optimisant sur le paramètre `cp`.

6.1.3 RandomForest

Nous avons testé deux modèles de RandomForest, car lorsque nous lancions l'algorithme naïvement, nous obtenions de moins bons résultats pour la classification minoritaire `M`. Nous avons donc entraîné un modèle sur des données équilibrées, et obtenu les graphes suivants.



(a) Sans données équilibrées (b) Avec données équilibrées

FIGURE 12 – Erreur OOB en fonction du nombre d'arbre et du type de données d'entraînement du modèle RandomForest

On observe bien sur la figure de droite que l'erreur `M` s'est stabilisée pour la classe `M` vers une valeur plus proche de l'erreur OOB.

6.1.4 Régression logistique

Nous avons réalisé une régression logistique sur nos données, mais nous avons eu des problèmes avec l'algorithme qui ne convergeait pas et qui n'était pas très bien qualibré. Nous avons donc réalisé de la régression logistique pénalisée type Lasso et type Ridge.

En obtenant des performances meilleures que la régression logistique non pénalisée, les modèles obtenus sont plus simples. Par exemple, la régression Lasso dépend de 12 variables sur les 31.

6.1.5 SVM avec noyau linéaire et RBF

Comme nous avons étudié les SVM (Support vector machine) lors de nos projets industriels, nous l'avons testé sur nos données avec un noyau linéaire et un noyau RBF, qui détecte les relations non linéaires.

6.1.6 ADABoost

Nous avons testé différents modèles d'ADABoost, que nous avons calibré. Nous avons également fait varier le modèle de prédiction pour Adaboost entre `bernoulli` et `adaboost`.

6.2 Comparaison des courbes ROC

Pour chacun de ces modèles, nous avons réalisé des prédictions sur les données de test. Nous avons ensuite tracé les courbes ROC pour chaque modèle.

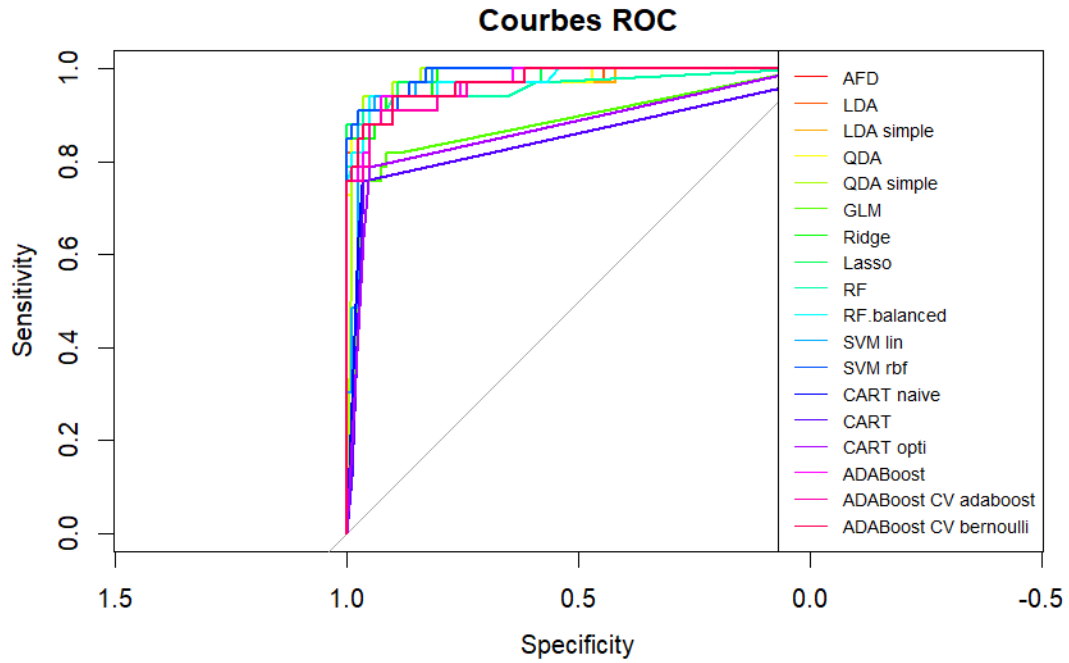


FIGURE 13 – Courbes ROC des différents modèles testés

Nous avons ensuite calculé l'AUC de chaque modèle.

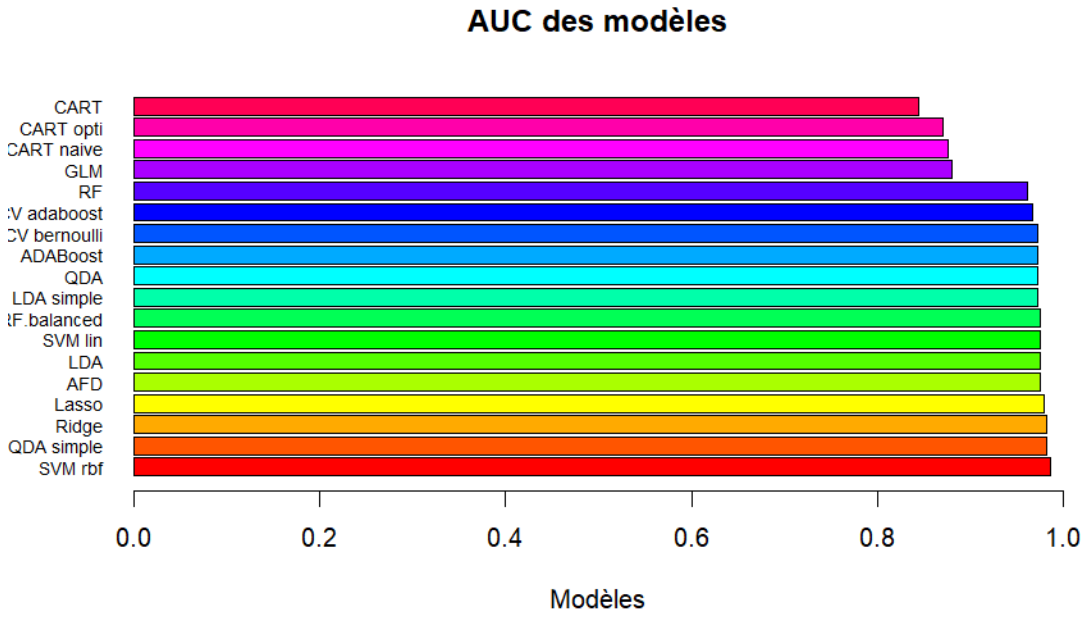


FIGURE 14 – AUC des différents modèles

```
## Min.      1st Qu.   Median    Mean     3rd Qu.    Max.
## 0.8447    0.9611    0.9729    0.9502   0.9753     0.9862
```

FIGURE 15 – Résumé des performances en AUC des modèles testés

En moyenne, on obtient une AUC de 0.9502 sur nos données test.

On obtient que le meilleur modèle relativement à l'AUC trouvé est le SVM avec noyau RBF (non-linéaire). Le meilleur modèle trouvé que l'on a vu dans ce cours est le modèle de régression logistique avec pénalisation de Ridge, avec une AUC de 0.981. De plus, le troisième meilleur modèle, à savoir la régression Lasso, a une AUC assez proche de celle de la régression Ridge, tout en dépendant de moins de variables.

Les modèles les moins bons sont les modèles CART, qui ont des AUC inférieures à 0.9, et de régression logistique non pénalisée, qui ont des performances très inférieures aux autres.

6.3 Comparaison des scores de précision

Nous avons pu calculer, pour chaque modèle, le score de précision ou accuracy.

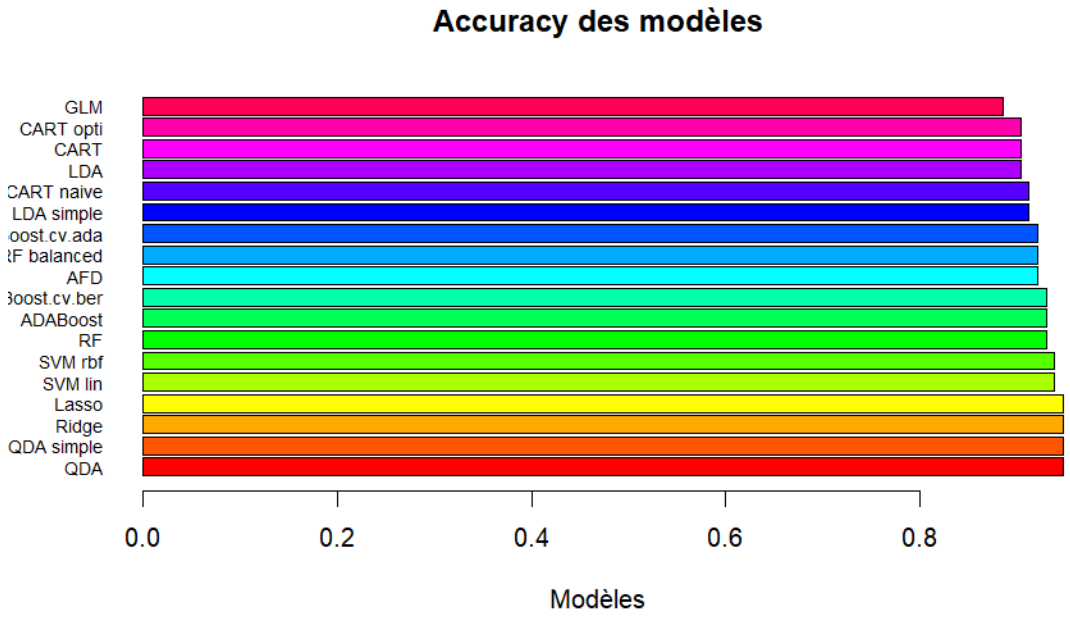


FIGURE 16 – AUC des différents modèles

```
## Min.    1st Qu.  Median   Mean     3rd Qu.    Max.
## 0.8860  0.9123   0.9211   0.9230   0.9386     0.9561
```

FIGURE 17 – Résumé des performances en accuracy des modèles testés

On obtient une accuracy de 0. Les modèles avec les meilleures accuracy sont les modèles de QDA, de régression logistique avec pénalisation de Ridge et de régression Lasso. Comme la QDA n'est pas dans les meilleurs modèles en terme d'AUC, on peut dire que les modèles de régression logistique avec pénalisation de Ridge et de régression Lasso sont les meilleurs modèles de statistiques prédictives que l'on a trouvé. Une très bonne accuracy pour la QDA peut être un signe de sur-apprentissage, il est donc à nuancer.
Le modèle qui a le moins fonctionné est le modèle de régression logistique sans pénalisation. En effet, lorsque nous lancions l'analyse, nous obtenions des warnings ce qui signifiait que le modèle n'était pas bien calibré. Le modèle CART a également eu des performances moins intéressantes que l'on a pu observer avec son accuracy plus basse que les autres, tout comme la LDA.

6.4 Conclusion sur l'apprentissage supervisé

Notre jeu de données peut s'adonner à de l'apprentissage supervisé. Nous avons pu prédire les diagnostics des données tests avec une précision très satisfaisante, avec notamment les modèles de régression logistique Ridge et Lasso. Pour mettre en perspective nos résultats, il faudrait faire de la cross-validation sur nos données de test et d'entraînement.

7 Conclusion

Nous avons réussi à faire des modèles de classification supervisée et non-supervisée sur nos données. Parmi les meilleurs modèles, on retrouve la régression logistique de type Ridge et Lasso qui correspondaient bien à nos données. Comme le diagnostic d'un cancer est une donnée très sensible, il est nécessaire de réaliser d'autres tests néanmoins.

Bien que nos résultats soient très encourageant (entre 90 et 95% de précision sur la plupart méthodes), nous pourrions tenter d'orienter de futures recherches dans le but de pouvoir annoncer à une patiente que la tumeur est bénigne sans se tromper.

8 Sources des images

Image de la page de garde
`https://blog.elueslocales.fr/delegation/handicap-sante-senior/octobre-rose-6-actions-ori`

Table des figures

9 Annexes

9.1 Récapitulatif des performances des modèles

model	accuracy	auc
AFD	0.921052631578947	0.975308641975309
LDA	0.903508771929825	0.975308641975309
LDA.simple	0.912280701754386	0.971941638608305
QDA	0.947368421052632	0.972315750093528
QDA.simple	0.956140350877193	0.964085297418631
GLM	0.885964912280702	0.880097268986158
Ridge	0.947368421052632	0.981668537224093
Lasso	0.947368421052632	0.979423868312757
RF	0.929824561403509	0.960157126823794
RF.bal	0.921052631578947	0.976056864945754
SVM lin	0.93859649122807	0.975308641975309
SVM rbf	0.93859649122807	0.986157875046764
CART naive	0.912280701754386	0.876169098391321
CART	0.903508771929825	0.844743733632622
CART opti	0.903508771929825	0.87037037037037
ADABoost	0.912280701754386	0.973438084549196
ADA.cv.ada	0.912280701754386	0.967078189300412
ADA.cv.ber	0.921052631578947	0.974186307519641

TABLE 3 – Tableau récapitulant les scores des différents modèles

9.2 Code du pré-traitement des données

Projet 1/5 Traitement-Des-Donnees

Nicolas SALVAN - Alexandre CORRIOU

2024-05-17

Lecture des données

Notre jeu de données contient des informations sur des patientes atteintes d'un cancer du sein. Nous allons commencer par lire les données et les afficher pour mieux les comprendre. Ce fichier contient le code pour *pré-traiter les données*.

Importation du dataset

```
data <- read.csv("data/breast-cancer.csv", header = TRUE, sep = ",")
```

Affichage des premières lignes

```
head(data)
```

```
##         id diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1   842302         M       17.99        10.38         122.80    1001.0
## 2   842517         M       20.57        17.77         132.90    1326.0
## 3 84300903         M       19.69        21.25         130.00    1203.0
## 4 84348301         M       11.42        20.38          77.58     386.1
## 5 84358402         M       20.29        14.34         135.10    1297.0
## 6   843786         M       12.45        15.70          82.57     477.1
##   smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1         0.11840          0.27760         0.3001             0.14710
## 2         0.08474          0.07864         0.0869             0.07017
## 3         0.10960          0.15990         0.1974             0.12790
## 4         0.14250          0.28390         0.2414             0.10520
## 5         0.10030          0.13280         0.1980             0.10430
## 6         0.12780          0.17000         0.1578             0.08089
##   symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1        0.2419                0.07871    1.0950     0.9053        8.589
## 2        0.1812                0.05667    0.5435     0.7339        3.398
## 3        0.2069                0.05999    0.7456     0.7869        4.585
## 4        0.2597                0.09744    0.4956     1.1560        3.445
## 5        0.1809                0.05883    0.7572     0.7813        5.438
## 6        0.2087                0.07613    0.3345     0.8902        2.217
##   area_se smoothness_se compactness_se concavity_se concave.points_se
```

```
## 1  153.40      0.006399        0.04904      0.05373           0.01587
## 2   74.08      0.005225        0.01308      0.01860           0.01340
## 3   94.03      0.006150        0.04006      0.03832           0.02058
## 4   27.23      0.009110        0.07458      0.05661           0.01867
## 5   94.44      0.011490        0.02461      0.05688           0.01885
## 6   27.19      0.007510        0.03345      0.03672           0.01137
##   symmetry_se fractal_dimension_se radius_worst texture_worst perimeter_worst
## 1     0.03003             0.006193        25.38         17.33          184.60
## 2     0.01389             0.003532        24.99         23.41          158.80
## 3     0.02250             0.004571        23.57         25.53          152.50
## 4     0.05963             0.009208        14.91         26.50           98.87
## 5     0.01756             0.005115        22.54         16.67          152.20
## 6     0.02165             0.005082        15.47         23.75          103.40
##   area_worst smoothness_worst compactness_worst concavity_worst
## 1     2019.0           0.1622            0.6656          0.7119
## 2     1956.0           0.1238            0.1866          0.2416
## 3     1709.0           0.1444            0.4245          0.4504
## 4      567.7           0.2098            0.8663          0.6869
## 5     1575.0           0.1374            0.2050          0.4000
## 6      741.6           0.1791            0.5249          0.5355
##   concave.points_worst symmetry_worst fractal_dimension_worst
## 1               0.2654         0.4601                 0.11890
## 2               0.1860         0.2750                 0.08902
## 3               0.2430         0.3613                 0.08758
## 4               0.2575         0.6638                 0.17300
## 5               0.1625         0.2364                 0.07678
## 6               0.1741         0.3985                 0.12440
```

On observe qu'il y a une variable qualitative "diagnosis" qui correspond au diagnostic de la patiente. Toutes les autres variables sont quantitatives, et décrives les caractéristiques du cancer.

Affichage des dimensions

```
dim(data)
```

```
## [1] 569  32
```

Notre jeu de données contient 569 observations et 32 variables.

Affichage des types des variables

```
str(data)
```

```
## 'data.frame':    569 obs. of  32 variables:
##  $ id                      : int  842302 842517 84300903 84348301 84358402 843786 844359 84458202 844
##  $ diagnosis               : chr  "M" "M" "M" "M" ...
##  $ radius_mean             : num  18 20.6 19.7 11.4 20.3 ...
##  $ texture_mean            : num  10.4 17.8 21.2 20.4 14.3 ...
##  $ perimeter_mean          : num  122.8 132.9 130 77.6 135.1 ...
```

```
##  $ area_mean              : num  1001 1326 1203 386 1297 ...
##  $ smoothness_mean        : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
##  $ compactness_mean       : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
##  $ concavity_mean         : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
##  $ concave.points_mean    : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
##  $ symmetry_mean          : num  0.242 0.181 0.207 0.26 0.181 ...
##  $ fractal_dimension_mean : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
##  $ radius_se              : num  1.095 0.543 0.746 0.496 0.757 ...
##  $ texture_se             : num  0.905 0.734 0.787 1.156 0.781 ...
##  $ perimeter_se           : num  8.59 3.4 4.58 3.44 5.44 ...
##  $ area_se                : num  153.4 74.1 94 27.2 94.4 ...
##  $ smoothness_se          : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
##  $ compactness_se         : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
##  $ concavity_se           : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
##  $ concave.points_se      : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
##  $ symmetry_se            : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
##  $ fractal_dimension_se   : num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
##  $ radius_worst           : num  25.4 25 23.6 14.9 22.5 ...
##  $ texture_worst          : num  17.3 23.4 25.5 26.5 16.7 ...
##  $ perimeter_worst        : num  184.6 158.8 152.5 98.9 152.2 ...
##  $ area_worst             : num  2019 1956 1709 568 1575 ...
##  $ smoothness_worst       : num  0.162 0.124 0.144 0.21 0.137 ...
##  $ compactness_worst      : num  0.666 0.187 0.424 0.866 0.205 ...
##  $ concavity_worst        : num  0.712 0.242 0.45 0.687 0.4 ...
##  $ concave.points_worst   : num  0.265 0.186 0.243 0.258 0.163 ...
##  $ symmetry_worst         : num  0.46 0.275 0.361 0.664 0.236 ...
##  $ fractal_dimension_worst: num  0.1189 0.089 0.0876 0.173 0.0768 ...
```

Conversion des données qualitatives en factor

On observe que la variable “diagnosis” est de type “chr”. Nous allons la convertir en facteur pour faciliter l’analyse.

```
data$diagnosis <- as.factor(data$diagnosis)
str(data)
```

```
## 'data.frame':    569 obs. of  32 variables:
##  $ id                     : int  842302 842517 84300903 84348301 84358402 843786 844359 84458202 844
##  $ diagnosis              : Factor w/ 2 levels "B","M": 2 2 2 2 2 2 2 2 2 2 ...
##  $ radius_mean            : num  18 20.6 19.7 11.4 20.3 ...
##  $ texture_mean           : num  10.4 17.8 21.2 20.4 14.3 ...
##  $ perimeter_mean         : num  122.8 132.9 130 77.6 135.1 ...
##  $ area_mean              : num  1001 1326 1203 386 1297 ...
##  $ smoothness_mean        : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
##  $ compactness_mean       : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
##  $ concavity_mean         : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
##  $ concave.points_mean    : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
##  $ symmetry_mean          : num  0.242 0.181 0.207 0.26 0.181 ...
##  $ fractal_dimension_mean : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
##  $ radius_se              : num  1.095 0.543 0.746 0.496 0.757 ...
##  $ texture_se             : num  0.905 0.734 0.787 1.156 0.781 ...
##  $ perimeter_se           : num  8.59 3.4 4.58 3.44 5.44 ...
##  $ area_se                : num  153.4 74.1 94 27.2 94.4 ...
```

```
##  $ smoothness_se          : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
##  $ compactness_se         : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
##  $ concavity_se           : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
##  $ concave.points_se      : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
##  $ symmetry_se            : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
##  $ fractal_dimension_se   : num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
##  $ radius_worst           : num  25.4 25 23.6 14.9 22.5 ...
##  $ texture_worst          : num  17.3 23.4 25.5 26.5 16.7 ...
##  $ perimeter_worst        : num  184.6 158.8 152.5 98.9 152.2 ...
##  $ area_worst             : num  2019 1956 1709 568 1575 ...
##  $ smoothness_worst       : num  0.162 0.124 0.144 0.21 0.137 ...
##  $ compactness_worst      : num  0.666 0.187 0.424 0.866 0.205 ...
##  $ concavity_worst        : num  0.712 0.242 0.45 0.687 0.4 ...
##  $ concave.points_worst   : num  0.265 0.186 0.243 0.258 0.163 ...
##  $ symmetry_worst         : num  0.46 0.275 0.361 0.664 0.236 ...
##  $ fractal_dimension_worst: num  0.1189 0.089 0.0876 0.173 0.0768 ...
```

Tous les types de variables semblent corrects.

Nettoyage

Certaines variables sont inutilisables, comme l'identifiant de la patiente. Nous allons les supprimer. Il nous faut également supprimer les NaNs pour éviter les erreurs dans les analyses.

```r
# suppression des NaNs
data <- na.omit(data)

# suppression des colonnes inutiles : identifiant de la patiente
data <- data[,-c(1)]
head(data)
```

```
##   diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean
## 1         M       17.99        10.38         122.80    1001.0         0.11840
## 2         M       20.57        17.77         132.90    1326.0         0.08474
## 3         M       19.69        21.25         130.00    1203.0         0.10960
## 4         M       11.42        20.38          77.58     386.1         0.14250
## 5         M       20.29        14.34         135.10    1297.0         0.10030
## 6         M       12.45        15.70          82.57     477.1         0.12780
##   compactness_mean concavity_mean concave.points_mean symmetry_mean
## 1          0.27760         0.3001             0.14710        0.2419
## 2          0.07864         0.0869             0.07017        0.1812
## 3          0.15990         0.1974             0.12790        0.2069
## 4          0.28390         0.2414             0.10520        0.2597
## 5          0.13280         0.1980             0.10430        0.1809
## 6          0.17000         0.1578             0.08089        0.2087
##   fractal_dimension_mean radius_se texture_se perimeter_se area_se
## 1                0.07871    1.0950     0.9053        8.589  153.40
## 2                0.05667    0.5435     0.7339        3.398   74.08
## 3                0.05999    0.7456     0.7869        4.585   94.03
## 4                0.09744    0.4956     1.1560        3.445   27.23
## 5                0.05883    0.7572     0.7813        5.438   94.44
## 6                0.07613    0.3345     0.8902        2.217   27.19
##   smoothness_se compactness_se concavity_se concave.points_se symmetry_se
```

```
## 1      0.006399        0.04904      0.05373           0.01587     0.03003
## 2      0.005225        0.01308      0.01860           0.01340     0.01389
## 3      0.006150        0.04006      0.03832           0.02058     0.02250
## 4      0.009110        0.07458      0.05661           0.01867     0.05963
## 5      0.011490        0.02461      0.05688           0.01885     0.01756
## 6      0.007510        0.03345      0.03672           0.01137     0.02165
##   fractal_dimension_se radius_worst texture_worst perimeter_worst area_worst
## 1             0.006193        25.38         17.33          184.60     2019.0
## 2             0.003532        24.99         23.41          158.80     1956.0
## 3             0.004571        23.57         25.53          152.50     1709.0
## 4             0.009208        14.91         26.50           98.87      567.7
## 5             0.005115        22.54         16.67          152.20     1575.0
## 6             0.005082        15.47         23.75          103.40      741.6
##   smoothness_worst compactness_worst concavity_worst concave.points_worst
## 1           0.1622            0.6656          0.7119               0.2654
## 2           0.1238            0.1866          0.2416               0.1860
## 3           0.1444            0.4245          0.4504               0.2430
## 4           0.2098            0.8663          0.6869               0.2575
## 5           0.1374            0.2050          0.4000               0.1625
## 6           0.1791            0.5249          0.5355               0.1741
##   symmetry_worst fractal_dimension_worst
## 1         0.4601                 0.11890
## 2         0.2750                 0.08902
## 3         0.3613                 0.08758
## 4         0.6638                 0.17300
## 5         0.2364                 0.07678
## 6         0.3985                 0.12440
```

Exportation des données

Nous allons exporter les données nettoyées pour les utiliser dans les analyses suivantes.

```r
write.csv(data, "data/data_cleaned.csv", row.names = FALSE)
```

Séparation des données

Nous allons séparer les données en deux parties : une partie pour l'apprentissage et une partie pour le test.

```r
split_data <- function (data, train_ratio) {
  set.seed(123)
  n <- nrow(data)
  p <- ncol(data)-1
  test.ratio <- 1 - train_ratio
  n.test <- round(n*test.ratio)
  train_index <- sample(1:nrow(data), n.test)
  train_data <- data[-train_index,]
  test_data <- data[train_index,]
  return(list(train_data = train_data, test_data = test_data))
}
```

```r
data_split <- split_data(data, 0.8) # 1/5 des données pour le test
train_data <- data_split$train_data
test_data <- data_split$test_data
```

Exportation des données d'apprentissage et de test

```r
write.csv(train_data, "data/train_data.csv", row.names = FALSE)
write.csv(test_data, "data/test_data.csv", row.names = FALSE)
```

Il faut noter qu'il faudra convertir la colonne "diagnosis" en facteur dans les données d'apprentissage et de test, mais aussi dans les données cleaned.

Données d'entrainement équilibrées

```r
train_data_balanced <- rbind(train_data[train_data$diagnosis == "M",], train_data[train_data$diagnosis =
table(train_data_balanced$diagnosis)
```

```
##
##   B   M
## 179 179
```

Nous avons maintenant des données d'entrainement équilibrées.

Exportation des données d'apprentissage équilibrées

```r
write.csv(train_data_balanced, "data/train_data_balanced.csv", row.names = FALSE)
```

9.3 Code pour les statistiques descriptives

Projet 2/5 Statistiques Descriptives

Nicolas SALVAN - Alexandre CORRIOU

2024-05-17

Ce fichier contient le code pour réaliser les *statistiques descriptives* sur les données nettoyées.

Lecture des données nettoyées

Importation du dataset

```
data <- read.csv("data/data_cleaned.csv", header = TRUE, sep = ",")
data$diagnosis <- as.factor(data$diagnosis)
```

Aperçu rapide

```
head(data)
```

```
##   diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean
## 1         M       17.99        10.38         122.80    1001.0         0.11840
## 2         M       20.57        17.77         132.90    1326.0         0.08474
## 3         M       19.69        21.25         130.00    1203.0         0.10960
## 4         M       11.42        20.38          77.58     386.1         0.14250
## 5         M       20.29        14.34         135.10    1297.0         0.10030
## 6         M       12.45        15.70          82.57     477.1         0.12780
##   compactness_mean concavity_mean concave.points_mean symmetry_mean
## 1          0.27760         0.3001             0.14710        0.2419
## 2          0.07864         0.0869             0.07017        0.1812
## 3          0.15990         0.1974             0.12790        0.2069
## 4          0.28390         0.2414             0.10520        0.2597
## 5          0.13280         0.1980             0.10430        0.1809
## 6          0.17000         0.1578             0.08089        0.2087
##   fractal_dimension_mean radius_se texture_se perimeter_se area_se
## 1                0.07871    1.0950     0.9053        8.589  153.40
## 2                0.05667    0.5435     0.7339        3.398   74.08
## 3                0.05999    0.7456     0.7869        4.585   94.03
## 4                0.09744    0.4956     1.1560        3.445   27.23
## 5                0.05883    0.7572     0.7813        5.438   94.44
## 6                0.07613    0.3345     0.8902        2.217   27.19
##   smoothness_se compactness_se concavity_se concave.points_se symmetry_se
## 1      0.006399        0.04904      0.05373           0.01587     0.03003
```

```
## 2      0.005225         0.01308      0.01860           0.01340     0.01389
## 3      0.006150         0.04006      0.03832           0.02058     0.02250
## 4      0.009110         0.07458      0.05661           0.01867     0.05963
## 5      0.011490         0.02461      0.05688           0.01885     0.01756
## 6      0.007510         0.03345      0.03672           0.01137     0.02165
##   fractal_dimension_se radius_worst texture_worst perimeter_worst area_worst
## 1             0.006193        25.38         17.33          184.60     2019.0
## 2             0.003532        24.99         23.41          158.80     1956.0
## 3             0.004571        23.57         25.53          152.50     1709.0
## 4             0.009208        14.91         26.50           98.87      567.7
## 5             0.005115        22.54         16.67          152.20     1575.0
## 6             0.005082        15.47         23.75          103.40      741.6
##   smoothness_worst compactness_worst concavity_worst concave.points_worst
## 1           0.1622            0.6656          0.7119               0.2654
## 2           0.1238            0.1866          0.2416               0.1860
## 3           0.1444            0.4245          0.4504               0.2430
## 4           0.2098            0.8663          0.6869               0.2575
## 5           0.1374            0.2050          0.4000               0.1625
## 6           0.1791            0.5249          0.5355               0.1741
##   symmetry_worst fractal_dimension_worst
## 1         0.4601                 0.11890
## 2         0.2750                 0.08902
## 3         0.3613                 0.08758
## 4         0.6638                 0.17300
## 5         0.2364                 0.07678
## 6         0.3985                 0.12440
```

```
# dim(data)
# str(data)
```

Statistiques descriptives

Nous allons maintenant réaliser des statistiques descriptives sur les données nettoyées.

Résumé des données

```
summary(data)
```

```
##  diagnosis  radius_mean      texture_mean   perimeter_mean     area_mean
##  B:357     Min.   : 6.981   Min.   : 9.71   Min.   : 43.79   Min.   : 143.5
##  M:212     1st Qu.:11.700   1st Qu.:16.17   1st Qu.: 75.17   1st Qu.: 420.3
##            Median :13.370   Median :18.84   Median : 86.24   Median : 551.1
##            Mean   :14.127   Mean   :19.29   Mean   : 91.97   Mean   : 654.9
##            3rd Qu.:15.780   3rd Qu.:21.80   3rd Qu.:104.10   3rd Qu.: 782.7
##            Max.   :28.110   Max.   :39.28   Max.   :188.50   Max.   :2501.0
##  smoothness_mean   compactness_mean  concavity_mean    concave.points_mean
##  Min.   :0.05263   Min.   :0.01938   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.08637   1st Qu.:0.06492   1st Qu.:0.02956   1st Qu.:0.02031
##  Median :0.09587   Median :0.09263   Median :0.06154   Median :0.03350
##  Mean   :0.09636   Mean   :0.10434   Mean   :0.08880   Mean   :0.04892
```

```
##  3rd Qu.:0.10530   3rd Qu.:0.13040   3rd Qu.:0.13070   3rd Qu.:0.07400
##  Max.   :0.16340   Max.   :0.34540   Max.   :0.42680   Max.   :0.20120
##  symmetry_mean    fractal_dimension_mean   radius_se        texture_se
##  Min.   :0.1060   Min.   :0.04996        Min.   :0.1115   Min.   :0.3602
##  1st Qu.:0.1619   1st Qu.:0.05770        1st Qu.:0.2324   1st Qu.:0.8339
##  Median :0.1792   Median :0.06154        Median :0.3242   Median :1.1080
##  Mean   :0.1812   Mean   :0.06280        Mean   :0.4052   Mean   :1.2169
##  3rd Qu.:0.1957   3rd Qu.:0.06612        3rd Qu.:0.4789   3rd Qu.:1.4740
##  Max.   :0.3040   Max.   :0.09744        Max.   :2.8730   Max.   :4.8850
##   perimeter_se       area_se        smoothness_se      compactness_se
##  Min.   : 0.757   Min.   :  6.802   Min.   :0.001713   Min.   :0.002252
##  1st Qu.: 1.606   1st Qu.: 17.850   1st Qu.:0.005169   1st Qu.:0.013080
##  Median : 2.287   Median : 24.530   Median :0.006380   Median :0.020450
##  Mean   : 2.866   Mean   : 40.337   Mean   :0.007041   Mean   :0.025478
##  3rd Qu.: 3.357   3rd Qu.: 45.190   3rd Qu.:0.008146   3rd Qu.:0.032450
##  Max.   :21.980   Max.   :542.200   Max.   :0.031130   Max.   :0.135400
##   concavity_se     concave.points_se   symmetry_se       fractal_dimension_se
##  Min.   :0.00000   Min.   :0.000000   Min.   :0.007882   Min.   :0.0008948
##  1st Qu.:0.01509   1st Qu.:0.007638   1st Qu.:0.015160   1st Qu.:0.0022480
##  Median :0.02589   Median :0.010930   Median :0.018730   Median :0.0031870
##  Mean   :0.03189   Mean   :0.011796   Mean   :0.020542   Mean   :0.0037949
##  3rd Qu.:0.04205   3rd Qu.:0.014710   3rd Qu.:0.023480   3rd Qu.:0.0045580
##  Max.   :0.39600   Max.   :0.052790   Max.   :0.078950   Max.   :0.0298400
##   radius_worst   texture_worst   perimeter_worst    area_worst
##  Min.   : 7.93   Min.   :12.02   Min.   : 50.41   Min.   : 185.2
##  1st Qu.:13.01   1st Qu.:21.08   1st Qu.: 84.11   1st Qu.: 515.3
##  Median :14.97   Median :25.41   Median : 97.66   Median : 686.5
##  Mean   :16.27   Mean   :25.68   Mean   :107.26   Mean   : 880.6
##  3rd Qu.:18.79   3rd Qu.:29.72   3rd Qu.:125.40   3rd Qu.:1084.0
##  Max.   :36.04   Max.   :49.54   Max.   :251.20   Max.   :4254.0
##  smoothness_worst  compactness_worst concavity_worst  concave.points_worst
##  Min.   :0.07117   Min.   :0.02729   Min.   :0.0000   Min.   :0.00000
##  1st Qu.:0.11660   1st Qu.:0.14720   1st Qu.:0.1145   1st Qu.:0.06493
##  Median :0.13130   Median :0.21190   Median :0.2267   Median :0.09993
##  Mean   :0.13237   Mean   :0.25427   Mean   :0.2722   Mean   :0.11461
##  3rd Qu.:0.14600   3rd Qu.:0.33910   3rd Qu.:0.3829   3rd Qu.:0.16140
##  Max.   :0.22260   Max.   :1.05800   Max.   :1.2520   Max.   :0.29100
##  symmetry_worst   fractal_dimension_worst
##  Min.   :0.1565   Min.   :0.05504
##  1st Qu.:0.2504   1st Qu.:0.07146
##  Median :0.2822   Median :0.08004
##  Mean   :0.2901   Mean   :0.08395
##  3rd Qu.:0.3179   3rd Qu.:0.09208
##  Max.   :0.6638   Max.   :0.20750
```

Le jeu de données contient 569 observations et 31 variables. On observe qu'il y a une variable qualitative "diagnosis" qui correspond au diagnostic de la patiente. Toutes les autres variables sont quantitatives, et décrivent les caractéristiques du cancer détecté.

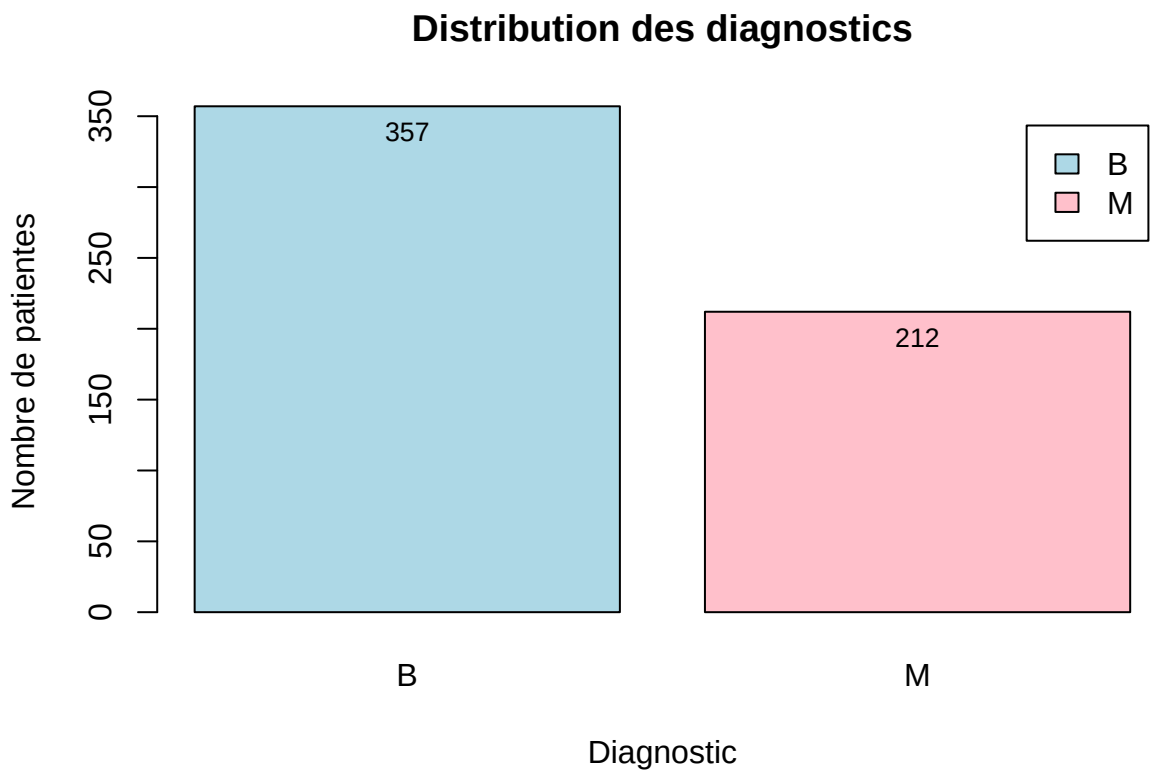
Cette sortie nous donne quelques statistiques descriptives sur nos données, notamment les moyennes, les médianes, les minimums et maximums, et les quartiles.

Distribution des données

Observons la distribution des différentes variables.

Distribution des diagnostics (variable qualitative)

```
counts <- table(data$diagnosis)
bp <- barplot(counts,
              main = "Distribution des diagnostics",
              xlab = "Diagnostic",
              ylab = "Nombre de patientes",
              col = c("lightblue", "pink"),
              legend = rownames(counts))

# Ajouter les labels au-dessus des barres
text(x = bp, y = counts, labels = counts, pos = 1, cex = 0.8, col = "black")
```



Ici, on peut voir les effectifs des deux diagnostics possibles : “M” pour “Malignant” ou Malin en français, et “B” pour “Benign” ou bénin.

```
proportions <- prop.table(table(data$diagnosis))
proportions
```

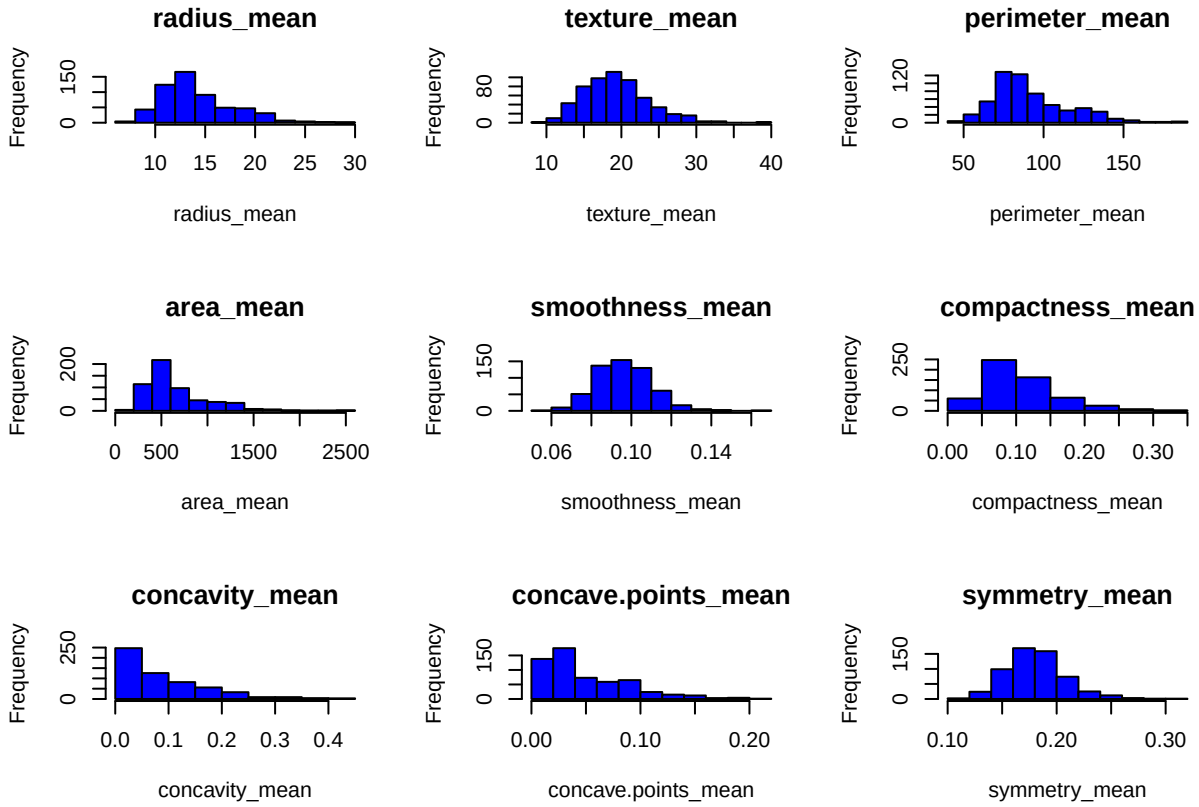
```
##
```

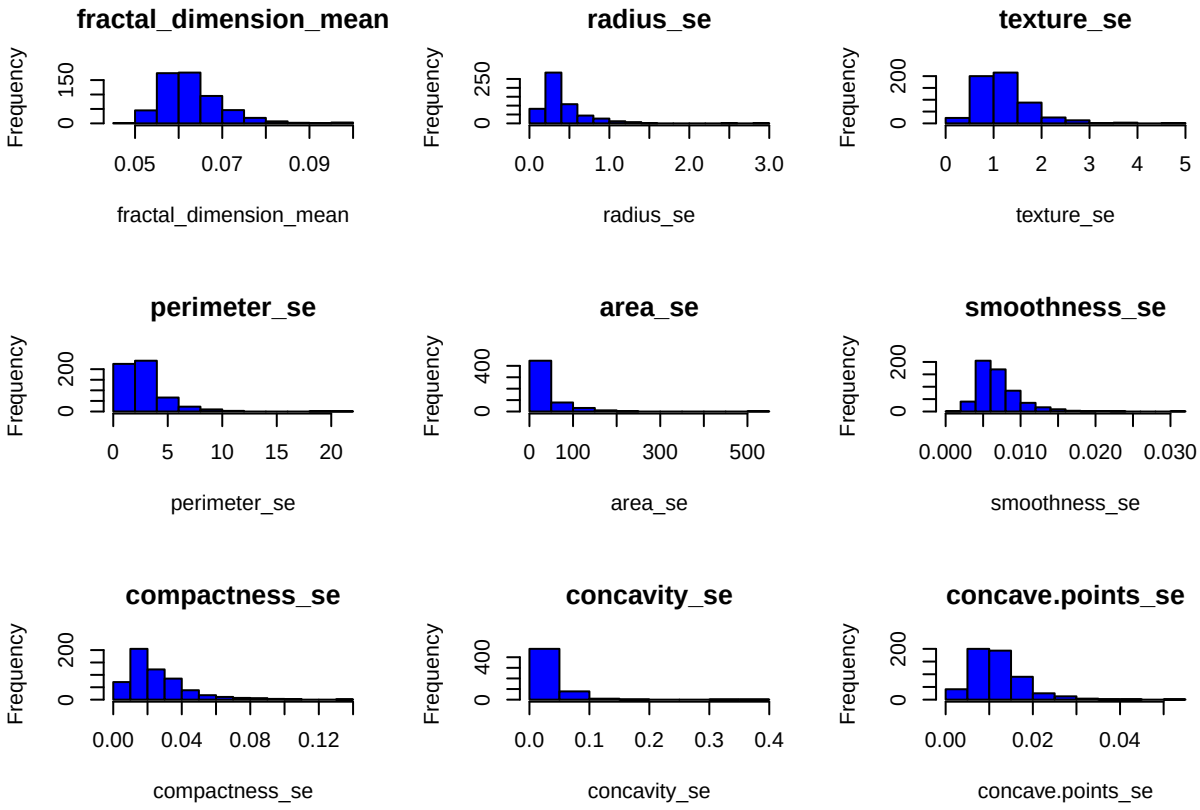
```
##         B         M
## 0.6274165 0.3725835
```

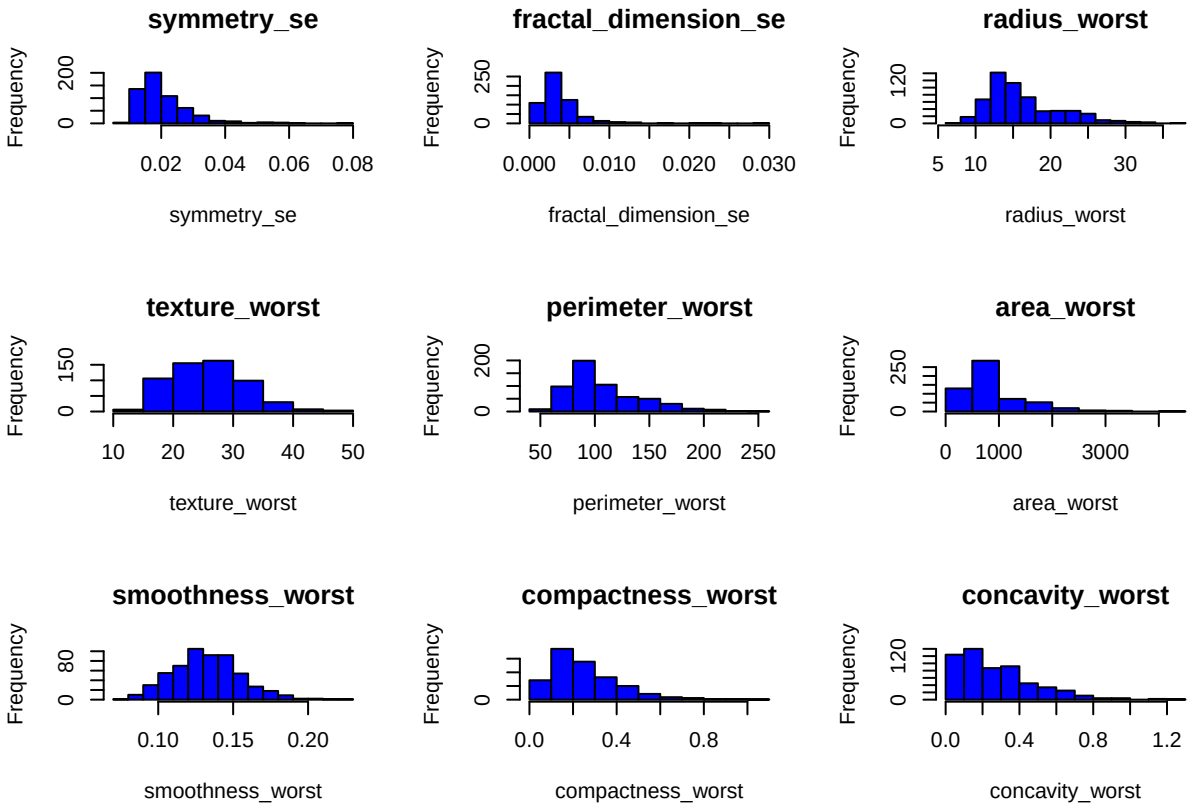
On observe que 63% des patientes ont un diagnostic bénin, et 37% un diagnostic malin. Il faut avoir cela en tête lorsque l'on étudiera notre jeu de données.

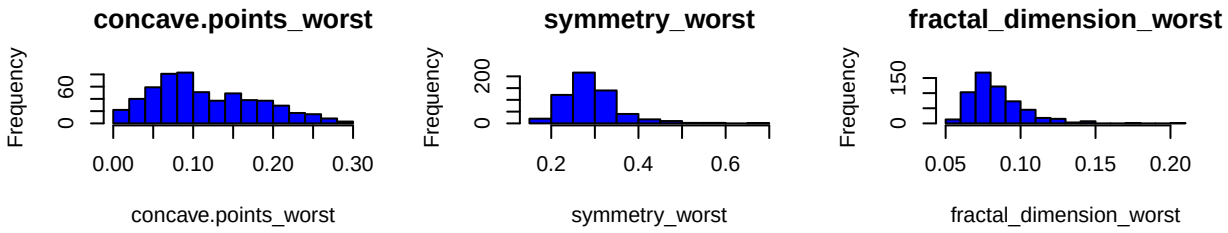
Distribution des variables

```
par(mfrow = c(3,3))
for(i in 2:31){
  hist(data[,i], main = colnames(data)[i], xlab = colnames(data)[i], col = "blue")
}
```




fractal_dimension_mean
Frequency
0
150
0.05
0.07
0.09
fractal_dimension_mean
radius_se
Frequency
0
250
0.0
1.0
2.0
3.0
radius_se
texture_se
Frequency
0
200
0
1
2
3
4
5
texture_se
perimeter_se
Frequency
0
200
0
5
10
15
20
perimeter_se
area_se
Frequency
0
400
0
100
300
500
area_se
smoothness_se
Frequency
0
200
0.000
0.010
0.020
0.030
smoothness_se
compactness_se
Frequency
0
200
0.00
0.04
0.08
0.12
compactness_se
concavity_se
Frequency
0
400
0.0
0.1
0.2
0.3
0.4
concavity_se
concave.points_se
Frequency
0
200
0.00
0.02
0.04
concave.points_se


symmetry_se
Frequency
0
200
0.02
0.04
0.06
0.08
symmetry_se
fractal_dimension_se
Frequency
0
250
0.000
0.010
0.020
0.030
fractal_dimension_se
radius_worst
Frequency
0
120
5
10
20
30
radius_worst
texture_worst
Frequency
0
150
10
20
30
40
50
texture_worst
perimeter_worst
Frequency
0
200
50
100
150
200
250
perimeter_worst
area_worst
Frequency
0
250
0
1000
3000
area_worst
smoothness_worst
Frequency
0
80
0.10
0.15
0.20
smoothness_worst
compactness_worst
Frequency
0
0.0
0.4
0.8
compactness_worst
concavity_worst
Frequency
0
120
0.0
0.4
0.8
1.2
concavity_worst



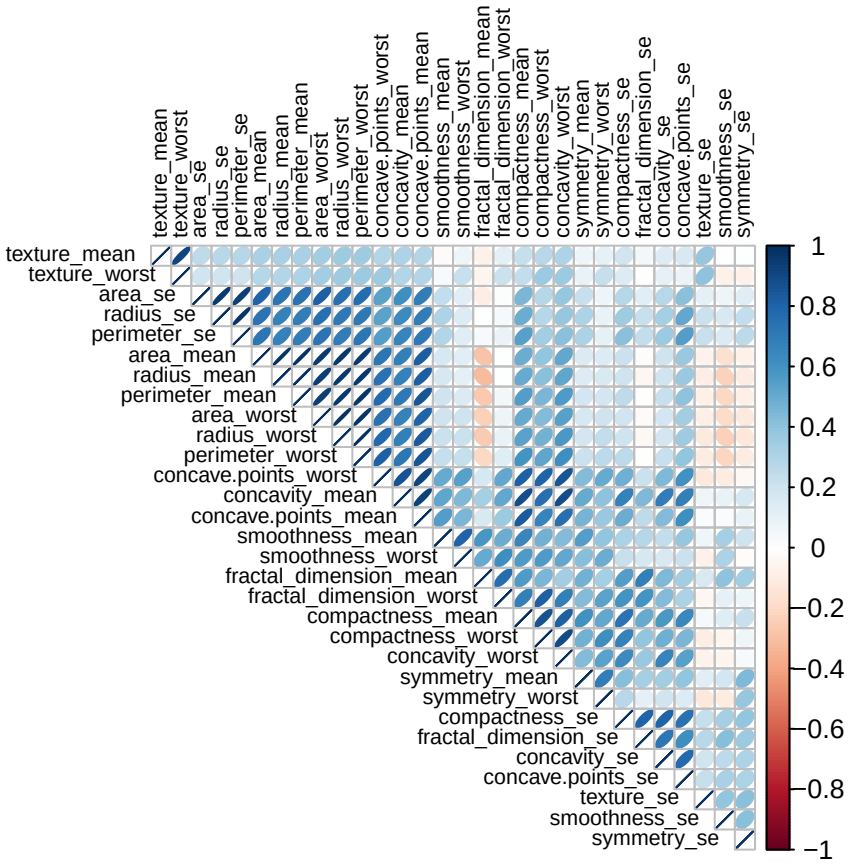
Matrice de corrélation

```
# install.packages("corrplot")
library("corrplot")
```

```
## corrplot 0.92 loaded
```

```
correlation <- cor(data[2:31])
```

```
corrplot(correlation, method = "ellipse", type = "upper", order = "hclust", tl.col = "black", tl.srt = 
```



On remarque que les variables sont très corrélées entre elles. Il faudra faire attention à la multicollinéarité lors de la modélisation.

On affiche les variables avec un coefficient de corrélation supérieur à 0.98.

```
# Fonction carrément volée sur internet https://rpubs.com/sediaz/Correlations
corr_check <- function(Dataset, threshold){
  matriz_cor <- cor(Dataset)
  matriz_cor

  for (i in 1:nrow(matriz_cor)){
    correlations <-  which((abs(matriz_cor[i,i:ncol(matriz_cor)]) > threshold) & (matriz_cor[i,i:ncol(ma

    if(length(correlations)> 0){
      lapply(correlations,FUN =  function(x) (cat(paste(colnames(Dataset)[i], "with",colnames(Dataset)[

    }
  }
}
```

```
corr_check(data[2:31], 0.98)
```

```
## radius_mean with perimeter_mean
## radius_mean with area_mean
## perimeter_mean with texture_mean
## radius_worst with perimeter_mean
## radius_worst with area_mean
```

On remarque que les colonnes liées sont le rayon, le périmètre, l'aire. On va supprimer le périmètre car, de part la forme circulaire des cancers, il peut être calculé comme étant 2 * pi * rayon. On devrait pouvoir l'observer dans les prochaines étapes de notre analyse.

Conclusion

Nous avons réalisé des statistiques descriptives sur notre jeu de données nettoyé. Nous avons pu observer la distribution des diagnostics, et des différentes variables. Nous avons également étudié la corrélation entre les variables, et avons identifié des variables fortement corrélées, qui devraient être prises en compte lors de la modélisation.

9.4 Code pour la partie ACP et AFD

Projet 4/5 ACP et AFD

Nicolas SALVAN - Alexandre CORRIOU

2024-05-24

Lecture des données nettoyées

Importation du dataset

```
data <- read.csv("data/data_cleaned.csv", header = TRUE, sep = ",")
data$diagnosis <- as.factor(data$diagnosis)
```

Aperçu rapide

```
head(data)
```

```
##   diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean
## 1         M       17.99        10.38         122.80    1001.0         0.11840
## 2         M       20.57        17.77         132.90    1326.0         0.08474
## 3         M       19.69        21.25         130.00    1203.0         0.10960
## 4         M       11.42        20.38          77.58     386.1         0.14250
## 5         M       20.29        14.34         135.10    1297.0         0.10030
## 6         M       12.45        15.70          82.57     477.1         0.12780
##   compactness_mean concavity_mean concave.points_mean symmetry_mean
## 1          0.27760         0.3001             0.14710        0.2419
## 2          0.07864         0.0869             0.07017        0.1812
## 3          0.15990         0.1974             0.12790        0.2069
## 4          0.28390         0.2414             0.10520        0.2597
## 5          0.13280         0.1980             0.10430        0.1809
## 6          0.17000         0.1578             0.08089        0.2087
##   fractal_dimension_mean radius_se texture_se perimeter_se area_se
## 1                0.07871    1.0950     0.9053        8.589  153.40
## 2                0.05667    0.5435     0.7339        3.398   74.08
## 3                0.05999    0.7456     0.7869        4.585   94.03
## 4                0.09744    0.4956     1.1560        3.445   27.23
## 5                0.05883    0.7572     0.7813        5.438   94.44
## 6                0.07613    0.3345     0.8902        2.217   27.19
##   smoothness_se compactness_se concavity_se concave.points_se symmetry_se
## 1      0.006399        0.04904      0.05373           0.01587     0.03003
## 2      0.005225        0.01308      0.01860           0.01340     0.01389
## 3      0.006150        0.04006      0.03832           0.02058     0.02250
## 4      0.009110        0.07458      0.05661           0.01867     0.05963
```

```
## 5      0.011490        0.02461      0.05688           0.01885     0.01756
## 6      0.007510        0.03345      0.03672           0.01137     0.02165
##   fractal_dimension_se radius_worst texture_worst perimeter_worst area_worst
## 1             0.006193        25.38         17.33          184.60     2019.0
## 2             0.003532        24.99         23.41          158.80     1956.0
## 3             0.004571        23.57         25.53          152.50     1709.0
## 4             0.009208        14.91         26.50           98.87      567.7
## 5             0.005115        22.54         16.67          152.20     1575.0
## 6             0.005082        15.47         23.75          103.40      741.6
##   smoothness_worst compactness_worst concavity_worst concave.points_worst
## 1           0.1622            0.6656          0.7119               0.2654
## 2           0.1238            0.1866          0.2416               0.1860
## 3           0.1444            0.4245          0.4504               0.2430
## 4           0.2098            0.8663          0.6869               0.2575
## 5           0.1374            0.2050          0.4000               0.1625
## 6           0.1791            0.5249          0.5355               0.1741
##   symmetry_worst fractal_dimension_worst
## 1         0.4601                 0.11890
## 2         0.2750                 0.08902
## 3         0.3613                 0.08758
## 4         0.6638                 0.17300
## 5         0.2364                 0.07678
## 6         0.3985                 0.12440
```

```
# dim(data)
# str(data)
```

ACP - Analyse en Composantes Principales

Lancement d'une ACP sur les données

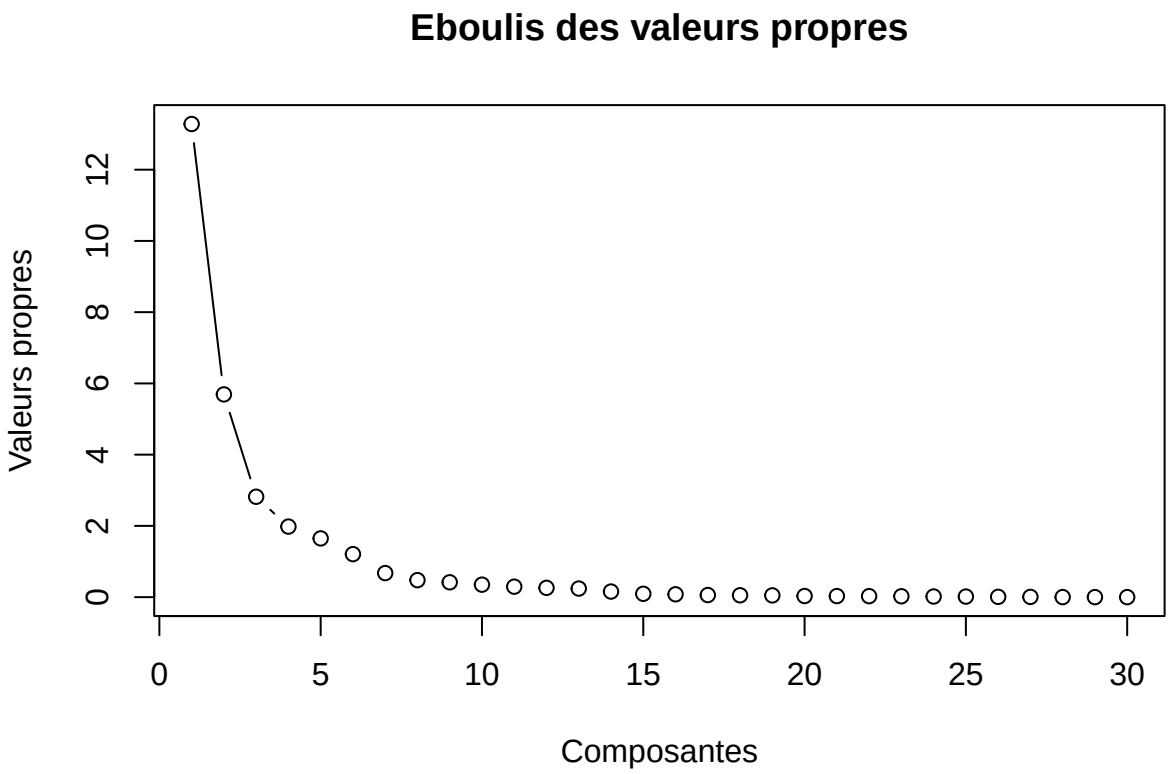
Nous allons réaliser une ACP sur nos données avec la bibliothèque `FactoMineR`.

```
# install.packages("FactoMineR")
library("FactoMineR")
```

```
res.pca <- PCA(data, scale.unit = TRUE, graph = FALSE, quali.sup = 1)
```

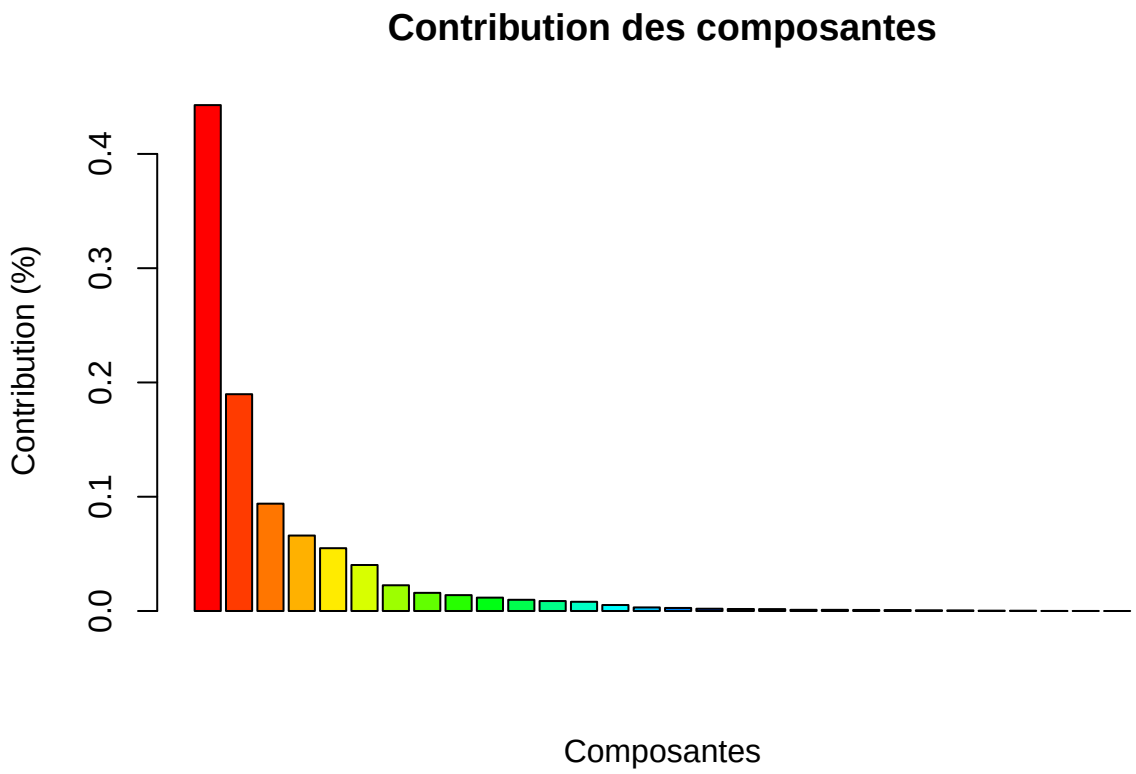
Choix du nombre de composantes

```
plot(res.pca$eig[,1], type = "b", xlab = "Composantes", ylab = "Valeurs propres", main = "Eboulis des va
```



On observe que les composantes principales sont les trois premières. On peut le vérifier en affichant leur contribution.

```
eig_percentage = res.pca$eig[,2]/sum(res.pca$eig[,2])
barplot(eig_percentage, names.arg = FALSE, col = rainbow(26), main = "Contribution des composantes", xl
```




```
eig_percentage[1:3]
```

```
##     comp 1     comp 2     comp 3
## 0.44272026 0.18971182 0.09393163
```

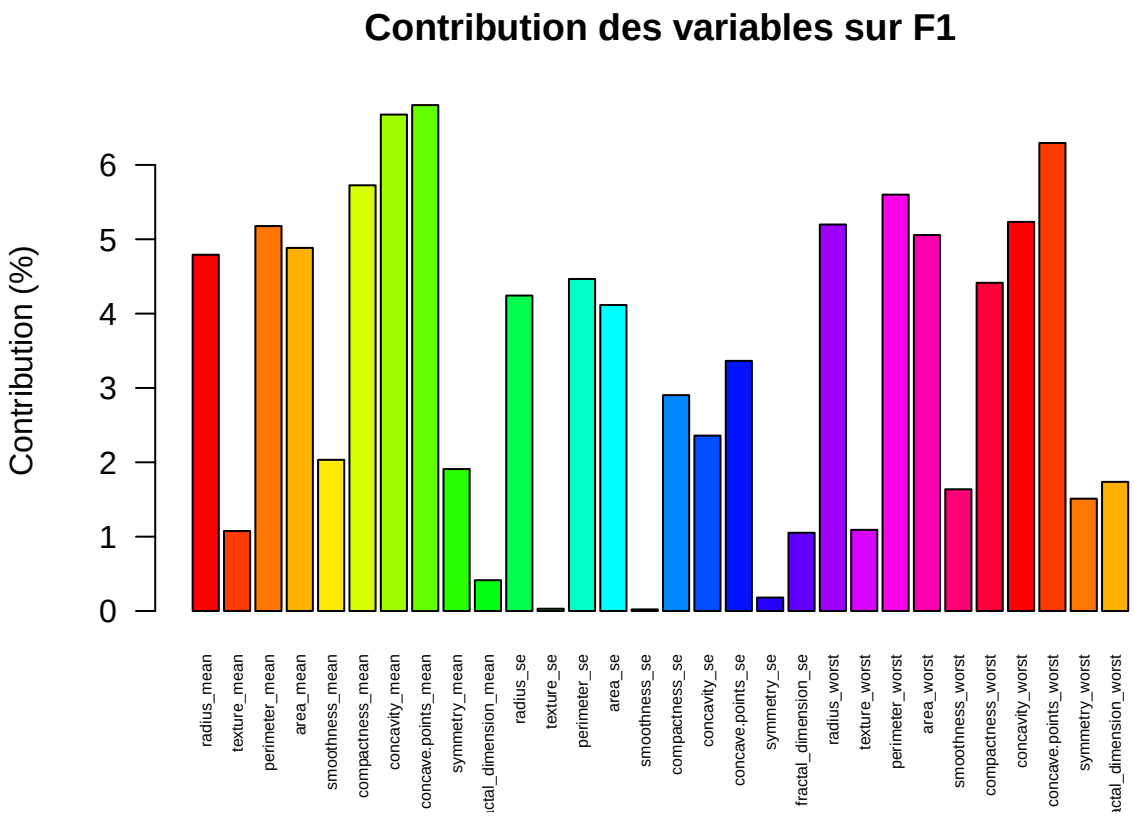
```
sum(eig_percentage[1:3])
```

```
## [1] 0.7263637
```

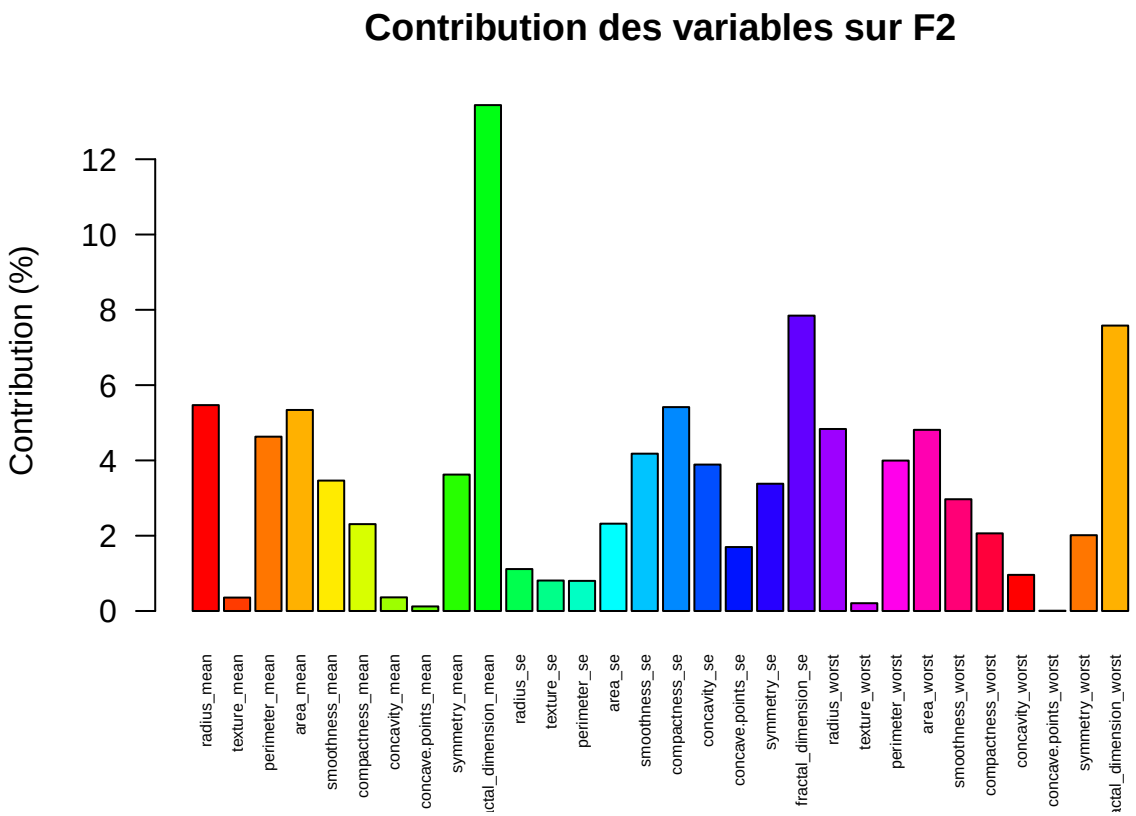
Les trois premières composantes représentent 72.6% de l'information.

Affichage des variables ayant le plus d'influence sur les axes

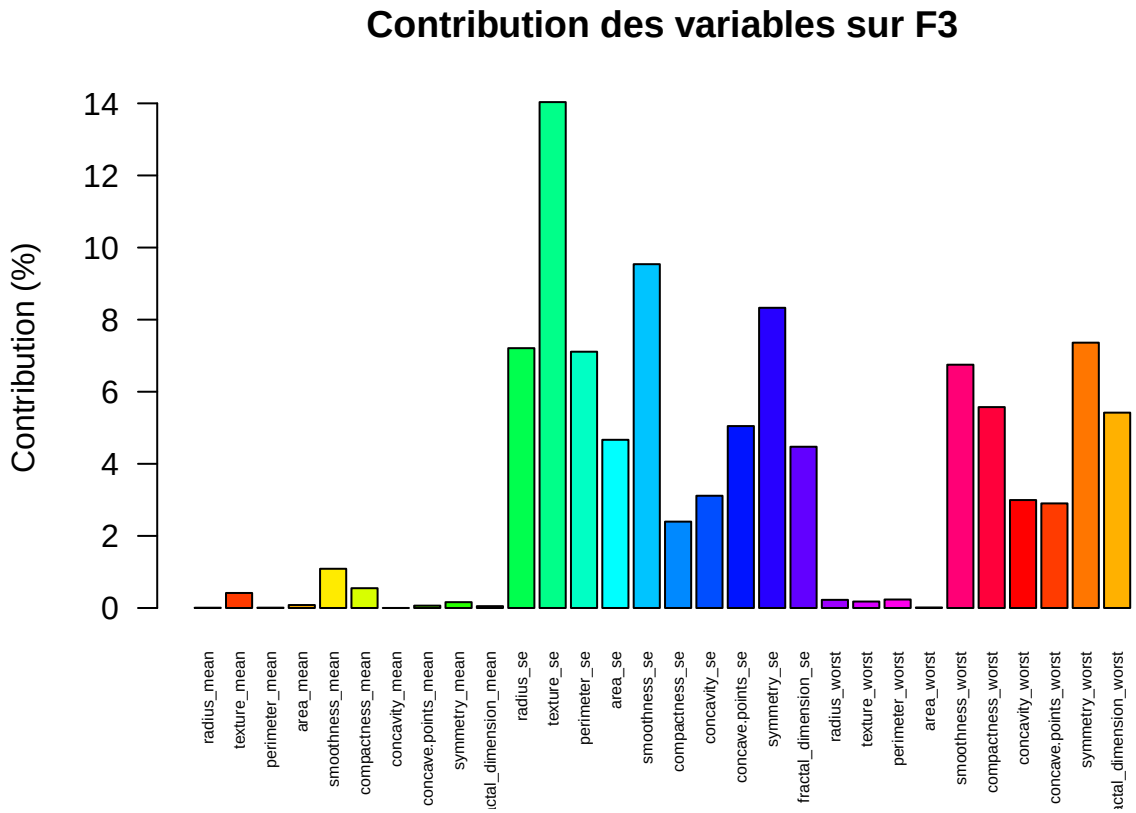
```
res.pca.contrib <- res.pca$var$contrib[, 1:3]
barplot(res.pca.contrib[,1], names.arg = rownames(res.pca$var$contrib), col = rainbow(26), main = "Cont
```




```
barplot(res.pca.contrib[,2], names.arg = rownames(res.pca$var$contrib), col = rainbow(26), main = "Cont
```



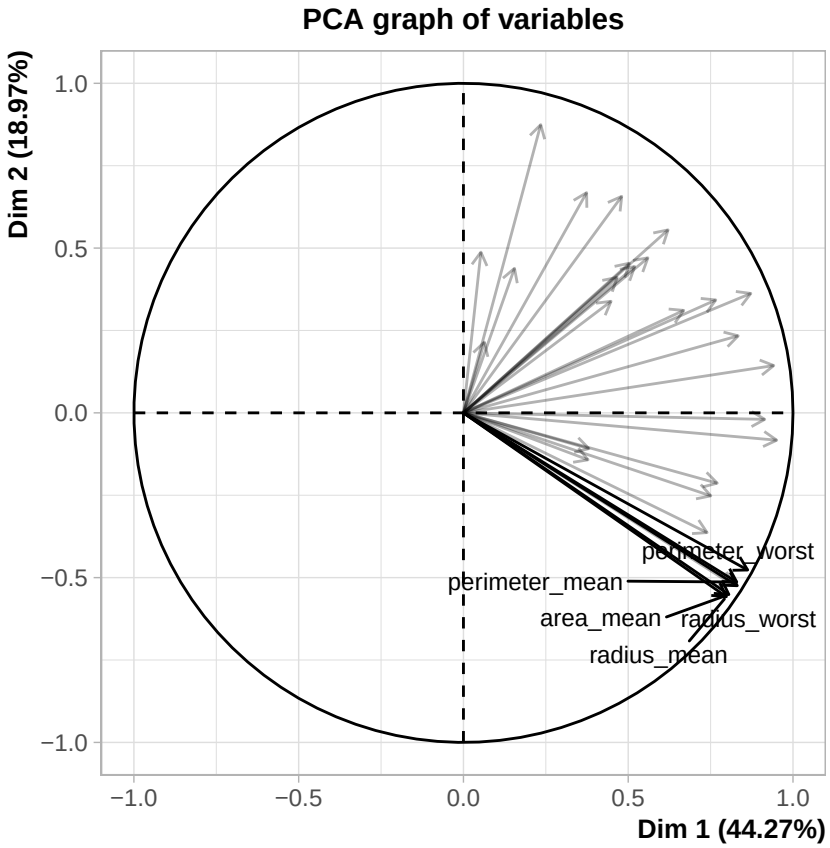

```
barplot(res.pca.contrib[,3], names.arg = rownames(res.pca$var$contrib), col = rainbow(26), main = "Cont
```



On observe ici les variables qui contribuent le plus dans les plans principaux de l'ACP.

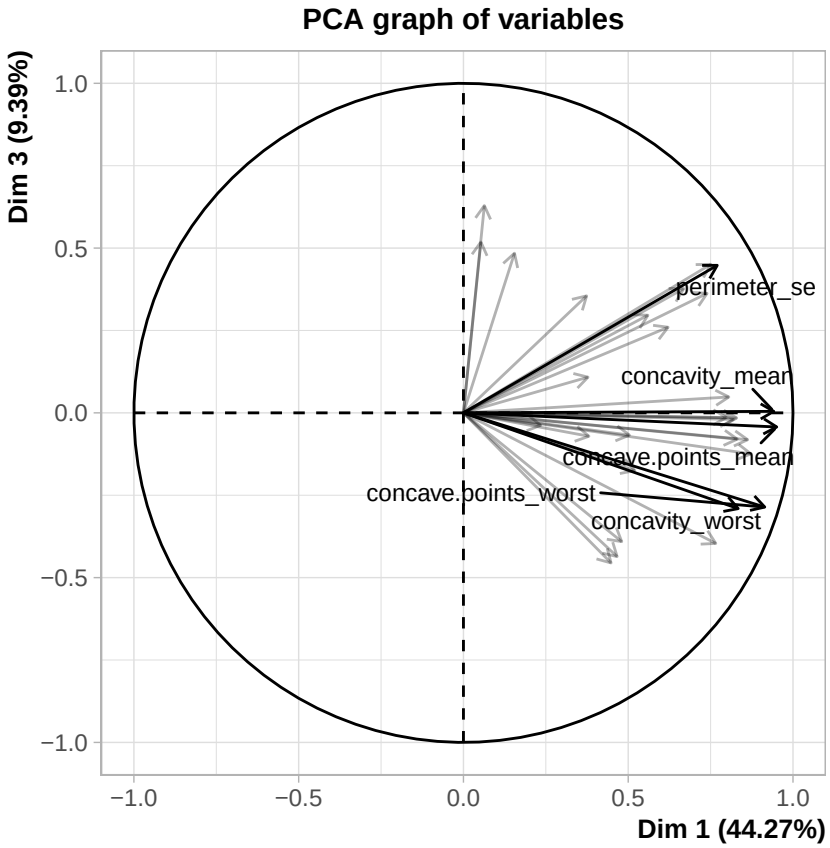
Plan (F1, F2)

```
plot(res.pca, choix = "var", cex = 0.8, col.var = "black", select = "contrib 5")
```



Plan (F1, F3)

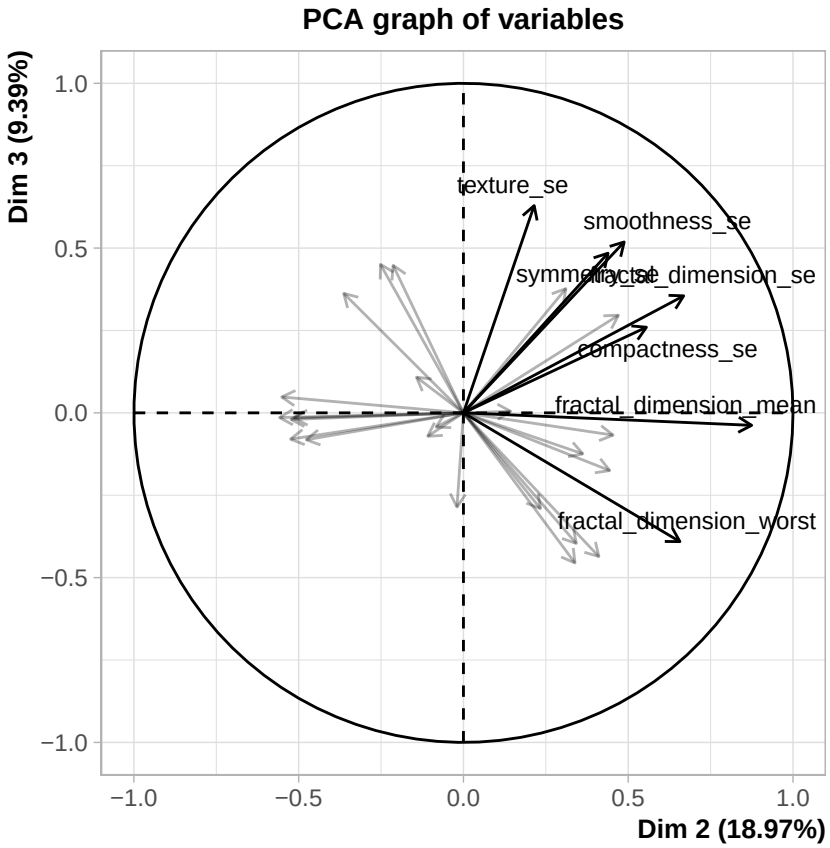
```
plot(res.pca, choix = "var", cex = 0.8, col.var = "black", select = "contrib 5", axes = c(1,3))
```



On obtient grâce à ce graphe les variables qui contribuent le plus dans ce plan.

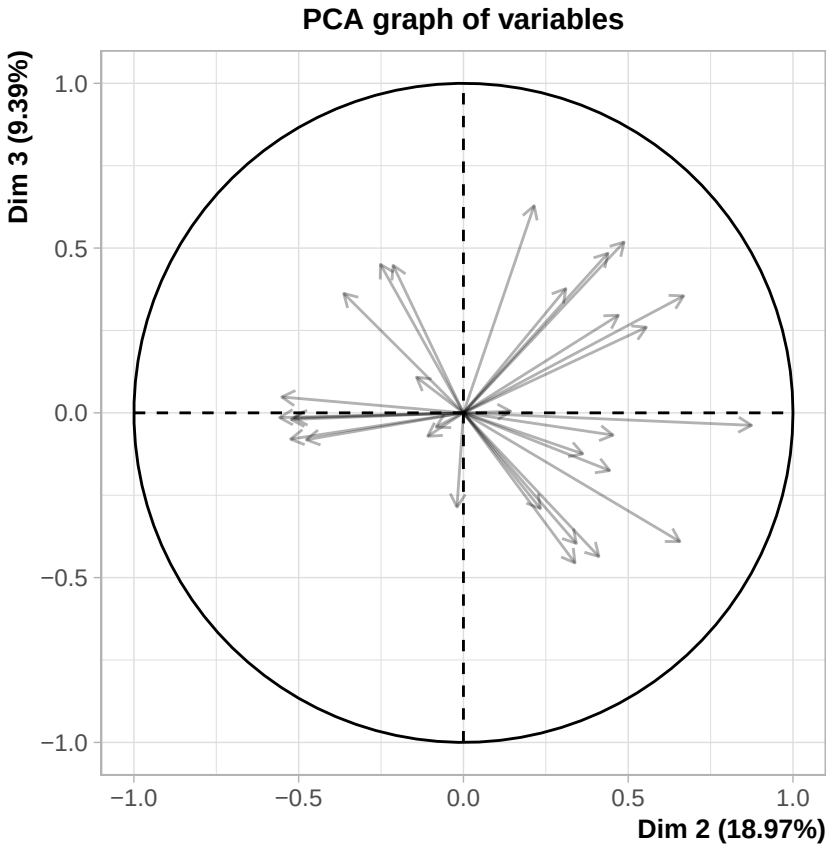
Plan (F2, F3)

```
plot(res.pca, choix = "var", cex = 0.8, col.var = "black", select = "contrib 7", axes = c(2,3))
```



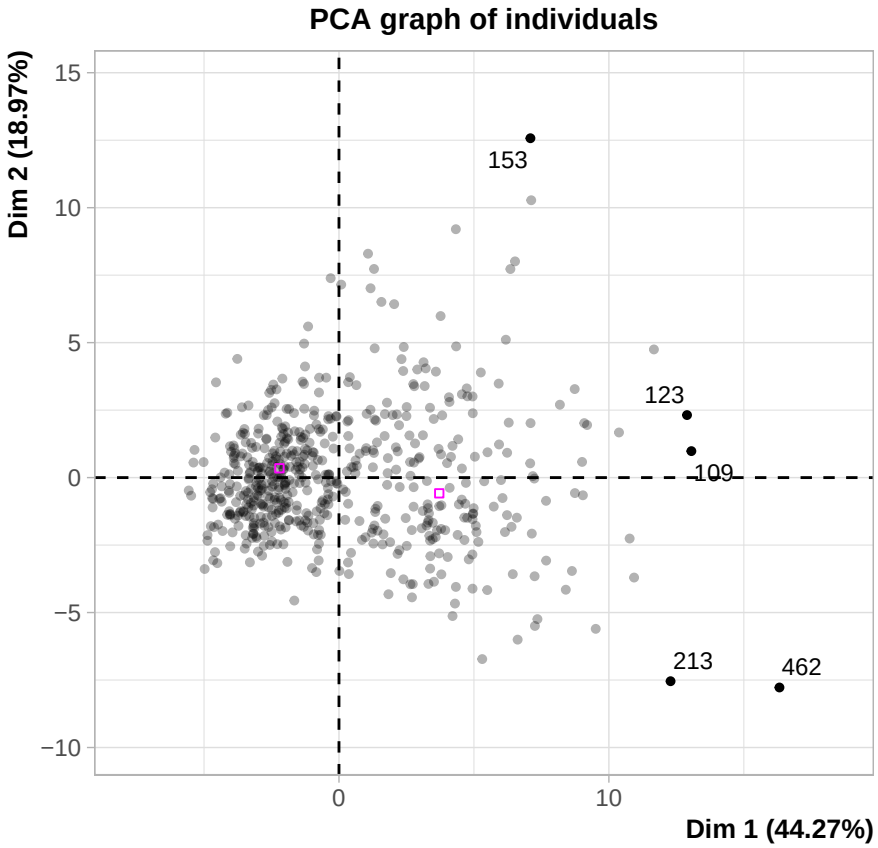
On observe que dans ce plan, les variables ne sont pas très bien représentées. On peut le voir en affichant celles avec un cos2 supérieur à 0.8 (aucune).

```
plot(res.pca, choix = "var", cex = 0.8, col.var = "black", select = "cos2 .8", axes = c(2,3))
```



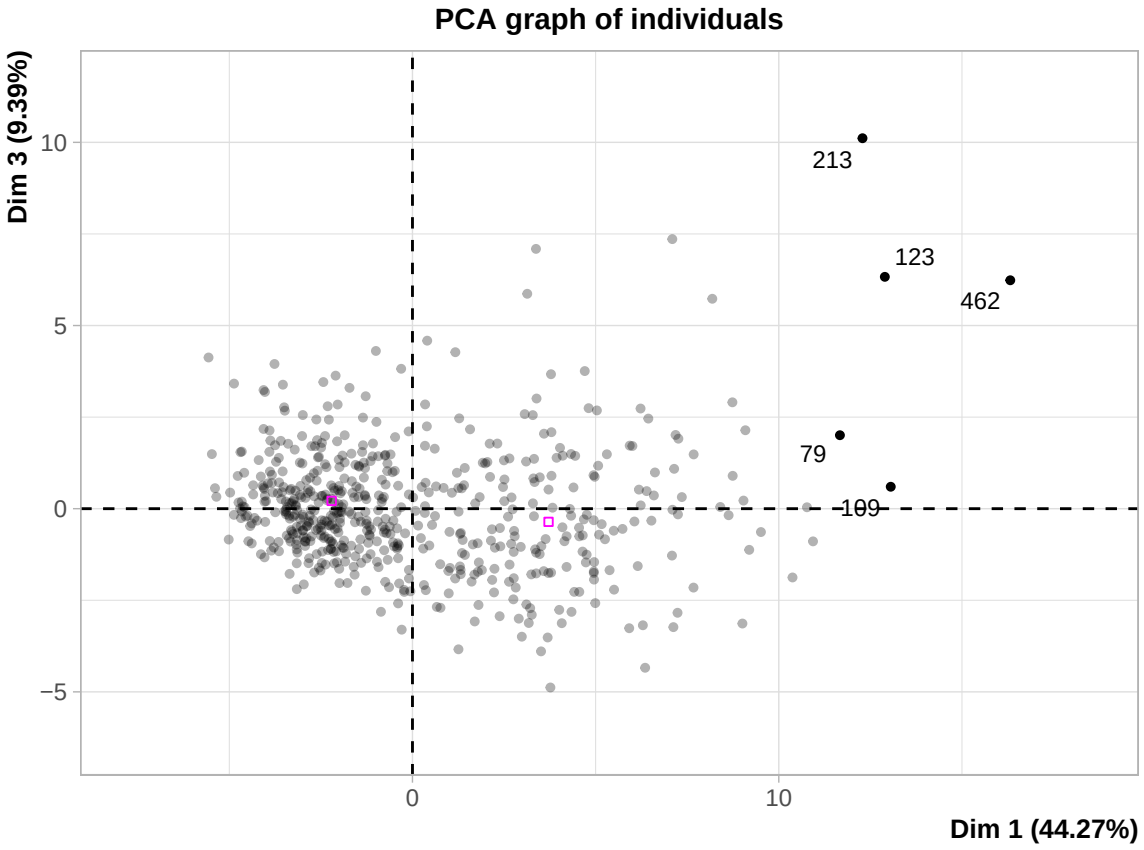
Représentation des individus

```
plot(res.pca, choix = "ind", cex = 0.8, col.ind = "black", select = "contrib 5", label = "ind")
```

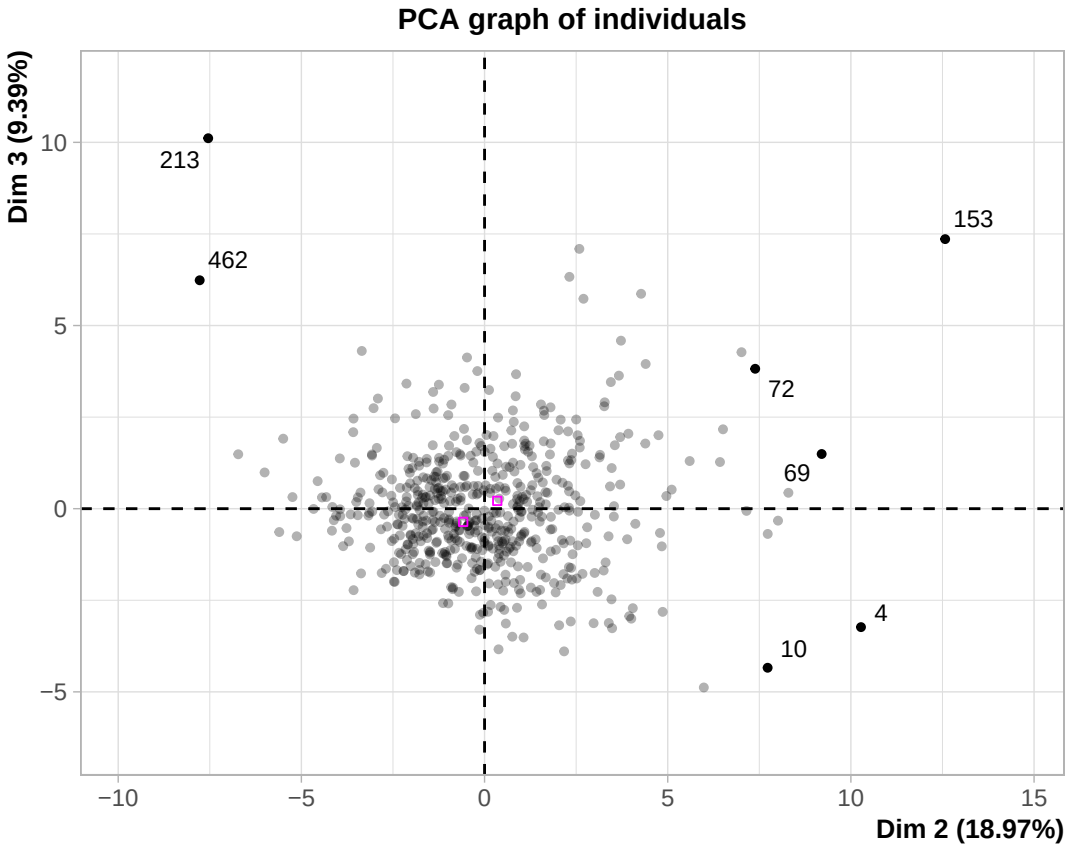


On observe que certains individus contribuent beaucouo dans ce plan. Regardons si ces individus ont autant d'influence sur les autres plans (F1, F3) et (F2, F3).

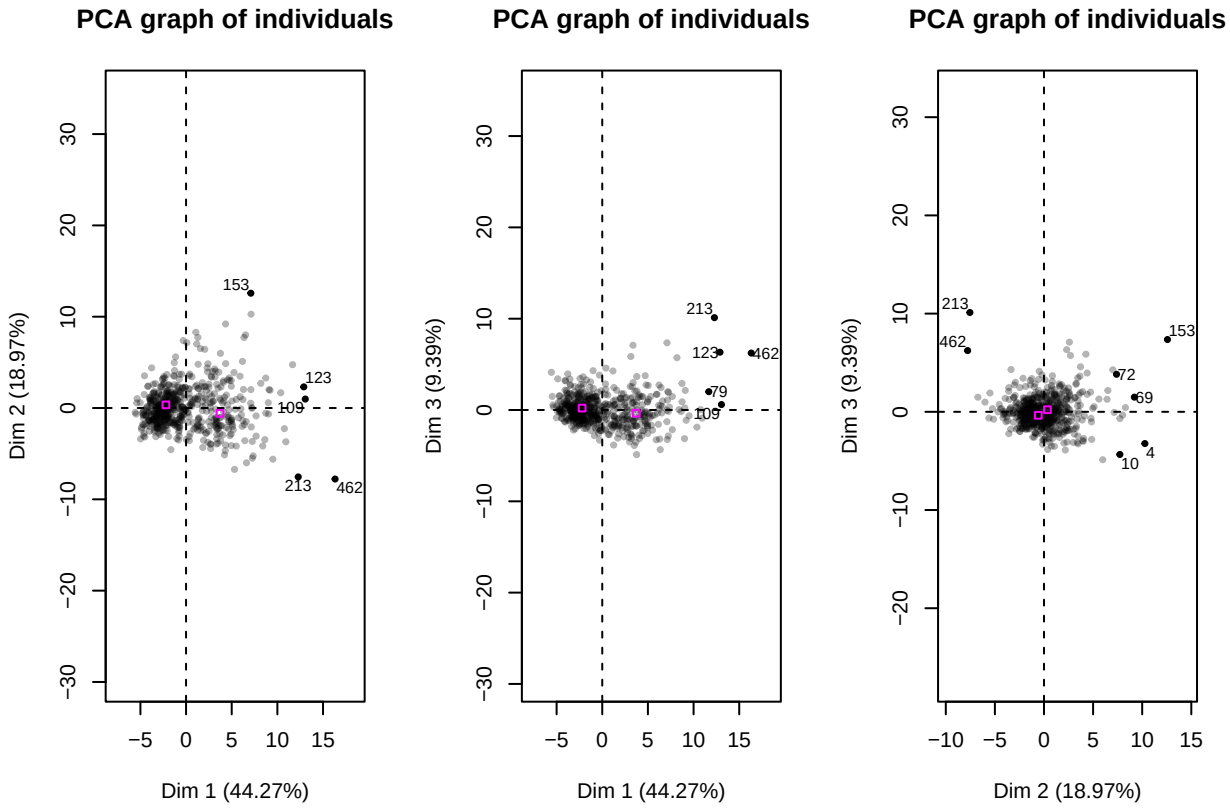
```
plot(res.pca, choix = "ind", cex = 0.8, col.ind = "black", select = "contrib 5", axes = c(1,3), label =
```




```
plot(res.pca, choix = "ind", cex = 0.8, col.ind = "black", select = "contrib 7", axes = c(2,3), label =
```




```
par(mfrow=c(1,3))

plot(res.pca, graph.type = "classic", choix = "ind", cex = 0.8, col.ind = "black", select = "contrib 5"

plot(res.pca, choix = "ind", graph.type = "classic", cex = 0.8, col.ind = "black", select = "contrib 5"

plot(res.pca, choix = "ind", graph.type = "classic", cex = 0.8, col.ind = "black", select = "contrib 7"
```



Les individus 462, 213, 123 sont très influents sur les plans (F1, F2) et (F1, F3). Il serait intéressant de les étudier plus en détail.

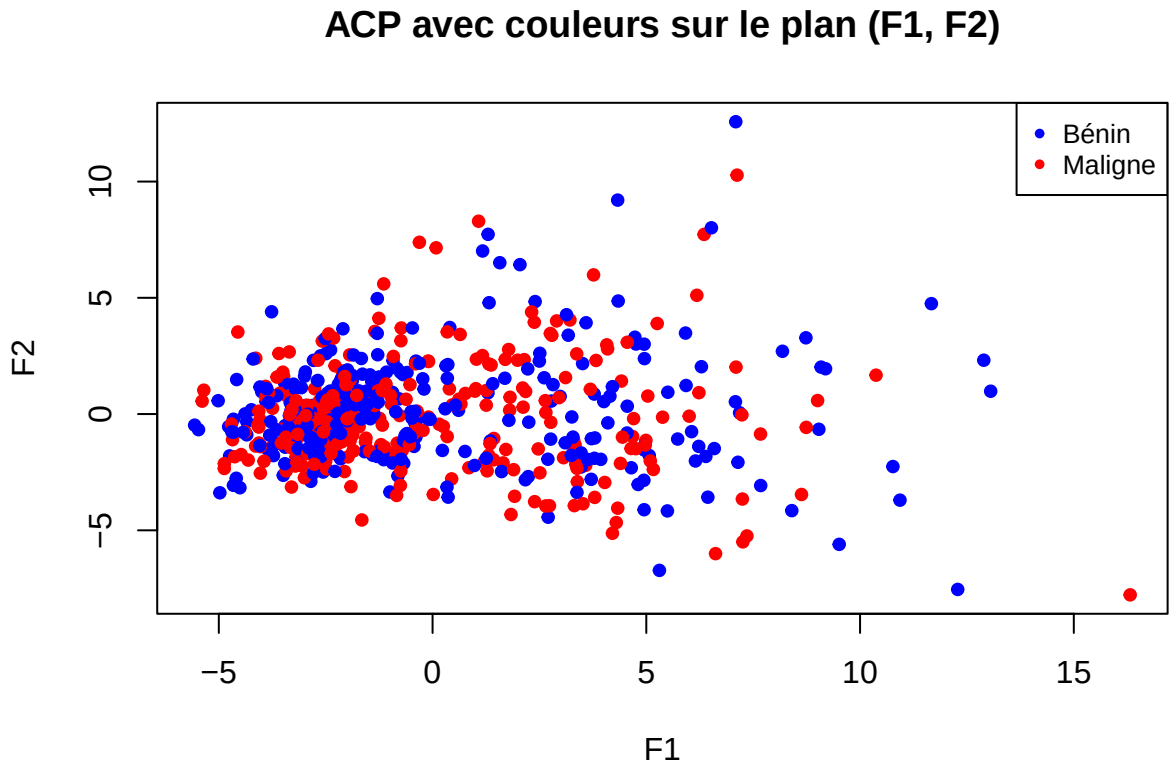
```
data[c(462, 213, 123),]
```

```
##     diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean
## 462         M       27.42        26.27          186.9      2501          0.1084
## 213         M       28.11        18.47          188.5      2499          0.1142
## 123         M       24.25        20.20          166.2      1761          0.1447
##     compactness_mean concavity_mean concave.points_mean symmetry_mean
## 462           0.1988         0.3635              0.1689        0.2061
## 213           0.1516         0.3201              0.1595        0.1648
## 123           0.2867         0.4268              0.2012        0.2655
##     fractal_dimension_mean radius_se texture_se perimeter_se area_se
## 462                0.05623     2.547      1.306       18.650   542.2
## 213                0.05525     2.873      1.476       21.980   525.6
## 123                0.06877     1.509      3.120        9.807   233.0
##     smoothness_se compactness_se concavity_se concave.points_se symmetry_se
## 462       0.00765        0.05374      0.08055           0.02598     0.01697
## 213       0.01345        0.02772      0.06389           0.01407     0.04783
## 123       0.02333        0.09806      0.12780           0.01822     0.04547
##     fractal_dimension_se radius_worst texture_worst perimeter_worst area_worst
## 462             0.004558        36.04         31.37           251.2       4254
## 213             0.004476        28.11         18.47           188.5       2499
## 123             0.009875        26.02         23.99           180.9       2073
##     smoothness_worst compactness_worst concavity_worst concave.points_worst
```

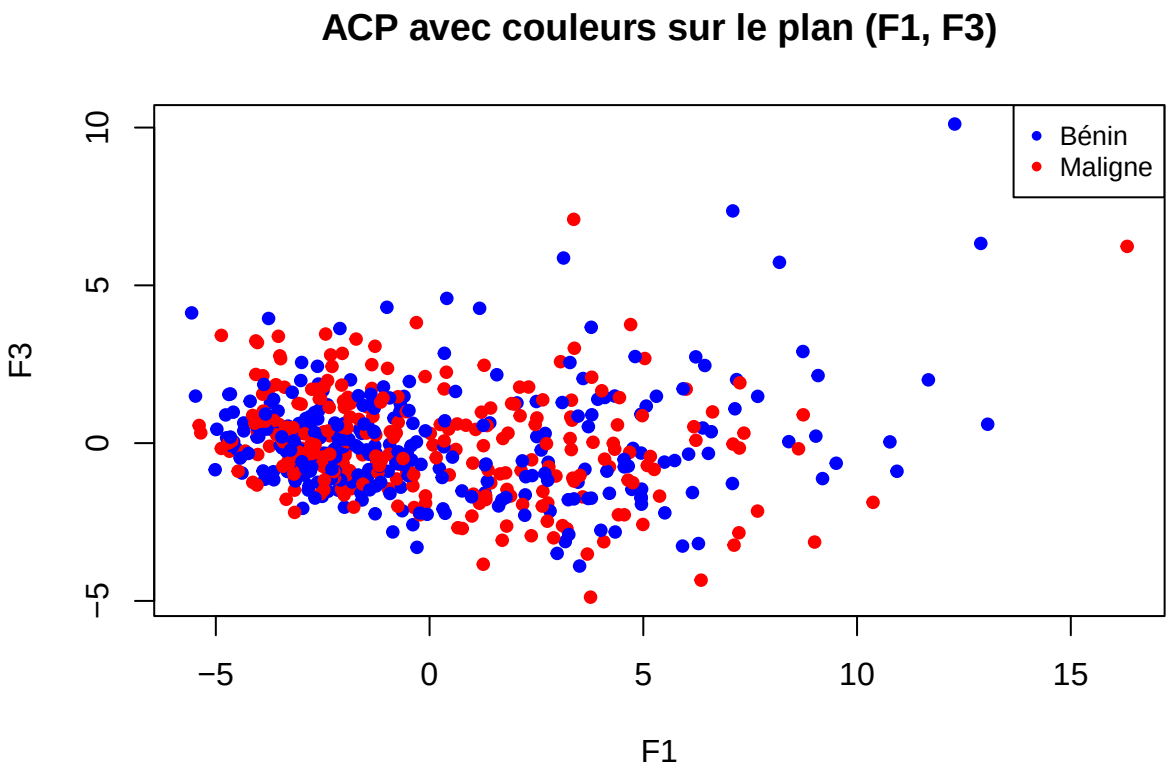
```
## 462           0.1357            0.4256          0.6833               0.2625
## 213           0.1142            0.1516          0.3201               0.1595
## 123           0.1696            0.4244          0.5803               0.2248
##     symmetry_worst fractal_dimension_worst
## 462         0.2641                 0.07427
## 213         0.1648                 0.05525
## 123         0.3222                 0.08009
```

Représentation des individus labelisés sur les plans principaux

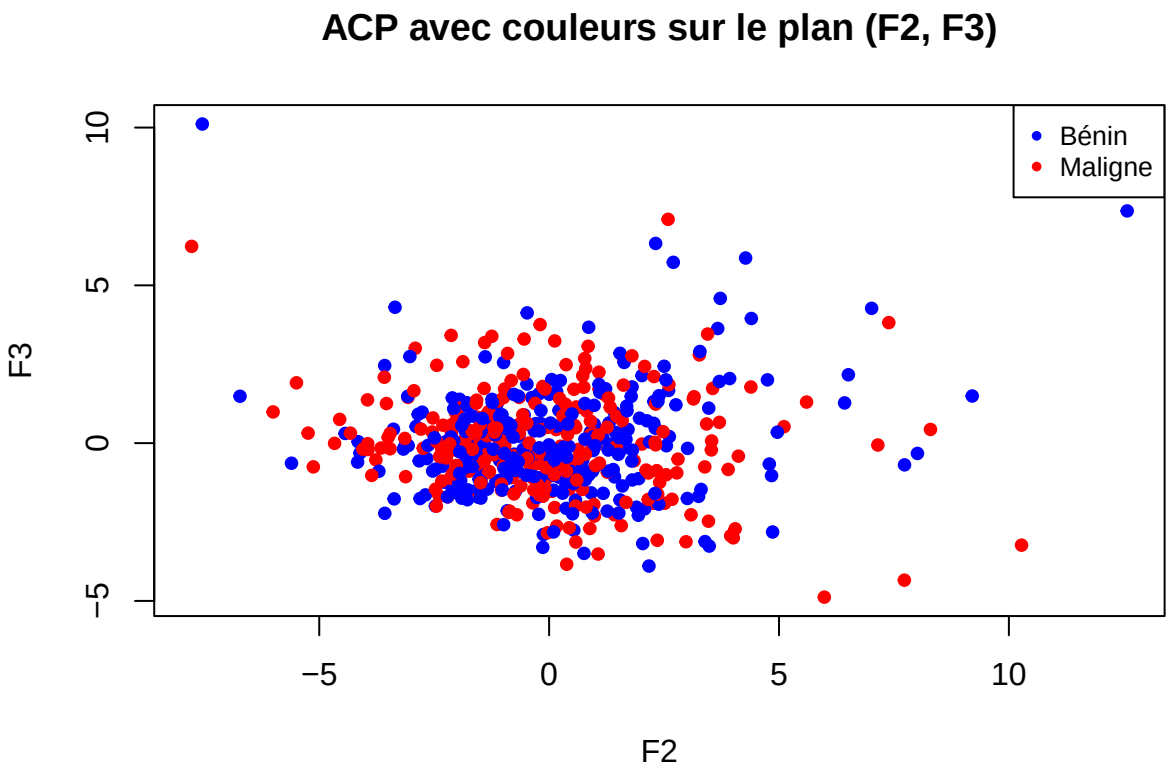
```
plot(res.pca$ind$coord[,1], res.pca$ind$coord[,2], col = c("blue", "red"), pch = 20, main = "ACP avec c
legend("topright", legend = c("Bénin", "Maligne"), col = c("blue", "red"), pch = 20, cex = 0.8)
```




```
plot(res.pca$ind$coord[,1], res.pca$ind$coord[,3], col = c("blue", "red"), pch = 20, main = "ACP avec c
legend("topright", legend = c("Bénin", "Maligne"), col = c("blue", "red"), pch = 20, cex = 0.8)
```



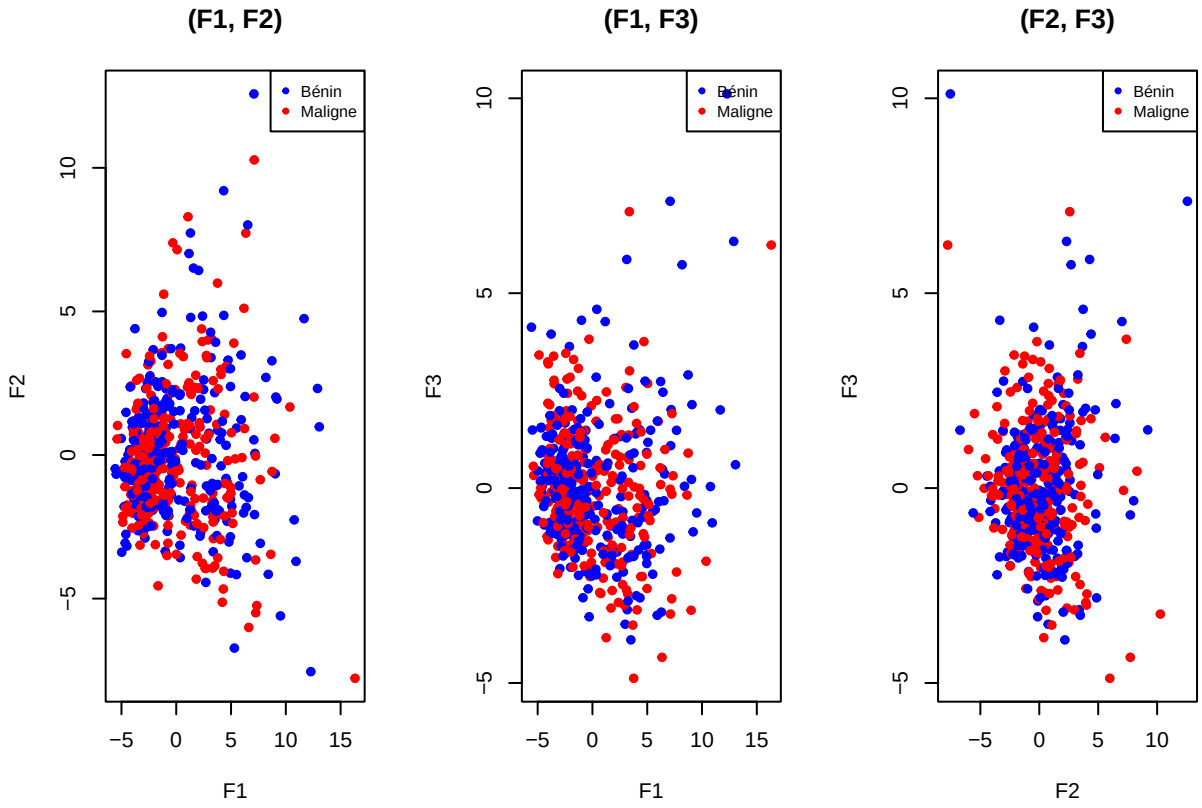

```
plot(res.pca$ind$coord[,2], res.pca$ind$coord[,3], col = c("blue", "red"), pch = 20, main = "ACP avec c
legend("topright", legend = c("Bénin", "Maligne"), col = c("blue", "red"), pch = 20, cex = 0.8)
```



On s'aperçoit qu'il est très difficile de distinguer les individus sur les plans principaux de l'ACP, d'où la nécessité de réaliser une AFD.

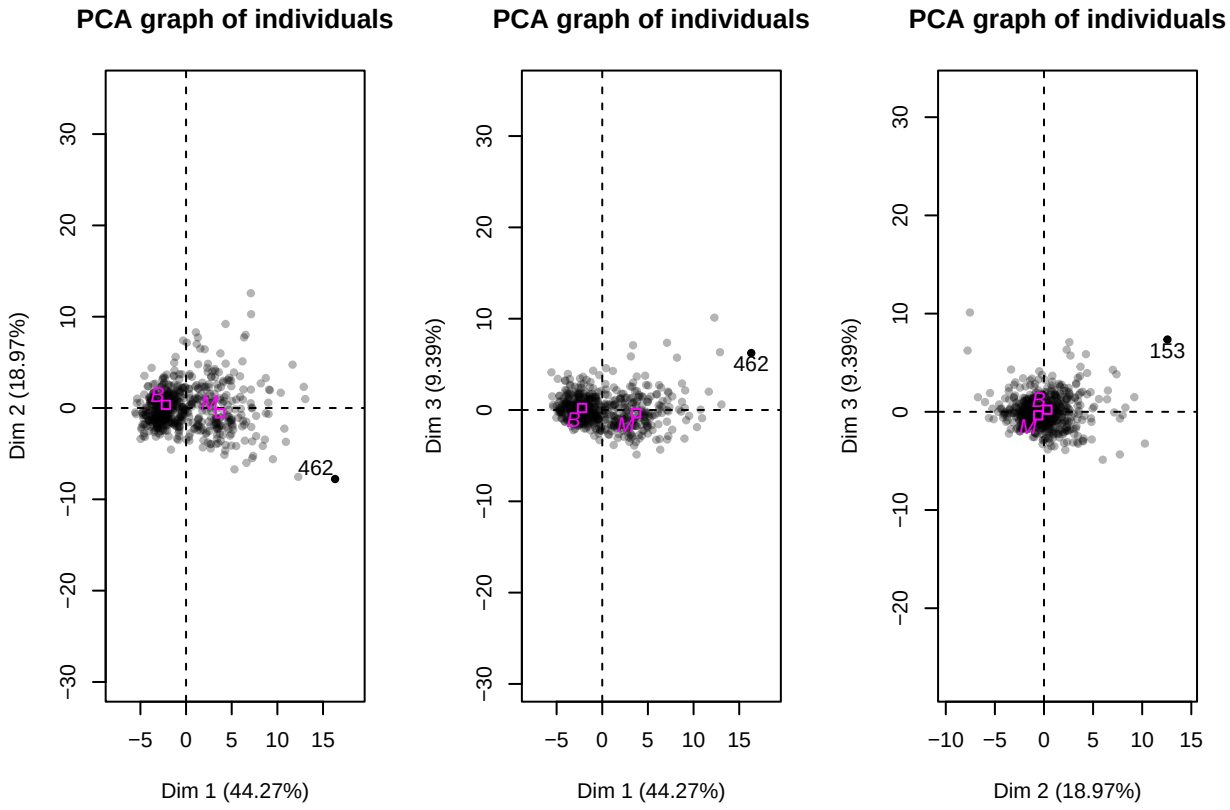
On peut représenter ces trois représentations ci-dessus sur un seul graphique.

```
par(mfrow=c(1,3))

plot(res.pca$ind$coord[,1], res.pca$ind$coord[,2], col = c("blue", "red"), pch = 20, main = "(F1, F2)",
legend("topright", legend = c("Bénin", "Maligne"), col = c("blue", "red"), pch = 20, cex = 0.8)

plot(res.pca$ind$coord[,1], res.pca$ind$coord[,3], col = c("blue", "red"), pch = 20, main = "(F1, F3)",
legend("topright", legend = c("Bénin", "Maligne"), col = c("blue", "red"), pch = 20, cex = 0.8)

plot(res.pca$ind$coord[,2], res.pca$ind$coord[,3], col = c("blue", "red"), pch = 20, main = "(F2, F3)",
legend("topright", legend = c("Bénin", "Maligne"), col = c("blue", "red"), pch = 20, cex = 0.8)
```



Représentation des catégories sur les plans principaux

```
par(mfrow=c(1,3))
plot(res.pca, graph= "classic", choix = "ind", cex = 1, select = "contrib 0")
plot(res.pca, graph= "classic", choix = "ind", cex = 1, select = "contrib 0", axes = c(1,3))
plot(res.pca, graph= "classic", choix = "ind", cex = 1, select = "contrib 0", axes = c(2,3))
```



On observe que l'axe 1 est le plus discriminant par rapport aux centres de gravité des groupes. En effet, les centres des catégories sont bien séparés le long de cet axe. Néanmoins, le manque de séparation visible sur les plans (F1, F2), (F1, F3) et (F2, F3) nous pousse à réaliser une AFD.

AFD - Analyse Factorielle discriminante

Lancement d'une AFD sur les données

Nous allons réaliser une AFD sur nos données avec la bibliothèque `MASS`.

```
library(MASS)
```

```
res.afd <- lda(data$diagnosis ~ ., data = data)
res.afd
```

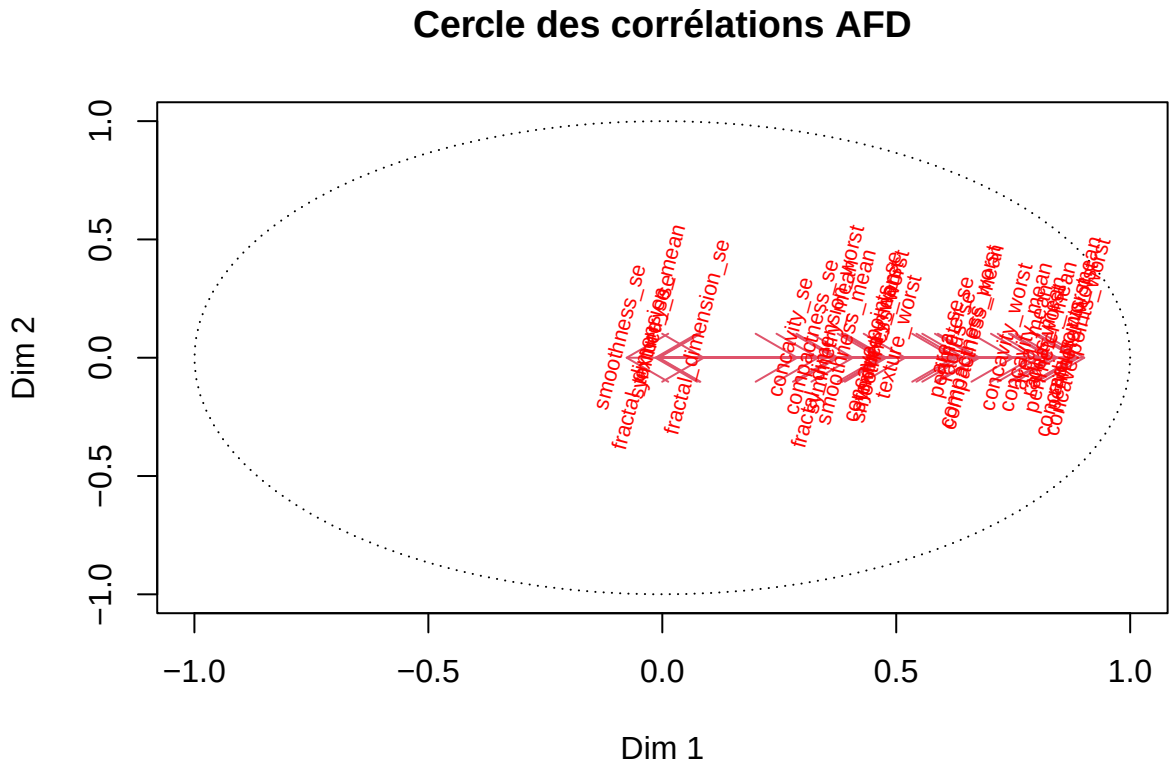
```
## Call:
## lda(data$diagnosis ~ ., data = data)
##
## Prior probabilities of groups:
##         B         M
## 0.6274165 0.3725835
##
## Group means:
##   radius_mean texture_mean perimeter_mean area_mean smoothness_mean
```

```
## B    12.14652     17.91476       78.07541  462.7902      0.09247765
## M    17.46283     21.60491      115.36538  978.3764      0.10289849
##   compactness_mean concavity_mean concave.points_mean symmetry_mean
## B       0.08008462     0.04605762          0.02571741      0.174186
## M       0.14518778     0.16077472          0.08799000      0.192909
##   fractal_dimension_mean radius_se texture_se perimeter_se  area_se
## B             0.06286739 0.2840824   1.220380     2.000321 21.13515
## M             0.06268009 0.6090825   1.210915     4.323929 72.67241
##   smoothness_se compactness_se concavity_se concave.points_se symmetry_se
## B   0.007195902     0.02143825   0.02599674       0.009857653  0.02058381
## M   0.006780094     0.03228117   0.04182401       0.015060472  0.02047240
##   fractal_dimension_se radius_worst texture_worst perimeter_worst area_worst
## B          0.003636051     13.37980      23.51507        87.00594   558.8994
## M          0.004062406     21.13481      29.31821       141.37033  1422.2863
##   smoothness_worst compactness_worst concavity_worst concave.points_worst
## B        0.1249595         0.1826725       0.1662377           0.07444434
## M        0.1448452         0.3748241       0.4506056           0.18223731
##   symmetry_worst fractal_dimension_worst
## B      0.2702459              0.07944207
## M      0.3234679              0.09152995
##
## Coefficients of linear discriminants:
##                                  LD1
## radius_mean              -1.075583600
## texture_mean              0.022450225
## perimeter_mean            0.117251982
## area_mean                 0.001569797
## smoothness_mean           0.418282533
## compactness_mean        -20.852775912
## concavity_mean            6.904756198
## concave.points_mean      10.578586272
## symmetry_mean             0.507284238
## fractal_dimension_mean    0.164280222
## radius_se                 2.148262164
## texture_se               -0.033380325
## perimeter_se             -0.111228320
## area_se                  -0.004559805
## smoothness_se            78.305030179
## compactness_se            0.320560148
## concavity_se            -17.609967822
## concave.points_se        52.195471457
## symmetry_se               8.383223501
## fractal_dimension_se    -35.296511336
## radius_worst              0.964016085
## texture_worst             0.035360398
## perimeter_worst          -0.012026798
## area_worst               -0.004994466
## smoothness_worst          2.681188528
## compactness_worst         0.331697102
## concavity_worst           1.882716394
## concave.points_worst      2.293242388
## symmetry_worst            2.749992654
## fractal_dimension_worst  21.255049570
```

Représentation des variables

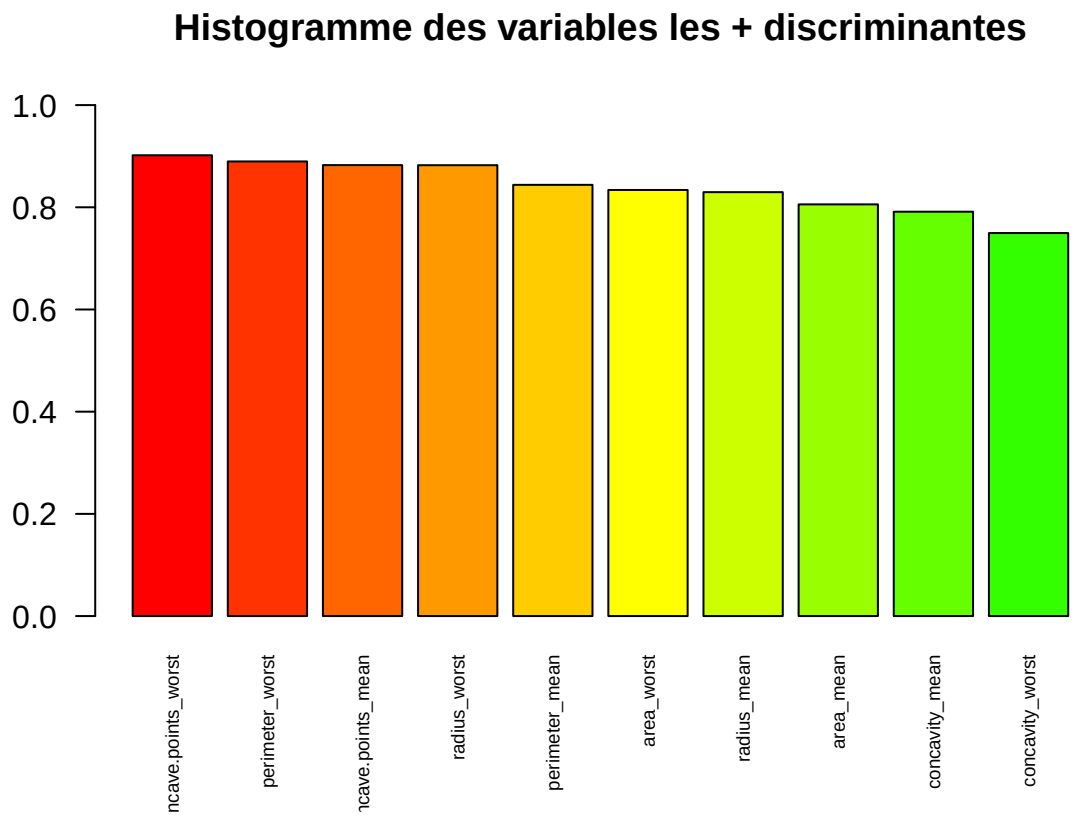
Pour tracer le cercle de corrélation, il nous faut les coordonnées des variables sur les axes discriminants. Or, comme il y a un seul axe, les coordonnées des variables sont les coefficients de l'axe discriminant.

```
F12 = predict(res.afd, prior=rep(1/2,2))$x
cercle_correlation=cor(data[,-1],F12)
cercle_correlation <- cbind(cercle_correlation, rep(0, nrow(cercle_correlation)))
a=seq(0,2* pi,length=100)
plot(cos(a), sin(a), type='l',lty=3,xlab='Dim 1', ylab='Dim 2',main="Cercle des corrélations AFD" )

arrows(0,0,cercle_correlation[,1],cercle_correlation[,2],col=2)
text(cercle_correlation,labels=colnames(data[,-1]), col = "red", cex = 0.7, pos = 3, offset = 0.5, srt =
```



On ne peut pas distinguer une variable en particulier qui influe sur la classification. En effet, le cercle manque un peu de visibilité. On peut faire un barplot des variables pour voir lesquelles sont les plus discriminantes.

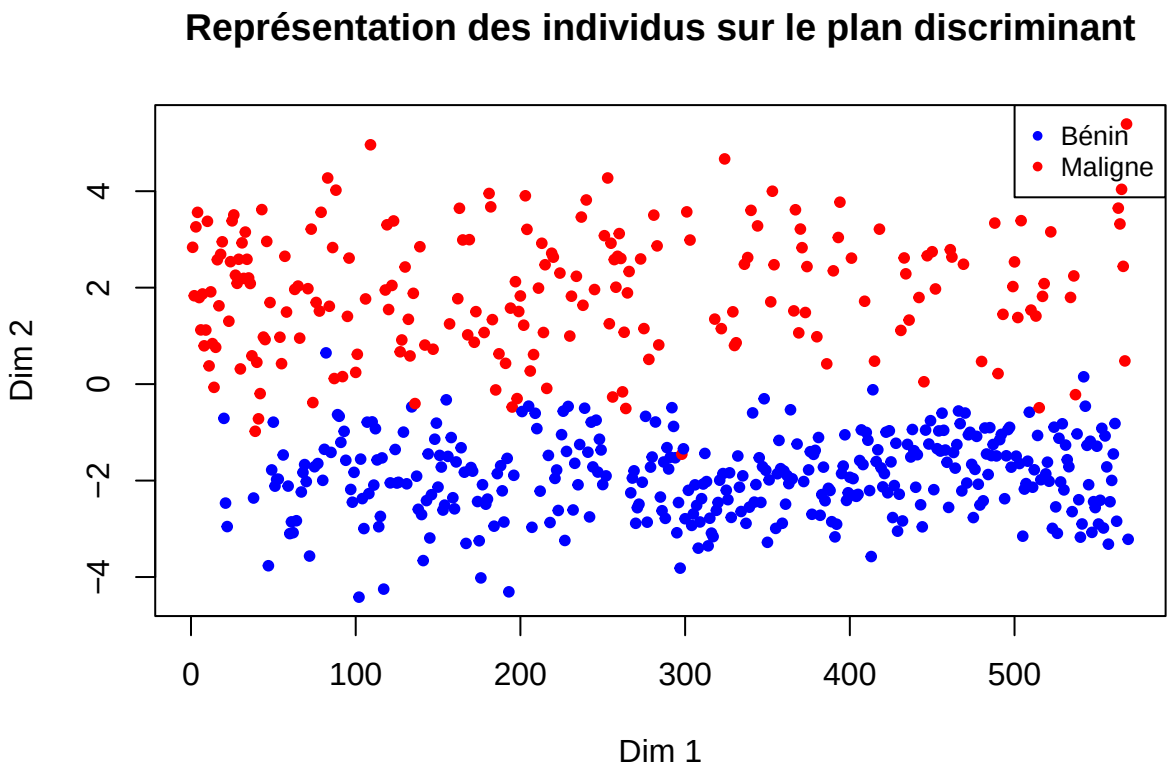
```
most_discriminant <- sort(abs(cercle_correlation[,1]), decreasing = TRUE)[1:10]
barplot(most_discriminant, names.arg=colnames(most_discriminant), las=2, col=rainbow(30), main="Histogra
```



Les variables les plus discriminantes sont les variables qui ont une influence sur l'axe discriminant. Parmi elles, on observe notamment les variables `concave.points_worst`, `perimeter.worst`, `concave.points_mean` et `radius_worst`.

Représentation des individus

```
plot(F12, col = c("blue", "red")[data$diagnosis], pch = 20, main = "Représentation des individus sur le
legend("topright", legend = c("Bénin", "Maligne"), col = c("blue", "red"), pch = 20, cex = 0.8)
```



On obtient une représentation des individus sur le plan discriminant. On observe que les individus sont bien séparés.

Conclusion

L'ACP sur les données a permis de mettre en évidence les variables qui contribuent le plus à la variance des données. L'AFD a permis de séparer les individus en fonction de leur diagnostic. Nous n'avons pas pu distinguer les variables les plus discriminantes sur le cercle de corrélation, mais nous avons pu les identifier grâce à un barplot. De plus amples recherches pourraient être faites sur les individus 462, 213 et 123 qui semblent être très influents sur les plans principaux de l'ACP.

9.5 Code pour la partie Clustering

Projet 5/5 Clustering

Alexandre CORRIOU - Nicolas SALVAN

2024-05-23

Ce document a pour objectif de réaliser une analyse de clustering, il contient toutes les sorties R et les commentaires associés.

Lecture des données nettoyées

Importation du dataset

```
data <- read.csv("data/data_cleaned.csv", header = TRUE, sep = ",")
data$diagnosis <- as.factor(data$diagnosis)
```

Aperçu rapide

```
head(data)
```

```
##   diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean
## 1         M       17.99        10.38         122.80    1001.0         0.11840
## 2         M       20.57        17.77         132.90    1326.0         0.08474
## 3         M       19.69        21.25         130.00    1203.0         0.10960
## 4         M       11.42        20.38          77.58     386.1         0.14250
## 5         M       20.29        14.34         135.10    1297.0         0.10030
## 6         M       12.45        15.70          82.57     477.1         0.12780
##   compactness_mean concavity_mean concave.points_mean symmetry_mean
## 1          0.27760         0.3001             0.14710        0.2419
## 2          0.07864         0.0869             0.07017        0.1812
## 3          0.15990         0.1974             0.12790        0.2069
## 4          0.28390         0.2414             0.10520        0.2597
## 5          0.13280         0.1980             0.10430        0.1809
## 6          0.17000         0.1578             0.08089        0.2087
##   fractal_dimension_mean radius_se texture_se perimeter_se area_se
## 1                0.07871    1.0950     0.9053        8.589  153.40
## 2                0.05667    0.5435     0.7339        3.398   74.08
## 3                0.05999    0.7456     0.7869        4.585   94.03
## 4                0.09744    0.4956     1.1560        3.445   27.23
## 5                0.05883    0.7572     0.7813        5.438   94.44
## 6                0.07613    0.3345     0.8902        2.217   27.19
##   smoothness_se compactness_se concavity_se concave.points_se symmetry_se
```

```
## 1      0.006399        0.04904      0.05373           0.01587     0.03003
## 2      0.005225        0.01308      0.01860           0.01340     0.01389
## 3      0.006150        0.04006      0.03832           0.02058     0.02250
## 4      0.009110        0.07458      0.05661           0.01867     0.05963
## 5      0.011490        0.02461      0.05688           0.01885     0.01756
## 6      0.007510        0.03345      0.03672           0.01137     0.02165
##   fractal_dimension_se radius_worst texture_worst perimeter_worst area_worst
## 1             0.006193        25.38         17.33          184.60     2019.0
## 2             0.003532        24.99         23.41          158.80     1956.0
## 3             0.004571        23.57         25.53          152.50     1709.0
## 4             0.009208        14.91         26.50           98.87      567.7
## 5             0.005115        22.54         16.67          152.20     1575.0
## 6             0.005082        15.47         23.75          103.40      741.6
##   smoothness_worst compactness_worst concavity_worst concave.points_worst
## 1           0.1622            0.6656          0.7119               0.2654
## 2           0.1238            0.1866          0.2416               0.1860
## 3           0.1444            0.4245          0.4504               0.2430
## 4           0.2098            0.8663          0.6869               0.2575
## 5           0.1374            0.2050          0.4000               0.1625
## 6           0.1791            0.5249          0.5355               0.1741
##   symmetry_worst fractal_dimension_worst
## 1         0.4601                 0.11890
## 2         0.2750                 0.08902
## 3         0.3613                 0.08758
## 4         0.6638                 0.17300
## 5         0.2364                 0.07678
## 6         0.3985                 0.12440
```

```r
# str(data)
# summary(data)
```

Séparation des données numériques et centrage-réduction

```r
data_num <- data[-1]
data.cent <- scale(data_num)
```

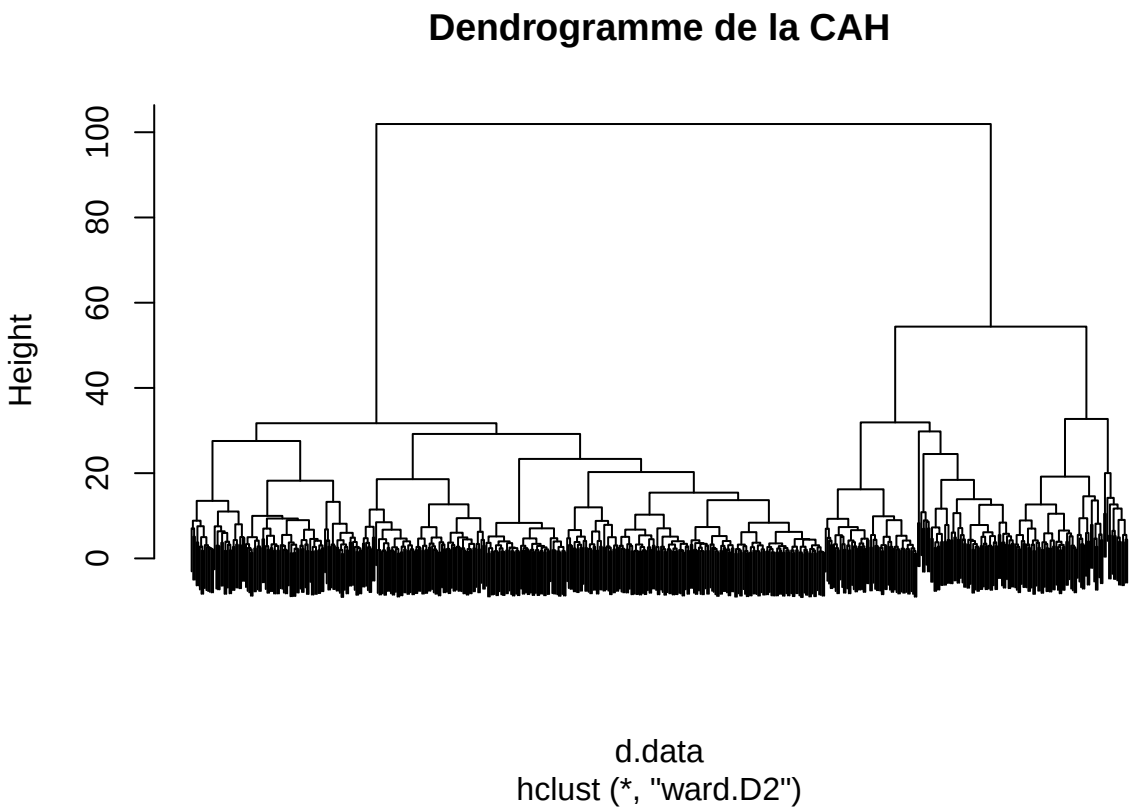
Modèles de clustering

Comme nous avons relativement peu d'individus, nous allons réaliser dans un premier temps une classification ascendante hiérarchique (CAH) pour avoir une idée du nombre de clusters à choisir, puis nous réaliserons un k-means pour obtenir les clusters.

CAH - Classification Ascendante Hiérarchique*

Dendrogramme

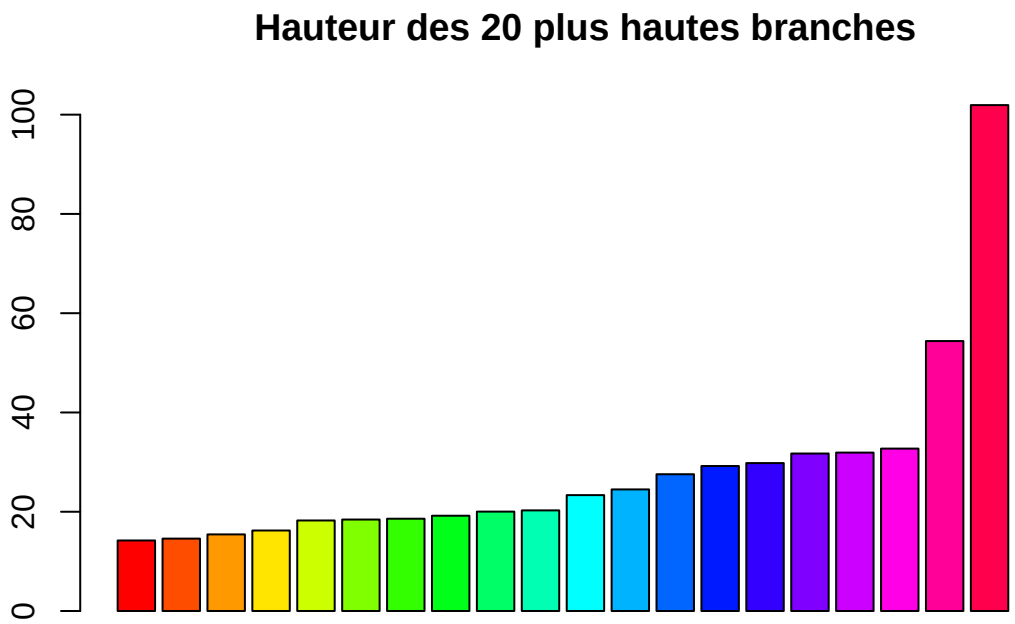
```r
d.data = dist(data.cent)
hc <- hclust(d.data, method = "ward.D2")
plot(hc, cex = 0.01, main = "Dendrogramme de la CAH" )
```



On peut voir qu'il y a 3 branches qui sont assez longues, on peut donc choisir 3 clusters. On peut le confirmer en affichant la hauteur des branches les plus hautes.

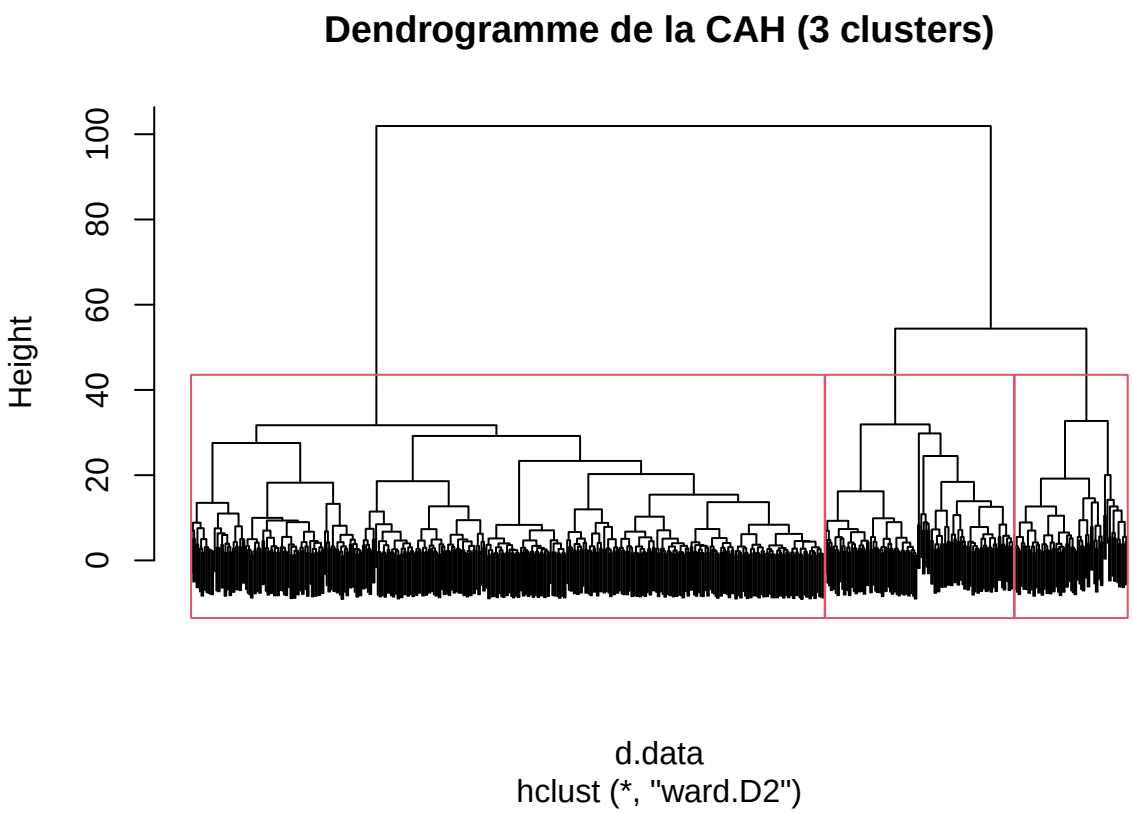
Hauteur des branches les plus hautes

```
N <- 20
barplot(hc$height[(dim(data)[1]-N):dim(data)[1]], col = rainbow(N), main = "Hauteur des 20 plus hautes
```



On peut bien déceler 3 branches qui sont assez longues. Nous choisirons donc 3 clusters pour la suite de l'analyse.

```
plot(hc, cex = 0.01, main = "Dendrogramme de la CAH (3 clusters)")
rect.hclust(hc, k = 3)
```



K-means

Clustering à 3 groupes

On calcule le clustering à 3 groupes en utilisant 1000 points de départ différents pour éviter les minima locaux.

```
kmeans.result=kmeans(data.cent, nstart = 1000, centers=3)
```

On constitue une base de données avec les données centrées-réduites et la classe de chaque individu obtenue avec le clustering.

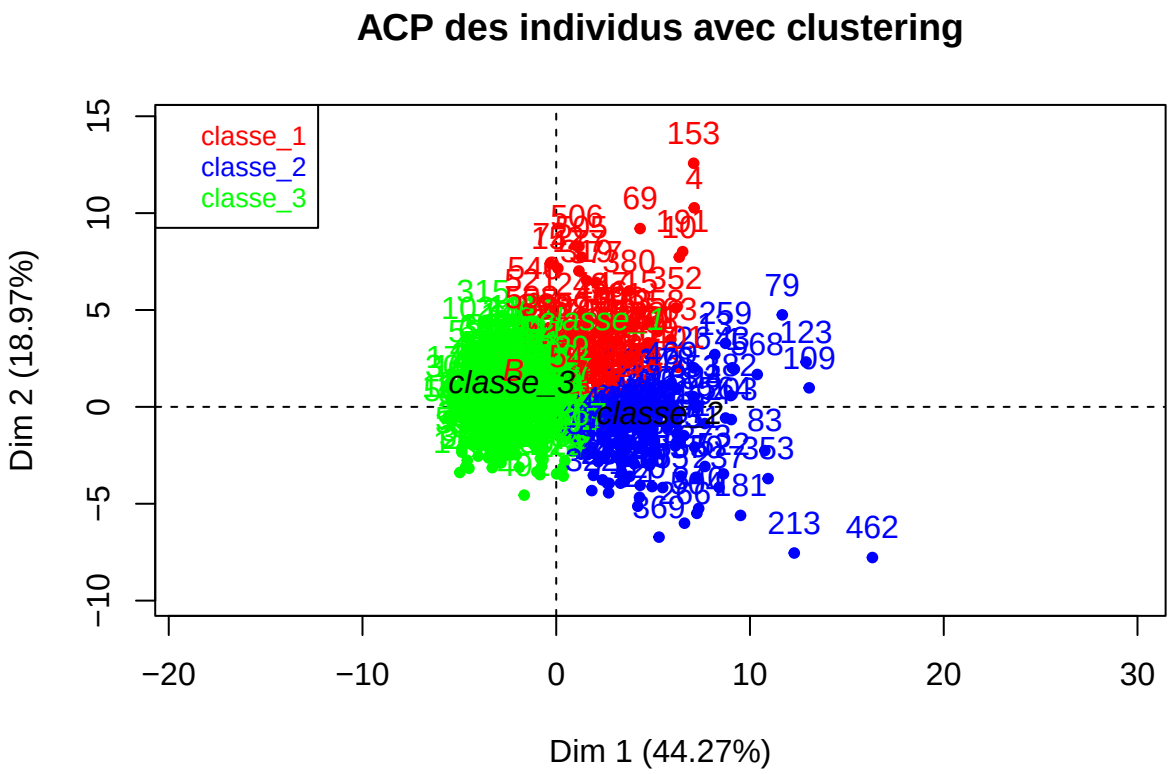
```
data_classe <- cbind.data.frame(data[1], data.cent, classe=factor(kmeans.result$cluster))
```

Visualisation des clusters avec une ACP

```
library(FactoMineR)
```

```
res.pca <- PCA(data_classe, graph = FALSE, quali.sup = c(1, 32))
```

```r
plot(res.pca, choix = "ind", graph.type = "classic", habillage = 32, col.hab = c("red", "blue", "green"))
```



On observe bien sur le plan principal de l'ACP que les clusters sont bien séparés.

Comparaison des clusters trouvés avec le diagnostic

On peut comparer les clusters trouvés avec le diagnostic initial pour voir si les clusters correspondent bien aux diagnostics.

```r
table(data_classe$diagnosis, data_classe$classe)
```

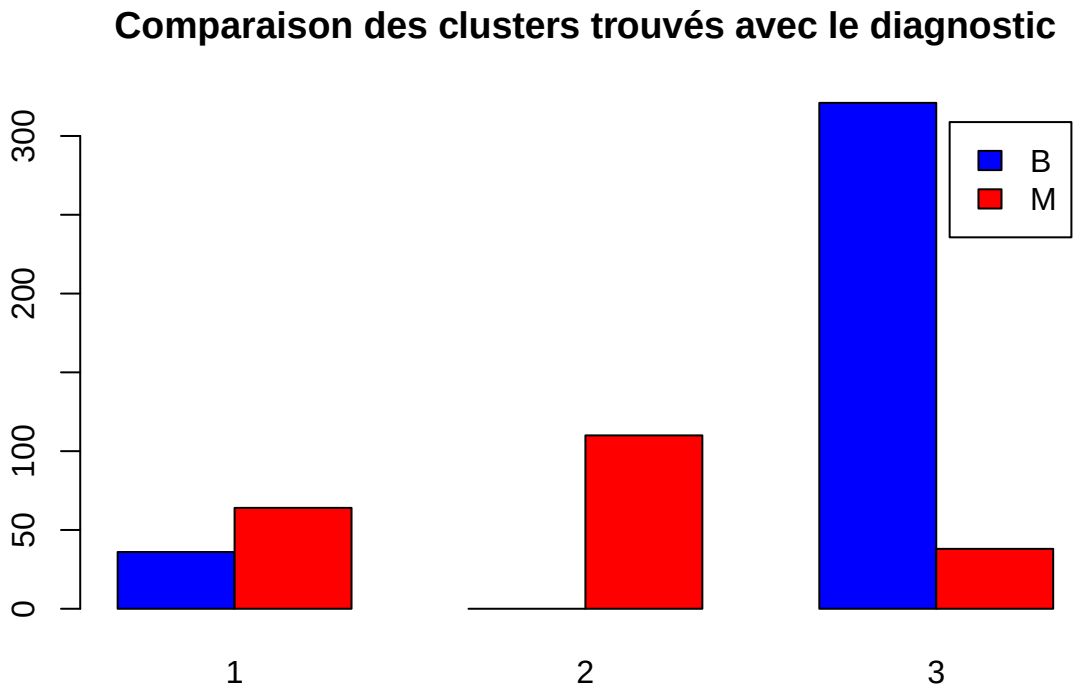
```
##
##       1   2   3
##   B  36   0 321
##   M  64 110  38
```

```r
table(data_classe$classe, data_classe$diagnosis)
```

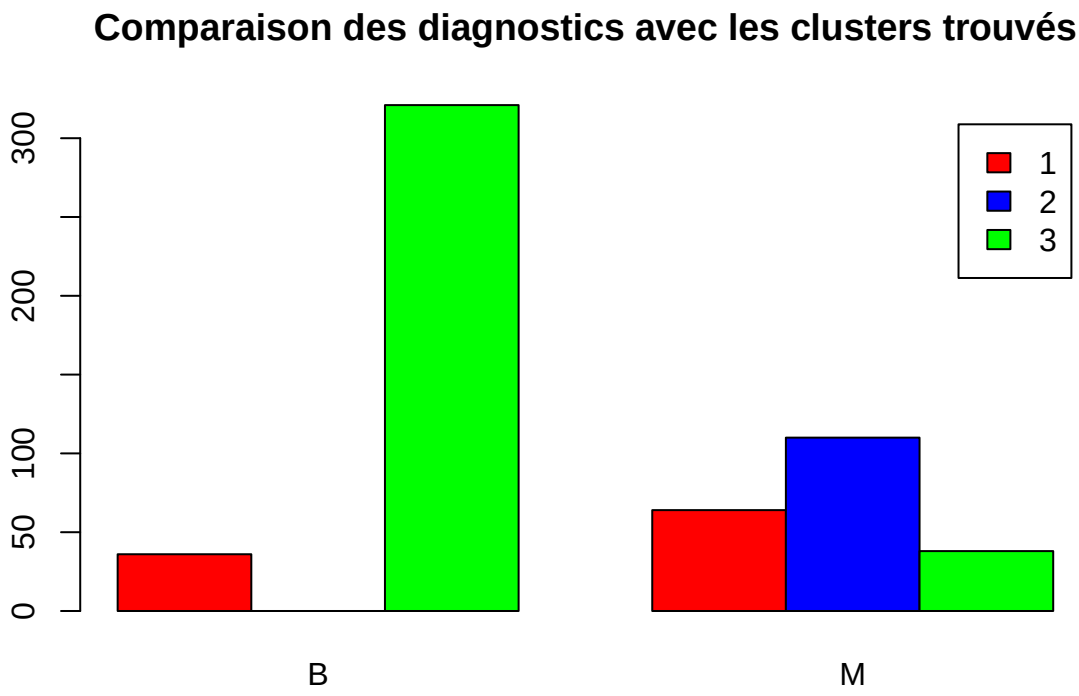
```
##
##       B   M
##   1  36  64
##   2   0 110
##   3 321  38
```

On observe que les clusters ne correspondent pas aux diagnostics. En effet, les clusters 1 et 2 correspondent à des diagnostics plutôt malins, et le cluster 3 correspond à des diagnostics plutôt bénins. Seul le cluster 2 correspond bien aux diagnostics malins.

```
barplot(table(data_classe$diagnosis, data_classe$classe), beside = TRUE, col = c("blue", "red"), legend
```




```
barplot(table(data_classe$classe, data_classe$diagnosis), beside = TRUE, col = c("red", "blue", "green"
```

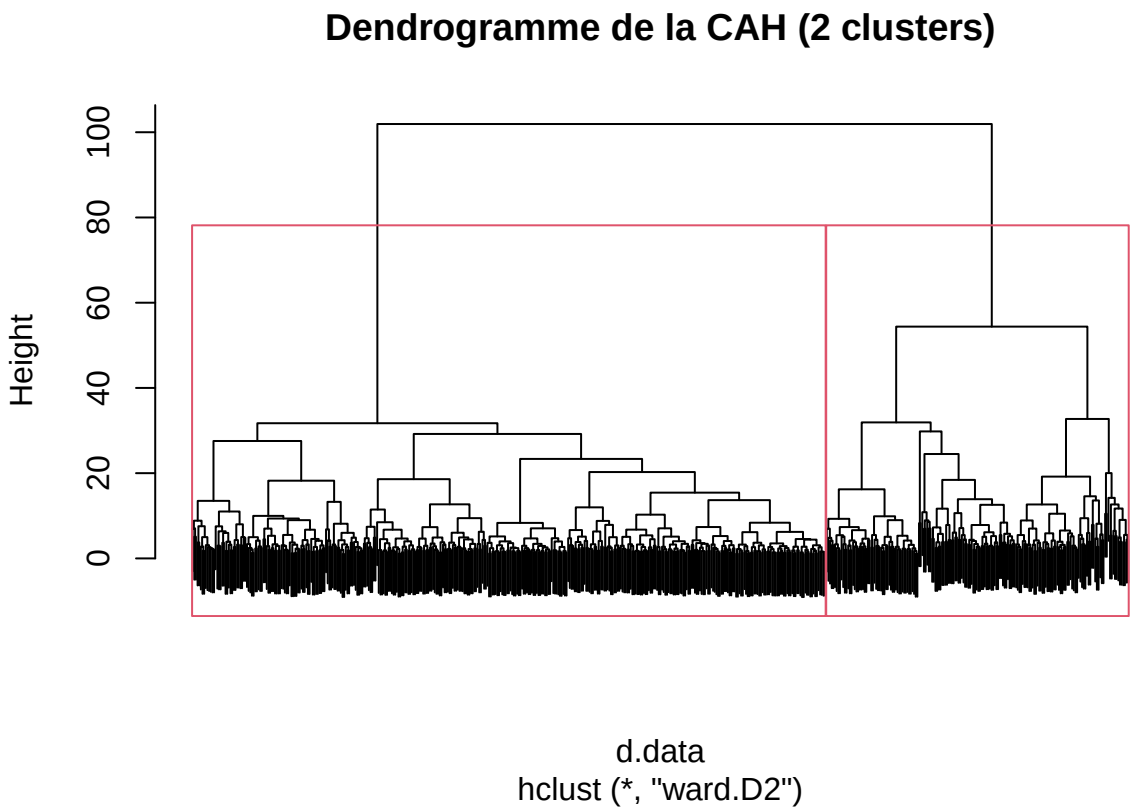


On peut visualiser ces résultats dans les histogrammes précédents.

Clustering à 2 groupes

On peut également réaliser un clustering à 2 groupes pour voir si les clusters correspondent mieux aux diagnostics.

```
plot(hc, cex = 0.01, main = "Dendrogramme de la CAH (2 clusters)")
rect.hclust(hc, k = 2)
```

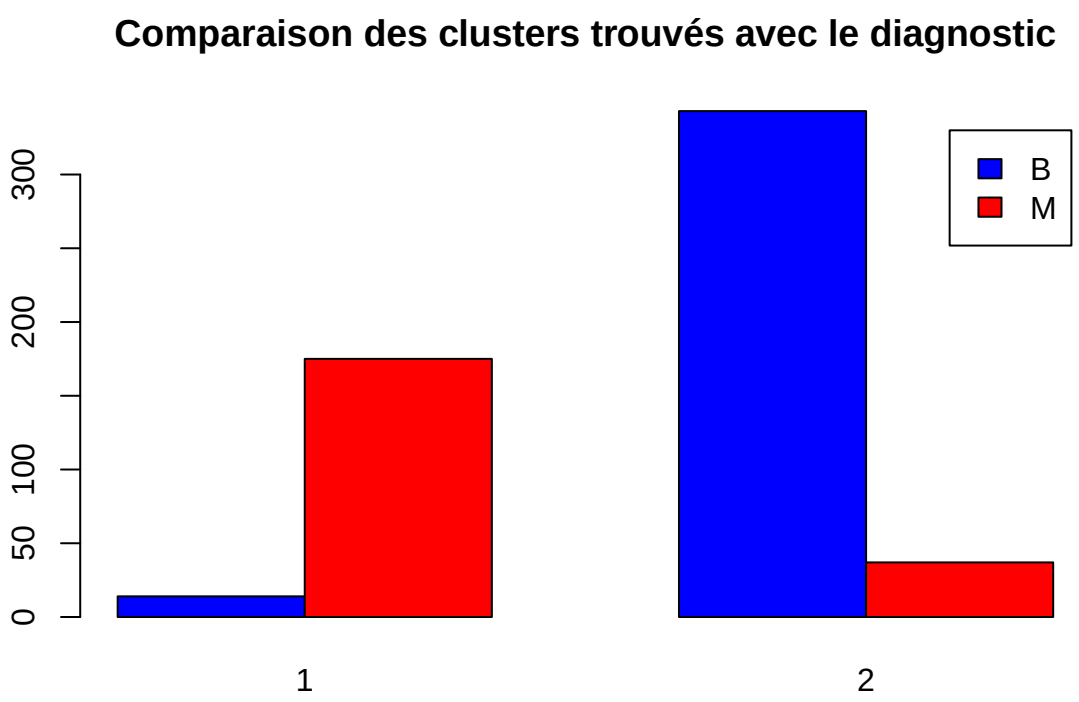



```
kmeans.result2=kmeans(data.cent, nstart = 1000, centers=2)
data_classe2 <- cbind.data.frame(data[1], data.cent, classe=factor(kmeans.result2$cluster))
table(data_classe2$diagnosis, data_classe2$classe)
```

```
##
##       1   2
##   B  14 343
##   M 175  37
```

On observe que les clusters correspondent mieux aux diagnostics. En effet, le cluster 1 correspond aux diagnostics bénins, et le cluster 2 correspond aux diagnostics malins.

```
barplot(table(data_classe2$diagnosis, data_classe2$classe), beside = TRUE, col = c("blue", "red"), lege
```



La classification à 2 groupes semble donc plus pertinente pour une analyse de clustering.

Conclusion

Nous avons réalisé une analyse de clustering sur les données nettoyées. Nous avons choisi de réaliser une CAH pour déterminer le nombre de clusters à choisir, puis un k-means pour obtenir les clusters. Nous avons choisi 3 clusters. Nous avons ensuite réalisé une ACP pour visualiser les clusters. Nous avons comparé les clusters trouvés avec les diagnostics initiaux. Nous avons observé que les clusters ne correspondaient pas forcément aux diagnostics. En effet, les clusters 1 et 2 correspondent à des diagnostics plutôt malins, et le cluster 3 correspond à des diagnostics plutôt bénins. Seul le cluster 2 correspond bien aux diagnostics malins.

Il est ainsi possible de déceler plusieurs catégories de tumeurs, mais il est plus difficile de savoir s'il s'agit de tumeurs bénignes ou malignes, d'où l'intérêt de réaliser une classification supervisée (ou pour les docteurs de faire d'autres tests).

9.6 Code pour la partie classification

Projet 3/5 Classification Supervisée

Nicolas SALVAN - Alexandre CORRIOU

2024-05-23

Ce document contient le code pour *modéliser les données*. Nous allons réaliser de l'apprentissage supervisé pour prédire le diagnostic des patientes.

Lecture des données nettoyées

Importation du dataset

```
data <- read.csv("data/data_cleaned.csv", header = TRUE, sep = ",")
data$diagnosis <- as.factor(data$diagnosis)

train_data <- read.csv("data/train_data.csv", header = TRUE, sep = ",")
train_data$diagnosis <- as.factor(train_data$diagnosis)

test_data <- read.csv("data/test_data.csv", header = TRUE, sep = ",")
test_data$diagnosis <- as.factor(test_data$diagnosis)
```

Aperçu rapide

```
head(data)
```

```
##   diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean
## 1         M       17.99        10.38         122.80    1001.0         0.11840
## 2         M       20.57        17.77         132.90    1326.0         0.08474
## 3         M       19.69        21.25         130.00    1203.0         0.10960
## 4         M       11.42        20.38          77.58     386.1         0.14250
## 5         M       20.29        14.34         135.10    1297.0         0.10030
## 6         M       12.45        15.70          82.57     477.1         0.12780
##   compactness_mean concavity_mean concave.points_mean symmetry_mean
## 1          0.27760         0.3001             0.14710        0.2419
## 2          0.07864         0.0869             0.07017        0.1812
## 3          0.15990         0.1974             0.12790        0.2069
## 4          0.28390         0.2414             0.10520        0.2597
## 5          0.13280         0.1980             0.10430        0.1809
## 6          0.17000         0.1578             0.08089        0.2087
##   fractal_dimension_mean radius_se texture_se perimeter_se area_se
## 1                0.07871    1.0950     0.9053        8.589  153.40
```

```
## 2               0.05667    0.5435     0.7339        3.398   74.08
## 3               0.05999    0.7456     0.7869        4.585   94.03
## 4               0.09744    0.4956     1.1560        3.445   27.23
## 5               0.05883    0.7572     0.7813        5.438   94.44
## 6               0.07613    0.3345     0.8902        2.217   27.19
##   smoothness_se compactness_se concavity_se concave.points_se symmetry_se
## 1      0.006399        0.04904      0.05373           0.01587     0.03003
## 2      0.005225        0.01308      0.01860           0.01340     0.01389
## 3      0.006150        0.04006      0.03832           0.02058     0.02250
## 4      0.009110        0.07458      0.05661           0.01867     0.05963
## 5      0.011490        0.02461      0.05688           0.01885     0.01756
## 6      0.007510        0.03345      0.03672           0.01137     0.02165
##   fractal_dimension_se radius_worst texture_worst perimeter_worst area_worst
## 1             0.006193        25.38         17.33          184.60     2019.0
## 2             0.003532        24.99         23.41          158.80     1956.0
## 3             0.004571        23.57         25.53          152.50     1709.0
## 4             0.009208        14.91         26.50           98.87      567.7
## 5             0.005115        22.54         16.67          152.20     1575.0
## 6             0.005082        15.47         23.75          103.40      741.6
##   smoothness_worst compactness_worst concavity_worst concave.points_worst
## 1           0.1622            0.6656          0.7119               0.2654
## 2           0.1238            0.1866          0.2416               0.1860
## 3           0.1444            0.4245          0.4504               0.2430
## 4           0.2098            0.8663          0.6869               0.2575
## 5           0.1374            0.2050          0.4000               0.1625
## 6           0.1791            0.5249          0.5355               0.1741
##   symmetry_worst fractal_dimension_worst
## 1         0.4601                 0.11890
## 2         0.2750                 0.08902
## 3         0.3613                 0.08758
## 4         0.6638                 0.17300
## 5         0.2364                 0.07678
## 6         0.3985                 0.12440
```

```
# dim(data)
# str(data)
```

Modélisation

AFD (Analyse Factorielle Discriminante)

Nous allons réaliser une AFD pour prédire le diagnostic des patientes.

```
# install.packages("MASS")
library(MASS)
library(pROC)
```

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

##
## Attachement du package : 'pROC'
```

```
## Les objets suivants sont masqués depuis 'package:stats':
##
##     cov, smooth, var
```

Lancement du modèle

```
afd_lda_model <- lda(diagnosis ~ ., data = train_data)
```

Prédiction

On prédit les données avec le modèle, en précisant les probabilités a priori. On obtient alors la table de confusion suivante.

```
pred_afd <- predict(afd_lda_model, test_data, prior=c(0.5, 0.5))
table(pred_afd$class, test_data$diagnosis)
```

```
##
##      B  M
##   B 81  9
##   M  0 24
```

On observe que le modèle a prédit 0 faux négatifs et 9 faux positifs. Peut-être que les données d'entrainement ne sont pas assez représentatives, et que l'on a des tailles de classes différentes.

```
table(train_data$diagnosis)
```

```
##
##   B   M
## 276 179
```

```
table(test_data$diagnosis)
```

```
##
##  B  M
## 81 33
```

On est en effet sur du 2/3 vs 1/3. Il faudrait réequilibrer les données pour obtenir des résultats plus fiables.

Performance du modèle

```
afd_accuracy <- mean(pred_afd$class == test_data$diagnosis) # accuracy
afd_accuracy
```

```
## [1] 0.9210526
```

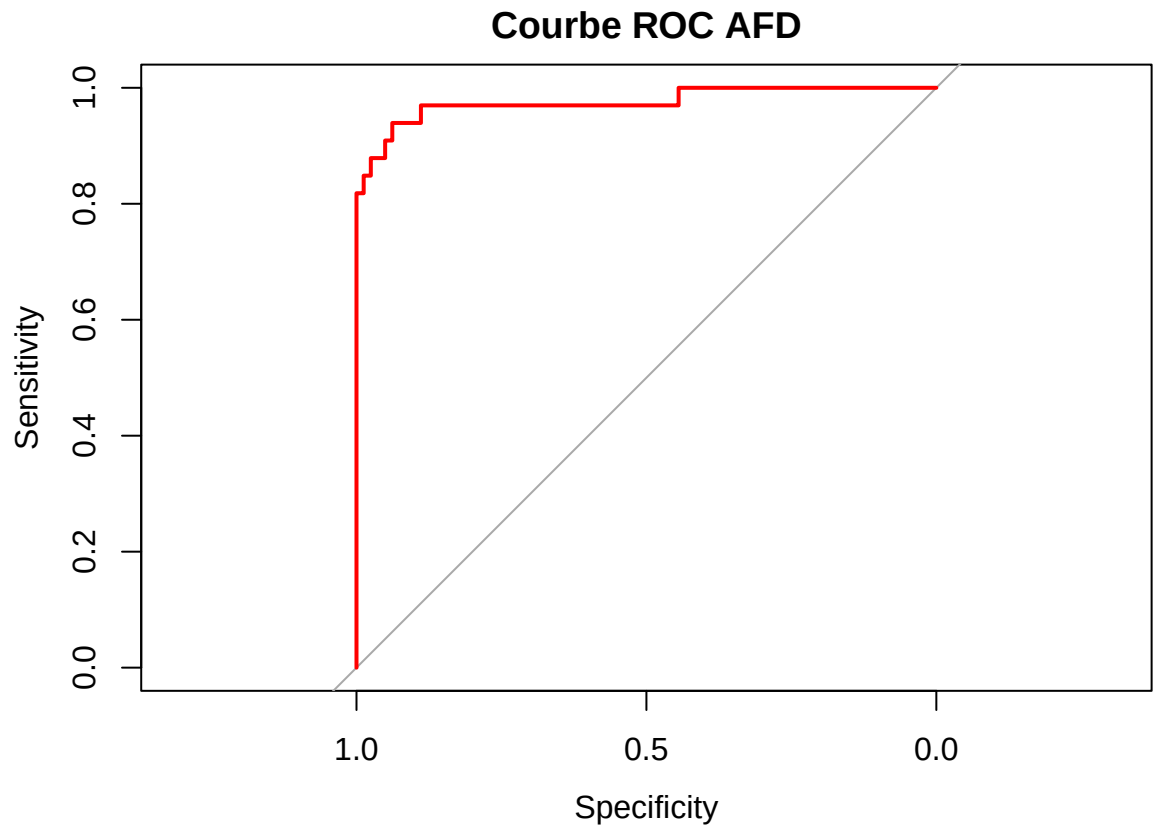
On obtient une accuracy de 0.92, ce qui est plutôt bon.

```r
roc_afd <- roc(test_data$diagnosis, pred_afd$posterior[,2])
```

```
## Setting levels: control = B, case = M
```

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

```r
plot(roc_afd, col = "red", AUC = TRUE, main = "Courbe ROC AFD")
```




```r
auc_afd <- auc(roc_afd)
auc_afd
```

```
## Area under the curve: 0.9753
```

LDA (Analyse Discriminante Linéaire)

Lancement du modèle

```r
# Il a déjà été lancé dans la partie AFD
# afd_lda_model <- lda(diagnosis ~ ., data = train_data)
```

Prédiction

```
pred_lda <- predict(afd_lda_model, test_data)
table(pred_lda$class, test_data$diagnosis)
```

```
##
##      B  M
##   B 81 11
##   M  0 22
```

Performance du modèle

```
lda_accuracy <- mean(pred_lda$class == test_data$diagnosis) # accuracy
lda_accuracy
```

```
## [1] 0.9035088
```

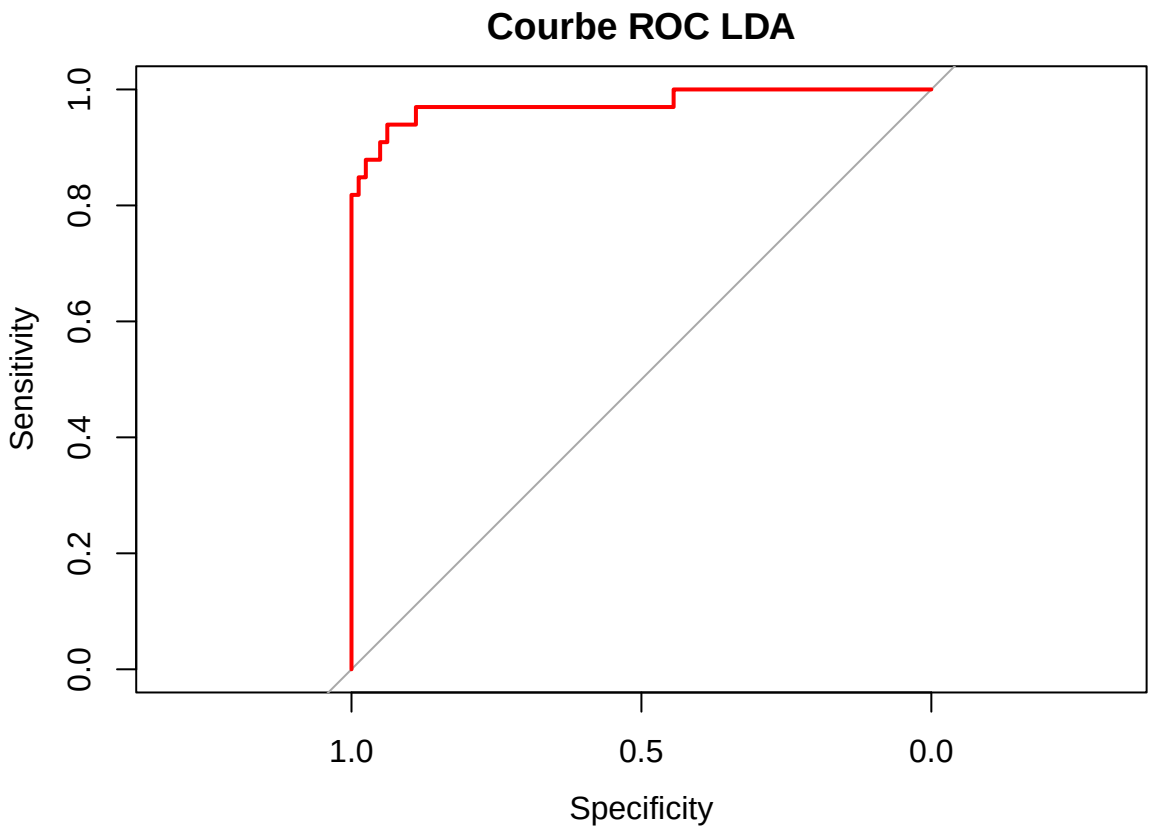
On obtient une accuracy un peu plus faible que l'AFD. Cela s'explique par le fait que l'AFD est plus adaptée aux données.

```
roc_lda <- roc(test_data$diagnosis, pred_lda$posterior[,2])
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_lda, col = "red", AUC = TRUE, main = "Courbe ROC LDA")
```




```
auc_lda <- auc(roc_lda)
auc_lda
```

```
## Area under the curve: 0.9753
```

Simplication du modèle

On peut simplifier le modèle en ne prenant que les variables les plus importantes.

```
library(klaR)
```

```
stepwise_lda <- stepclass(diagnosis~., data=train_data, method="lda", direction="backward", output = FA
```

```
summary(stepwise_lda$model$name)
```

```
##    Length     Class      Mode
##        28      AsIs character
```

On a pu supprimer cinq variables inutiles. On relance une lda sur le modèle simplifié.

```
lda_simple_model <- lda(stepwise_lda$formula, data=train_data)
```

```r
pred_lda_simple <- predict(lda_simple_model, test_data)
table(pred_lda_simple$class, test_data$diagnosis)
```

```
##
##      B  M
##   B 81 10
##   M  0 23
```

```r
lda_simple_accuracy <- mean(pred_lda_simple$class == test_data$diagnosis) # accuracy
lda_simple_accuracy
```

```
## [1] 0.9122807
```

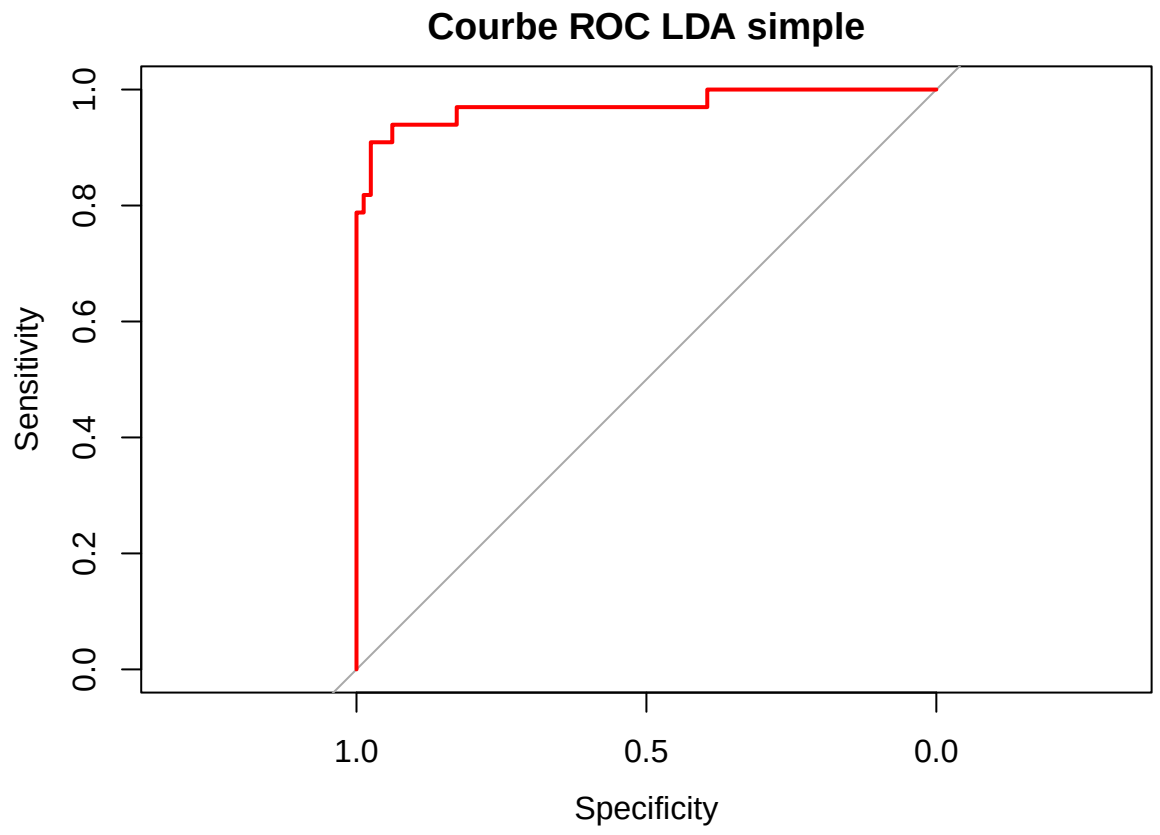
On obtient un score meilleur avec le modèle plus léger. Nous avons peut être fait du sur-apprentissage.

```r
roc_lda_simple <- roc(test_data$diagnosis, pred_lda_simple$posterior[,2])
```

```
## Setting levels: control = B, case = M
```

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

```r
plot(roc_lda_simple, col = "red", AUC = TRUE, main = "Courbe ROC LDA simple")
```



```r
auc_lda_simple <- auc(roc_lda_simple)
auc_lda_simple
```

```
## Area under the curve: 0.9719
```

QDA (Analyse Discriminante Quadratique)

Lancement du modèle

```r
afd_qda_model <- qda(diagnosis ~ ., data = train_data)
```

Prédiction

```r
pred_qda <- predict(afd_qda_model, test_data)
table(pred_qda$class, test_data$diagnosis)
```

```
##
##      B  M
##   B 79  4
##   M  2 29
```

Performance du modèle

```r
qda_accuracy <- mean(pred_qda$class == test_data$diagnosis) # accuracy
qda_accuracy
```

```
## [1] 0.9473684
```

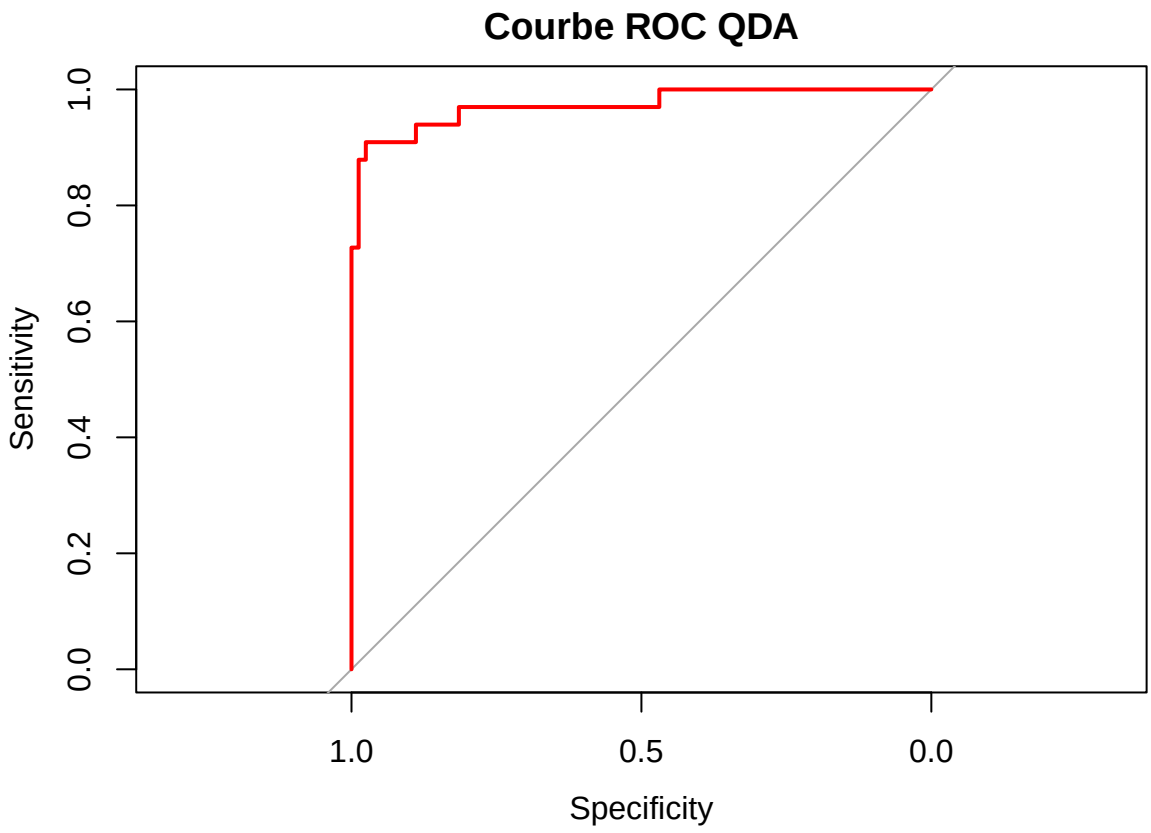
On obtient une accuracy de 0.947, ce qui est plutôt bon.

```r
roc_qda <- roc(test_data$diagnosis, pred_qda$posterior[,2])
```

```
## Setting levels: control = B, case = M
```

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

```r
plot(roc_qda, col = "red", AUC = TRUE, main = "Courbe ROC QDA")
```




```
auc_qda <- auc(roc_qda)
auc_qda
```

```
## Area under the curve: 0.9723
```

Simplification du modèle

On peut simplifier le modèle en ne prenant que les variables les plus importantes.

```
stepwise_qda <- stepclass(diagnosis~., data=train_data, method="qda", direction="backward", output = FA
```

```
qda_simple_model <- qda(stepwise_qda$formula, data=train_data)
```

```
summary(stepwise_qda$model$name)
```

```
##    Length     Class      Mode
##        27      AsIs character
```

On a pu supprimer deux variables inutiles. On relance une qda sur le modèle simplifié.

```
pred_qda_simple <- predict(qda_simple_model, test_data)
table(pred_qda_simple$class, test_data$diagnosis)
```

```
##
##      B  M
##   B 79  3
##   M  2 30
```

```
qda_simple_accuracy <- mean(pred_qda_simple$class == test_data$diagnosis) # accuracy
qda_simple_accuracy
```

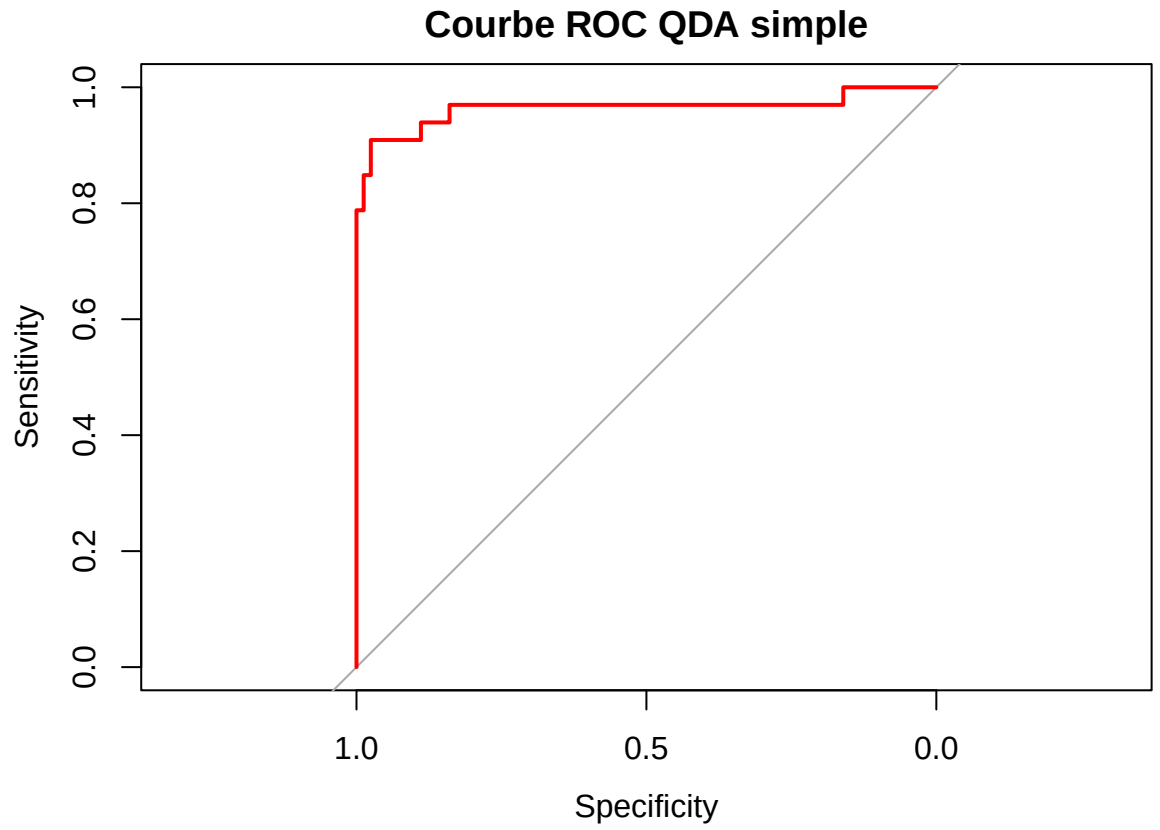
```
## [1] 0.9561404
```

```
roc_qda_simple <- roc(test_data$diagnosis, pred_qda_simple$posterior[,2])
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_qda_simple, col = "red", AUC = TRUE, main = "Courbe ROC QDA simple")
```




```
auc_qda_simple <- auc(roc_qda_simple)
auc_qda_simple
```

```
## Area under the curve: 0.9641
```

CART

```
library(rpart.plot)
```

```
## Le chargement a nécessité le package : rpart
```

```
library(rpart)
```

Lancement du modèle

Nous allons construire un arbre naïf, avec les paramètres par défaut.

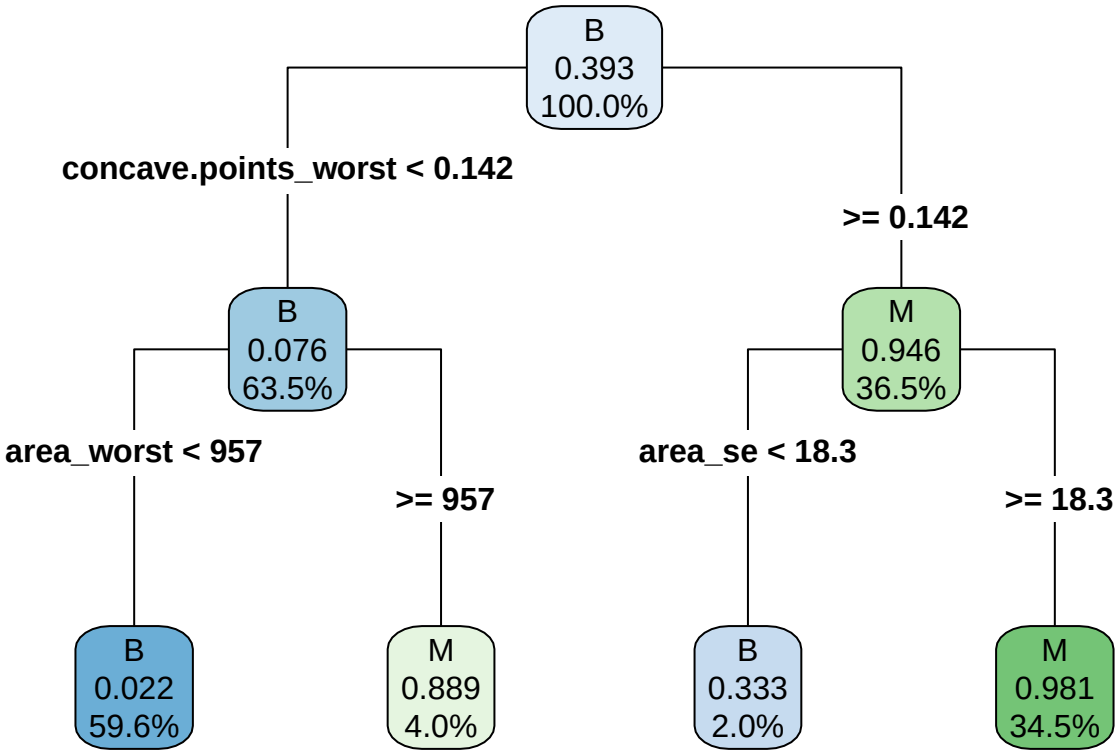
```
tree <- rpart(diagnosis~., train_data)
```

```
print(tree)
```

```
## n= 455
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 455 179 B (0.60659341 0.39340659)
##   2) concave.points_worst< 0.14235 289  22 B (0.92387543 0.07612457)
##     4) area_worst< 957.45 271   6 B (0.97785978 0.02214022) *
##     5) area_worst>=957.45 18   2 M (0.11111111 0.88888889) *
##   3) concave.points_worst>=0.14235 166   9 M (0.05421687 0.94578313)
##     6) area_se< 18.335 9   3 B (0.66666667 0.33333333) *
##     7) area_se>=18.335 157   3 M (0.01910828 0.98089172) *
```

Visualisation de l'arbre

```
rpart.plot(tree, type=4, digits=3,roundint=FALSE)
```



Prédiction et performance du modèle naïf

```
pred_cart_naive <- predict(tree, newdata=test_data, type="prob")
pred_cart_naive_qual <- ifelse(pred_cart_naive[,2] > 0.5, "M", "B")
table(pred_cart_naive_qual, test_data$diagnosis)
```

```
##
## pred_cart_naive_qual  B  M
##                    B 78  7
##                    M  3 26
```

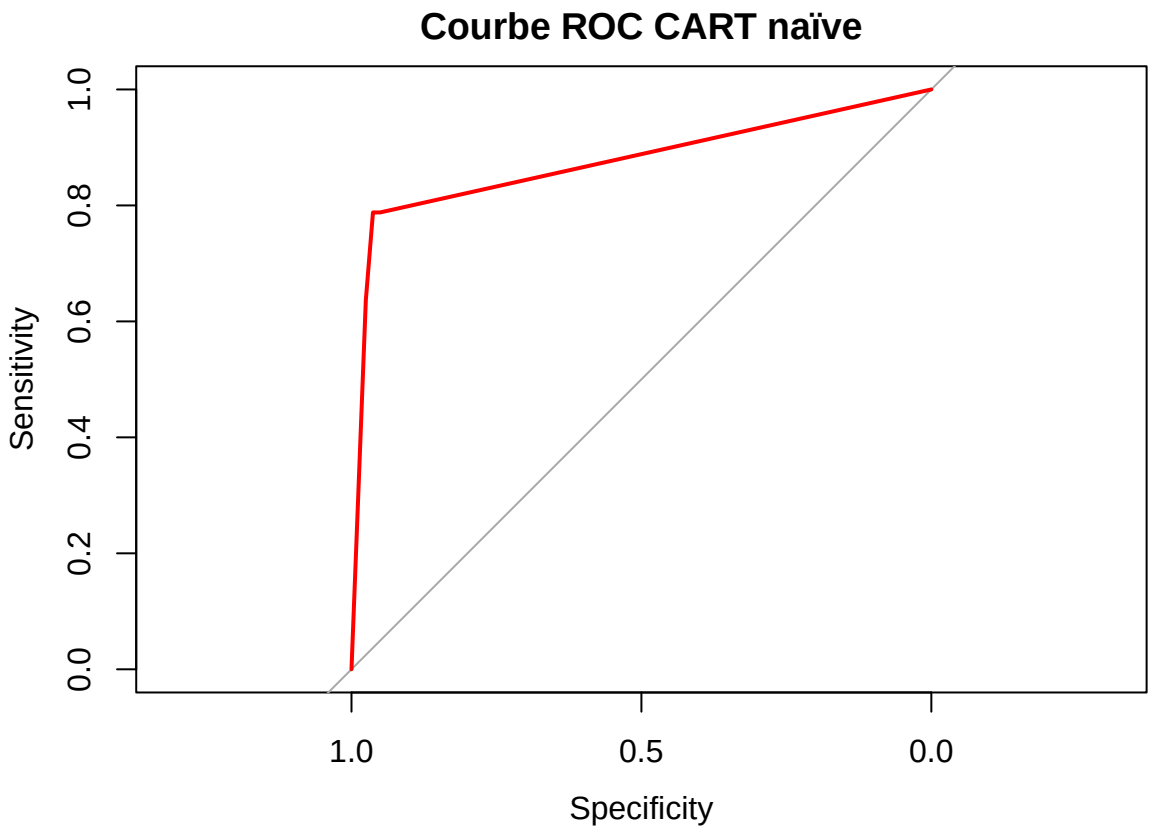
```
cart_naive_accuracy <- mean(pred_cart_naive_qual == test_data$diagnosis) # accuracy
cart_naive_accuracy
```

```
## [1] 0.9122807
```

```
roc_cart_naive <- roc(test_data$diagnosis, pred_cart_naive[,2], plot=TRUE, col = "red", main = "Courbe
```

```
## Setting levels: control = B, case = M
```

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




```
auc_cart_naive <- auc(roc_cart_naive)
auc_cart_naive
```

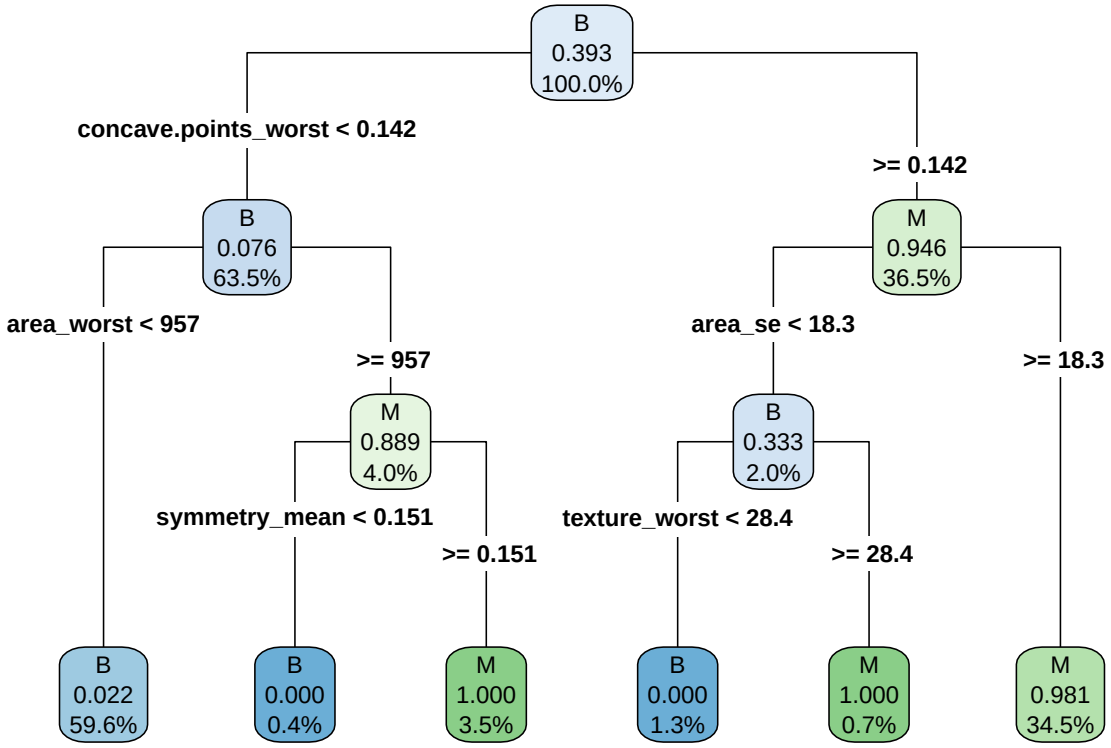
```
## Area under the curve: 0.8762
```

On n'obtient pas de très bonnes performances avec le modèle naïf. Nous allons essayer de l'améliorer.

Arbre simplifié

Nous allons simplifier l'arbre pour éviter le sur-apprentissage.

```
tree2 <- rpart(diagnosis~.,train_data,control=rpart.control(minsplit=5))
rpart.plot(tree2, type=4, digits=3)
```



Prédiction et performance du modèle simplifié

```
pred_cart <- predict(tree2, newdata=test_data, type="prob")
pred_cart_qual <- ifelse(pred_cart[,2] > 0.5, "M", "B")
table(pred_cart_qual, test_data$diagnosis)
```

```
##
## pred_cart_qual  B  M
##              B 78  8
##              M  3 25
```

```
cart_accuracy <- mean(pred_cart_qual == test_data$diagnosis) # accuracy
cart_accuracy
```

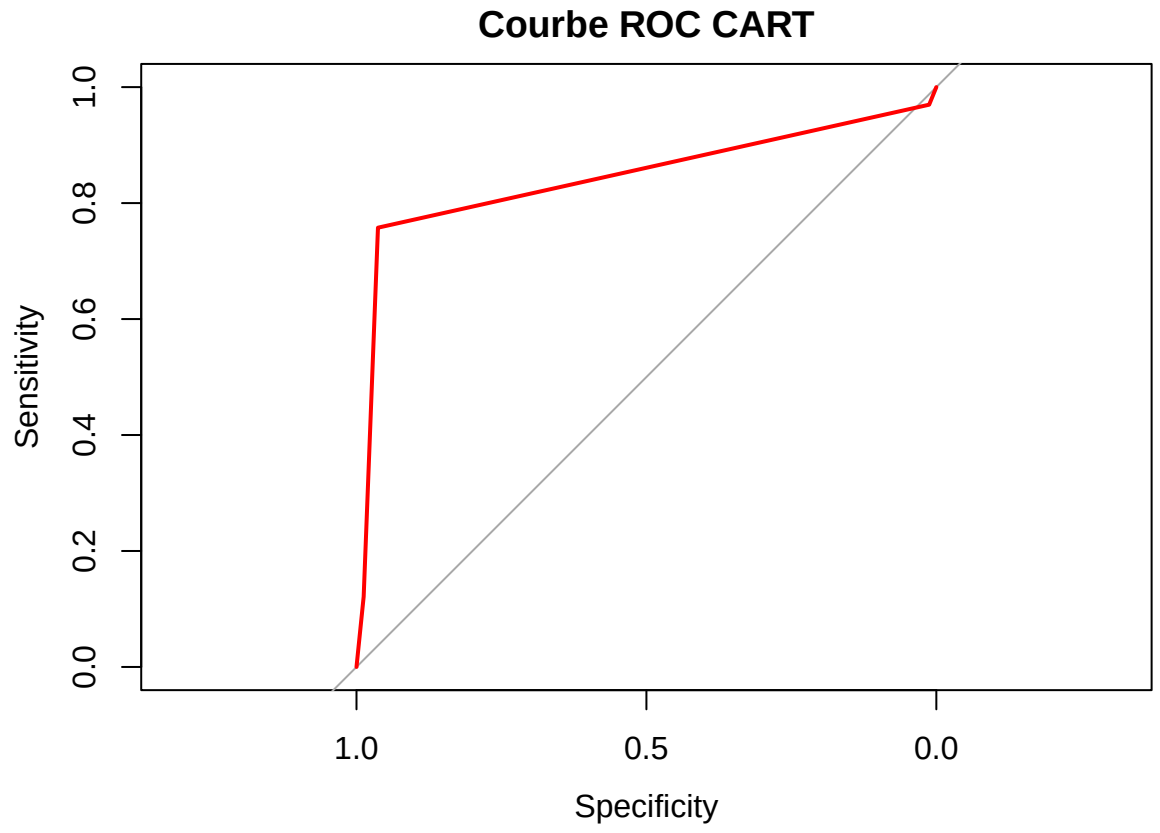
```
## [1] 0.9035088
```

```
roc_cart <- roc(test_data$diagnosis, pred_cart[,2])
```

```
## Setting levels: control = B, case = M
```

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

```r
plot(roc_cart, col = "red", AUC = TRUE, main = "Courbe ROC CART")
```




```r
auc_cart <- auc(roc_cart)
auc_cart
```

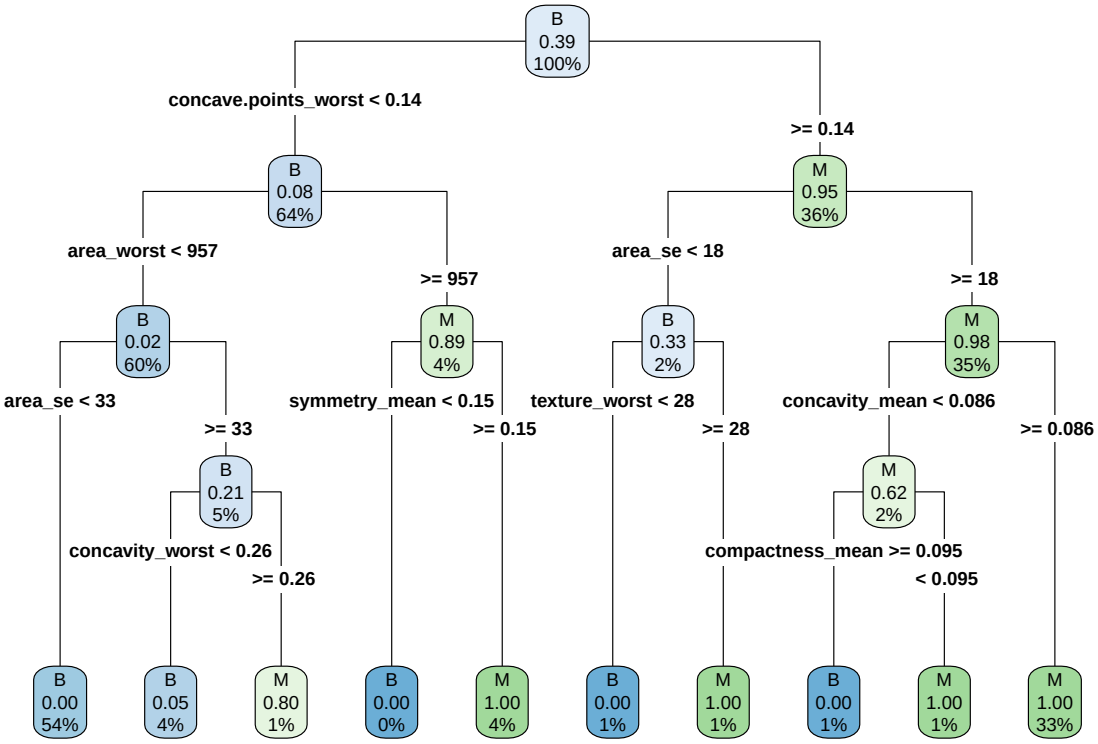
```
## Area under the curve: 0.8447
```

Cela n'a pas amélioré le modèle. Nous allons essayer de faire de l'élagage.

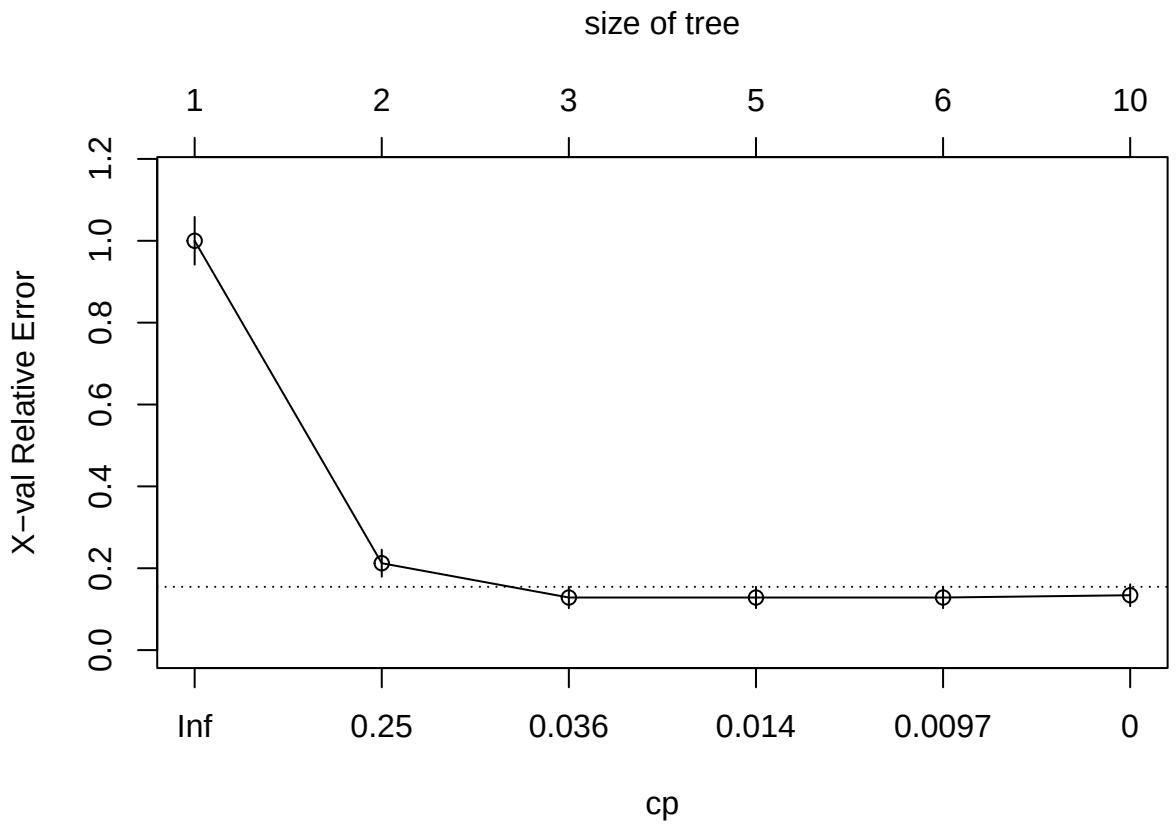
Elagage des arbres

Nous allons élaguer l'arbre pour éviter le sur-apprentissage.

```r
tree_elag <- rpart(diagnosis~.,train_data,control=rpart.control(minsplit=5,cp=0))
rpart.plot(tree_elag, type=4)
```

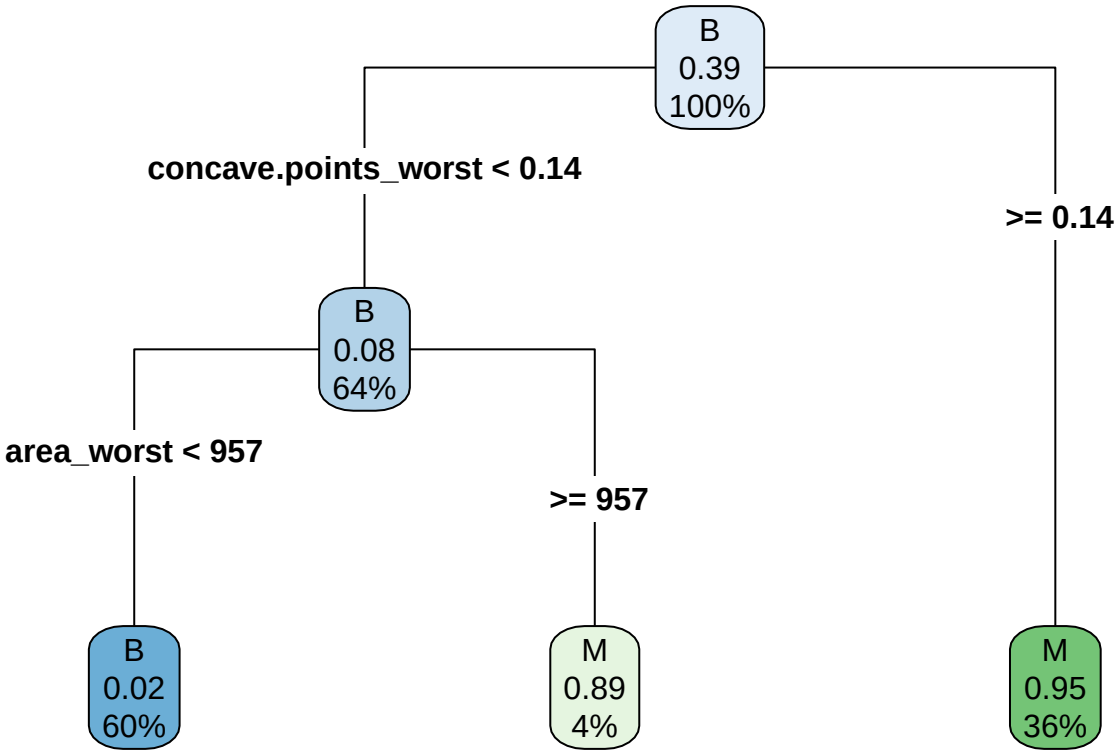



```
plotcp(tree_elag)
```



On observe une erreur minimale pour arbre de taille entre 4 et 7.

```
cp.opt <- tree_elag$cptable[which.min(tree_elag$cptable[, "xerror"]), "CP"]
tree.opt <- prune(tree_elag,cp=cp.opt)
rpart.plot(tree.opt, type=4)
```



Prediction

```
pred_cart_opti <- predict(tree.opt, newdata=test_data, type="prob")
pred_cart_opti_qual <- ifelse(pred_cart_opti[,2] > 0.5, "M", "B")
table(pred_cart_opti_qual, test_data$diagnosis)
```

```
##
## pred_cart_opti_qual  B  M
##                   B 77  7
##                   M  4 26
```

Performance du modèle

```
cart_opti_accuracy <- mean(pred_cart_opti_qual == test_data$diagnosis) # accuracy
cart_opti_accuracy
```

```
## [1] 0.9035088
```

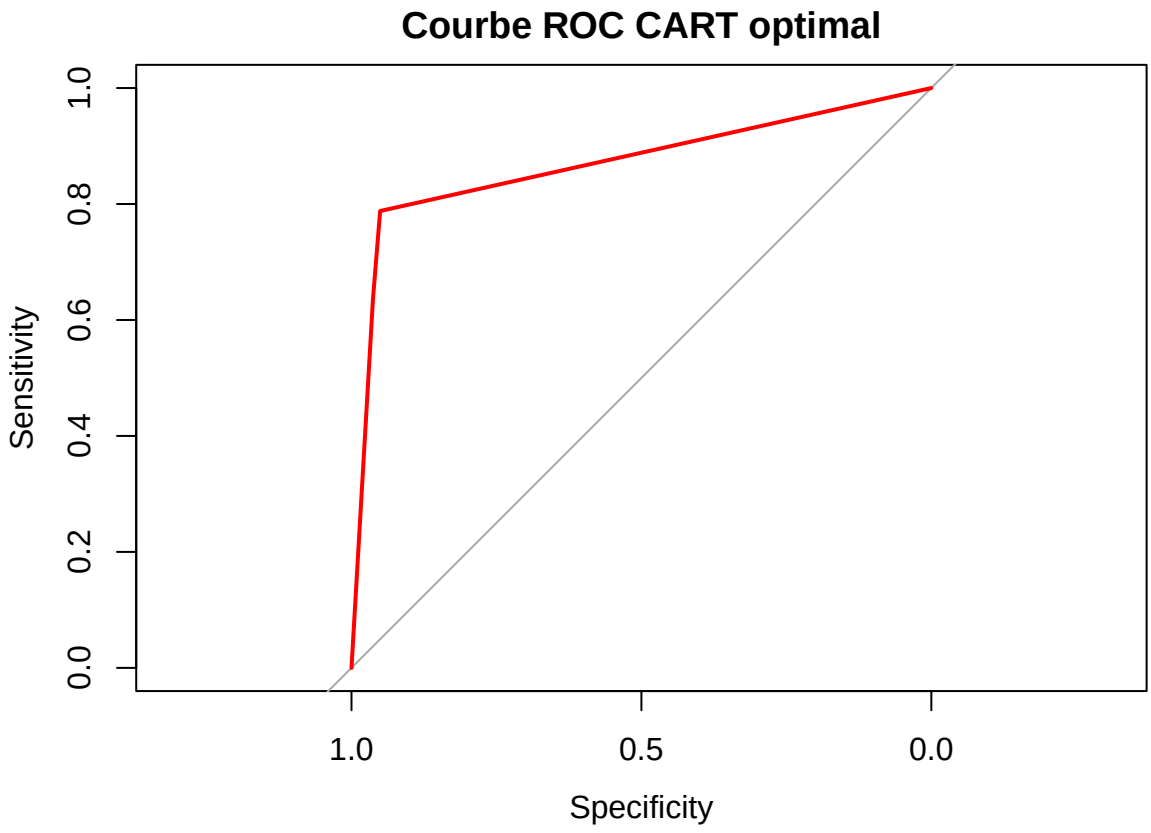
Pour le modèle CART optimal, on obtient une accuracy de 0.91, ce qui est meilleur que les version précédentes de CART.

```
roc_cart_opti <- roc(test_data$diagnosis, pred_cart_opti[,2])
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_cart_opti, col = "red", AUC = TRUE, main = "Courbe ROC CART optimal")
```




```
auc_cart_opti <- auc(roc_cart_opti)
auc_cart_opti
```

```
## Area under the curve: 0.8704
```

On obtient une AUC de 0.8788, ce qui est moins bon que les modèles précédents.

Random Forest

Lancement du modèle

```
# install.packages("randomForest")
library(randomForest)
```

```
## randomForest 4.7-1.1

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

```
rf_model <- randomForest(train_data$diagnosis ~ ., data = train_data, ntree = 100)
```

Prédiction

```
pred_rf <- predict(rf_model, test_data, type="prob")
pred_rf_fact <- ifelse(pred_rf[,2] > 0.5, "M", "B")
prob_rf <- pred_rf[, "B"]
table(pred_rf_fact, test_data$diagnosis)
```

```
##
## pred_rf_fact  B  M
##            B 79  6
##            M  2 27
```

Performance du modèle

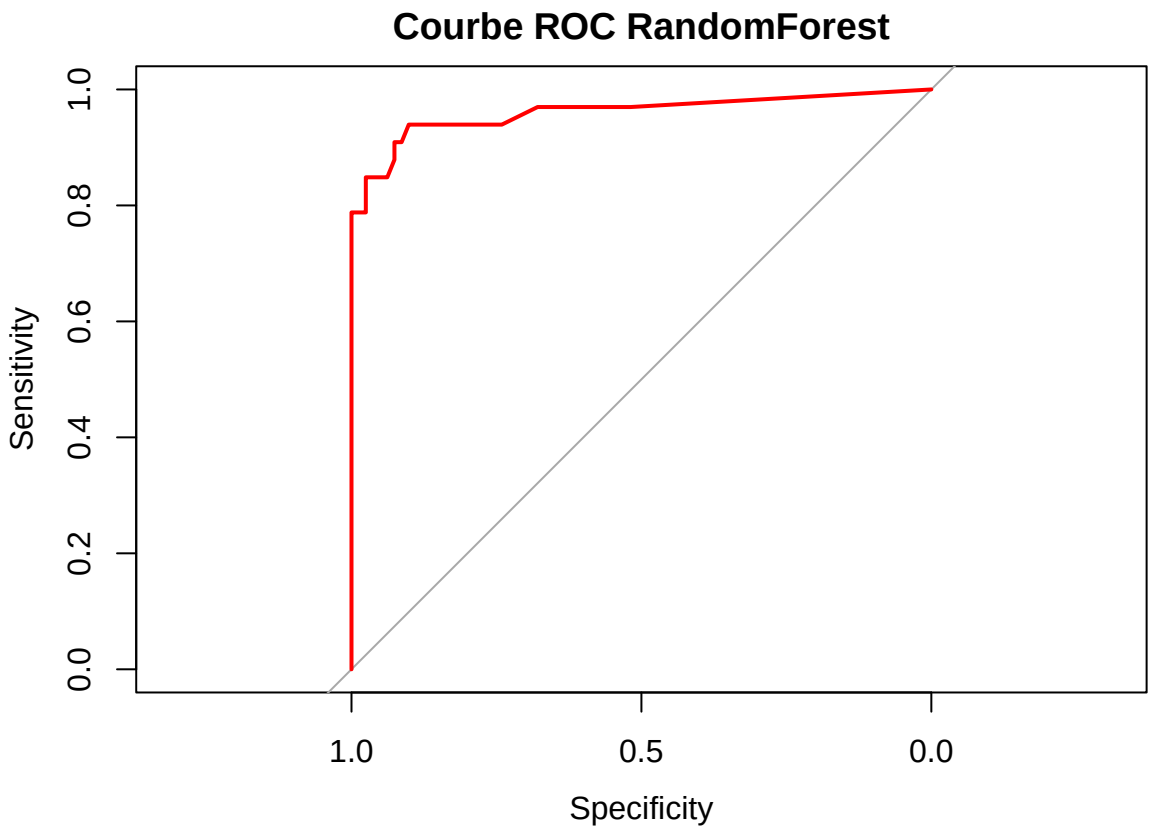
```
rf_accuracy <- mean(pred_rf_fact == test_data$diagnosis) # accuracy
rf_accuracy
```

```
## [1] 0.9298246
```

```
roc_rf <- roc(test_data$diagnosis, prob_rf)
```

```
## Setting levels: control = B, case = M

## Setting direction: controls > cases
```

```
plot(roc_rf, col = "red", AUC = TRUE, main = "Courbe ROC RandomForest")
```

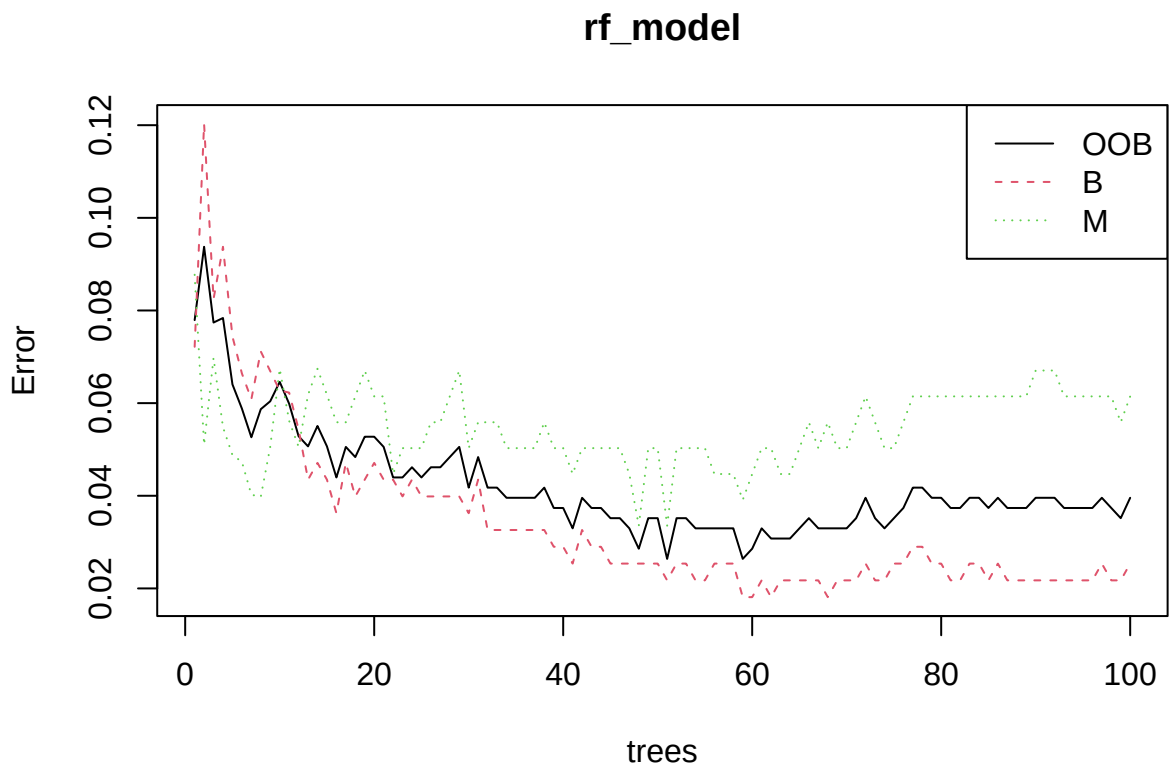



```
auc_rf <- auc(roc_rf)
auc_rf
```

```
## Area under the curve: 0.9602
```

Erreur OOB (Out-of-Bag)

```
plot(rf_model)
legend("topright", colnames(rf_model$err.rate), col=1:3, lty=1:3)
```



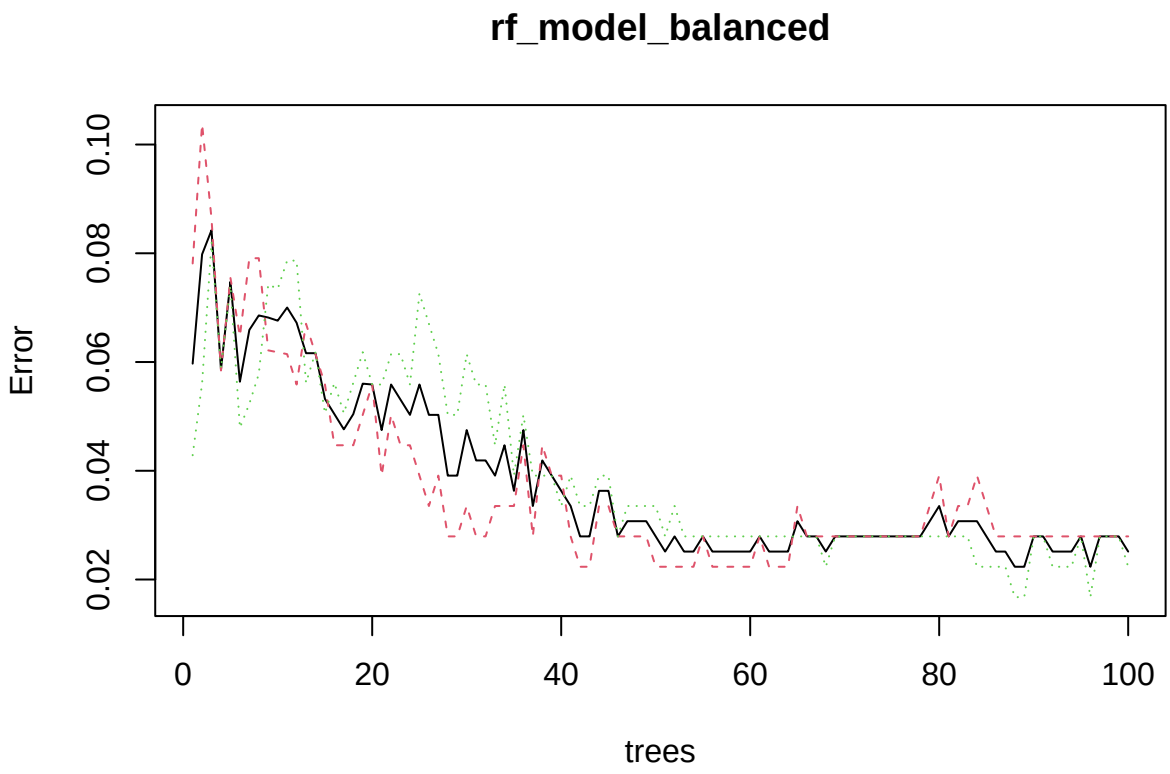
Le taux d'erreur semble s'être stabilisé. On observe que l'erreur OOB pour la classification "M" semble plus élevée que pour la classification "B". Il y a donc un déséquilibre dans les classes. Essayons d'équilibrer le jeu d'entrainement.

Modèle avec données équilibrées

```
train_data_balanced <- read.csv("data/train_data_balanced.csv", header = TRUE, sep = ",")
train_data_balanced$diagnosis <- as.factor(train_data_balanced$diagnosis)
```

```
rf_model_balanced <- randomForest(train_data_balanced$diagnosis ~ ., data = train_data_balanced, ntree =
```

```
plot(rf_model_balanced)
```



Le taux d'erreur semble s'être stabilisé.

```
pred_rf_balanced <- predict(rf_model_balanced, test_data, type="prob")
pred_rf_fact_balanced <- ifelse(pred_rf_balanced[,2] > 0.5, "M", "B")
prob_rf_balanced <- pred_rf_balanced[, "B"]
table(pred_rf_fact_balanced, test_data$diagnosis)
```

```
##
## pred_rf_fact_balanced  B  M
##                     B 78  6
##                     M  3 27
```

```
rf_accuracy_balanced <- mean(pred_rf_fact_balanced == test_data$diagnosis) # accuracy
rf_accuracy_balanced
```

```
## [1] 0.9210526
```

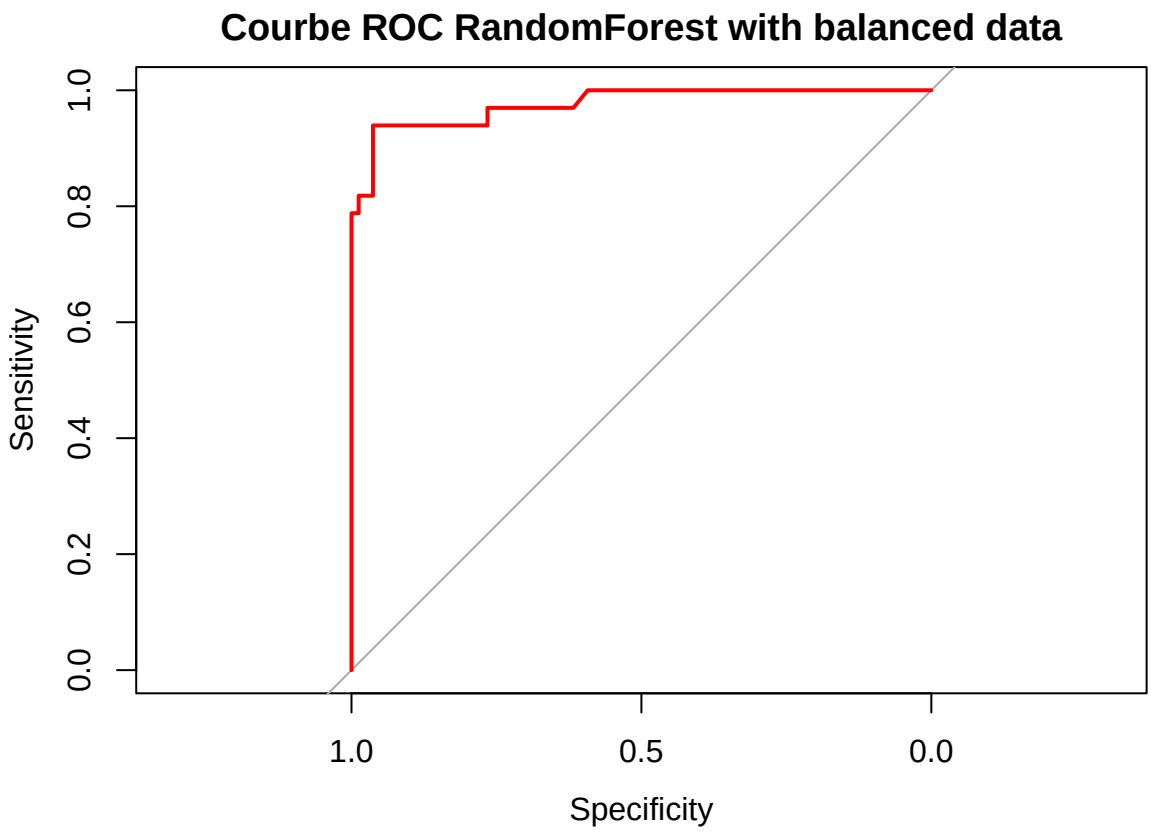
On obtient des performances similaires.

```
roc_rf_balanced <- roc(test_data$diagnosis, prob_rf_balanced)
```

```
## Setting levels: control = B, case = M
```

```
## Setting direction: controls > cases
```

```
plot(roc_rf_balanced, col = "red", AUC = TRUE, main = "Courbe ROC RandomForest with balanced data")
```




```
auc_rf_balanced <- auc(roc_rf_balanced)
auc_rf_balanced
```

```
## Area under the curve: 0.9761
```

Régression Logistique

Lancement du modèle

```
glm_model <- glm(diagnosis ~ ., data = train_data, family = binomial,control = glm.control(maxit = 50))
```

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

Il y a des warnings, ce qui signifie que le modèle n'est pas bien calibré.

```
summary(glm_model)
```

```
##
## Call:
```

```
## glm(formula = diagnosis ~ ., family = binomial, data = train_data,
##     control = glm.control(maxit = 50))
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -7.485e-06  -2.110e-08  -2.110e-08   2.110e-08   6.376e-06
##
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## (Intercept)              -9.329e+02  2.877e+07       0        1
## radius_mean               2.774e+02  4.530e+06       0        1
## texture_mean              7.408e+00  1.584e+05       0        1
## perimeter_mean           -4.363e+01  6.753e+05       0        1
## area_mean                 1.673e-01  2.183e+04       0        1
## smoothness_mean          -2.848e+03  7.949e+07       0        1
## compactness_mean         -1.647e+03  3.579e+07       0        1
## concavity_mean           -5.379e+02  1.613e+07       0        1
## concave.points_mean       4.974e+03  7.770e+07       0        1
## symmetry_mean            -3.795e+01  4.227e+07       0        1
## fractal_dimension_mean    6.281e+03  9.972e+07       0        1
## radius_se                 1.546e+03  1.939e+07       0        1
## texture_se                4.741e+01  2.238e+06       0        1
## perimeter_se             -1.366e+02  1.256e+06       0        1
## area_se                  -3.931e+00  1.295e+05       0        1
## smoothness_se            -2.454e+04  2.655e+08       0        1
## compactness_se           -1.251e+03  1.211e+08       0        1
## concavity_se              5.385e+02  6.007e+07       0        1
## concave.points_se         8.235e+03  9.647e+07       0        1
## symmetry_se              -1.094e+03  1.026e+08       0        1
## fractal_dimension_se     -4.290e+03  4.748e+08       0        1
## radius_worst             -1.891e+02  8.543e+05       0        1
## texture_worst             1.326e+00  2.978e+05       0        1
## perimeter_worst           1.938e+01  1.137e+05       0        1
## area_worst                9.042e-01  1.133e+04       0        1
## smoothness_worst          3.807e+03  4.315e+07       0        1
## compactness_worst        -3.655e+02  2.026e+07       0        1
## concavity_worst           4.967e+02  1.215e+07       0        1
## concave.points_worst     -5.145e+02  2.301e+07       0        1
## symmetry_worst            8.065e+02  2.182e+07       0        1
## fractal_dimension_worst   1.398e+03  6.463e+07       0        1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.0993e+02  on 454  degrees of freedom
## Residual deviance: 4.0275e-10  on 424  degrees of freedom
## AIC: 62
##
## Number of Fisher Scoring iterations: 31
```

Calcul des odds ratios (OR)

```r
exp(coef(glm_model))
```

```
##             (Intercept)             radius_mean            texture_mean
##            0.000000e+00           2.970743e+120           1.649803e+03
##          perimeter_mean               area_mean         smoothness_mean
##            1.127913e-19            1.182149e+00            0.000000e+00
##        compactness_mean          concavity_mean     concave.points_mean
##            0.000000e+00           2.540332e-234                     Inf
##           symmetry_mean  fractal_dimension_mean               radius_se
##            3.303280e-17                     Inf                     Inf
##              texture_se            perimeter_se                 area_se
##            3.900971e+20            4.791425e-60            1.962475e-02
##           smoothness_se          compactness_se            concavity_se
##            0.000000e+00            0.000000e+00          7.466853e+233
##       concave.points_se             symmetry_se    fractal_dimension_se
##                     Inf            0.000000e+00            0.000000e+00
##            radius_worst           texture_worst         perimeter_worst
##            7.667257e-83            3.764820e+00            2.616095e+08
##              area_worst        smoothness_worst       compactness_worst
##            2.469858e+00                     Inf           1.929610e-159
##         concavity_worst    concave.points_worst          symmetry_worst
##           5.313389e+215           3.556905e-224                     Inf
## fractal_dimension_worst
##                     Inf
```

On observe des extrêmes pour les odds ratios. Cela est dû à la présence de variables corrélées, nous allons donc plus tard simplifier le modèle.

Intérêt des variables explicatives

```r
res0 =glm(diagnosis ~ 1, family = "binomial", data=train_data)
anova(res0,glm_model,test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: diagnosis ~ 1
## Model 2: diagnosis ~ radius_mean + texture_mean + perimeter_mean + area_mean +
##     smoothness_mean + compactness_mean + concavity_mean + concave.points_mean +
##     symmetry_mean + fractal_dimension_mean + radius_se + texture_se +
##     perimeter_se + area_se + smoothness_se + compactness_se +
##     concavity_se + concave.points_se + symmetry_se + fractal_dimension_se +
##     radius_worst + texture_worst + perimeter_worst + area_worst +
##     smoothness_worst + compactness_worst + concavity_worst +
##     concave.points_worst + symmetry_worst + fractal_dimension_worst
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1       454     609.93
## 2       424       0.00 30   609.93 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

On observe que l'on a une p-value proche de 0, donc on rejette l'hypothèse nulle et on peut conclure qu'il y a au moins une variables explicative qui est significative.

Prédiction

```
pred_glm <- predict(glm_model, test_data, type = "response")
pred_glm_qual <- ifelse(pred_glm > 0.5, "M", "B")
table(pred_glm_qual, test_data$diagnosis)
```

```
##
## pred_glm_qual  B  M
##             B 75  7
##             M  6 26
```

Performance du modèle

```
glm_accuracy <- mean(pred_glm_qual == test_data$diagnosis) # accuracy
glm_accuracy
```

```
## [1] 0.8859649
```

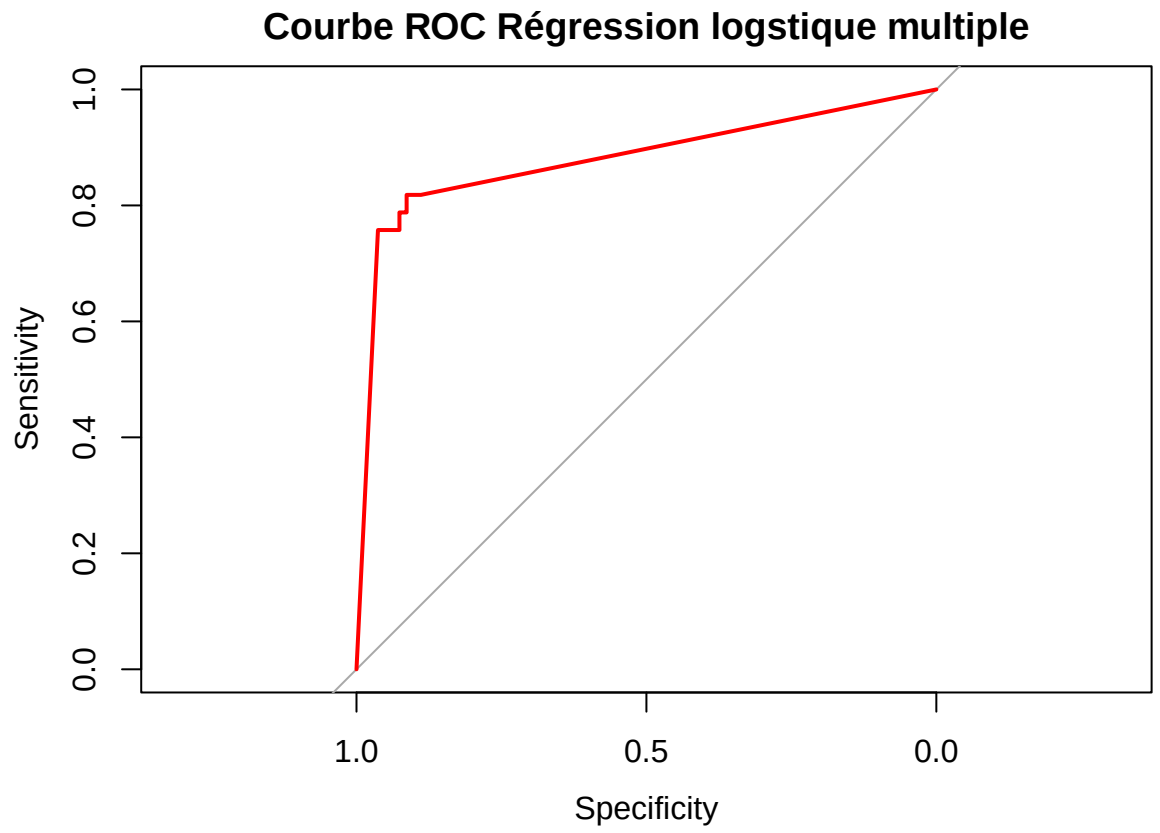
On obtient une accuracy de 88.6%, ce qui est plutôt bon mais pas aussi bon que les autres modèles.

```
roc_glm <- roc(test_data$diagnosis, pred_glm)
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_glm, col = "red", AUC = TRUE, main = "Courbe ROC Régression logstique multiple")
```




```
auc_glm <- auc(roc_glm)
auc_glm
```

```
## Area under the curve: 0.8801
```

Simplification du modèle avec des régressions logistiques pénalisées

Nous allons simplifier le modèle de régression logistique pour supprimer les variables inutiles. Pour se faire, nous allons réaliser une régression de type Ridge et de type Lasso

```
# install.packages("glmnet")
library(glmnet)
```

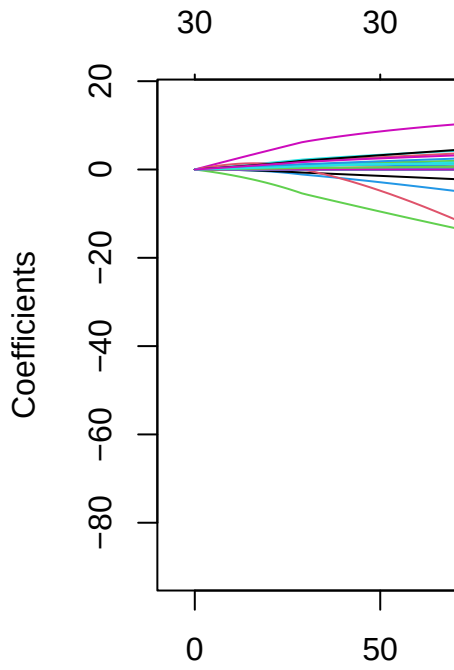
```
## Le chargement a nécessité le package : Matrix
```

```
## Loaded glmnet 4.1-8
```

```
ridge_model <- glmnet(as.matrix(train_data[, -1]), train_data$diagnosis, alpha = 0, family = "binomial"
lasso_model <- glmnet(as.matrix(train_data[, -1]), train_data$diagnosis, alpha = 1, family = "binomial"
```

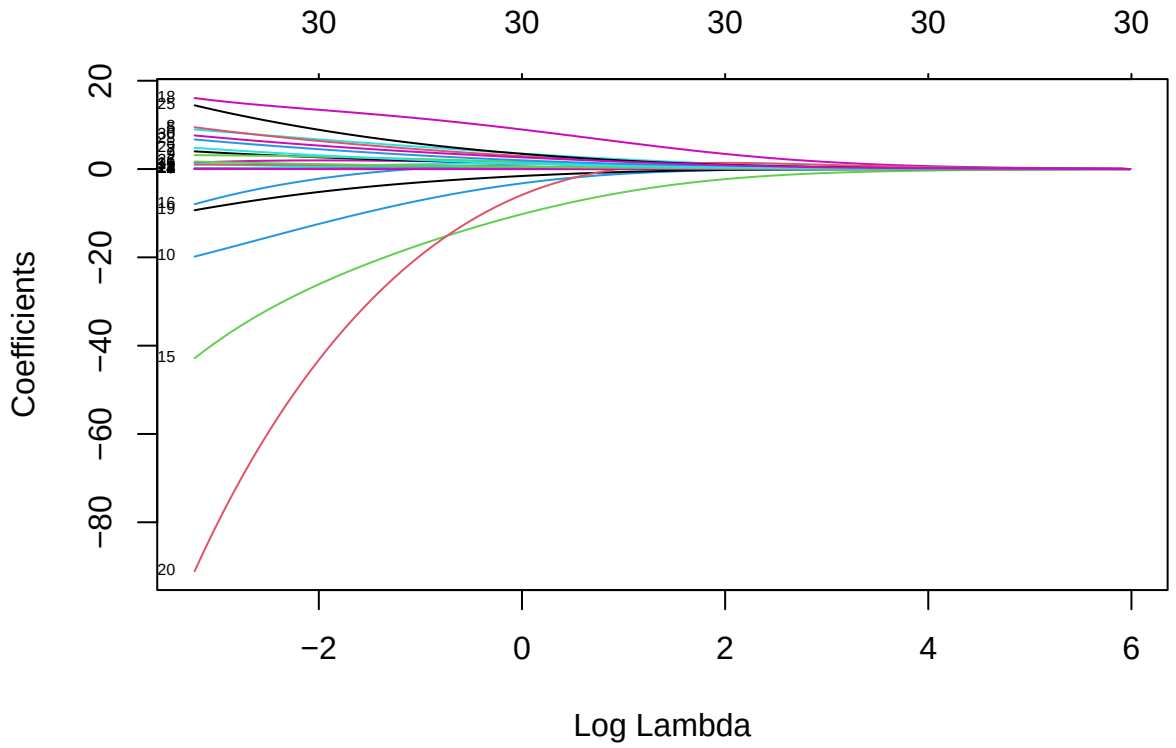
Lancement des algorithmes L'algorithme a bien convergé.

```r
plot(ridge_model, label = TRUE)
```

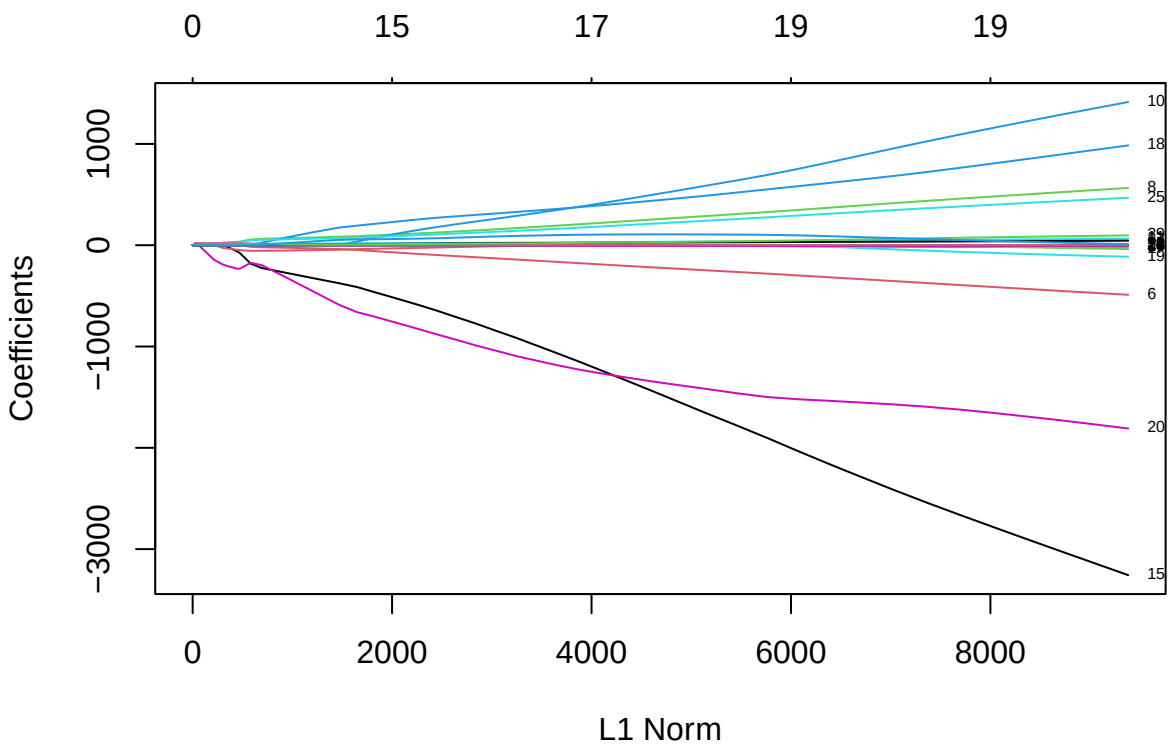


Visualisation des chemins de régularisation des estimateurs `ridge` et `lasso`

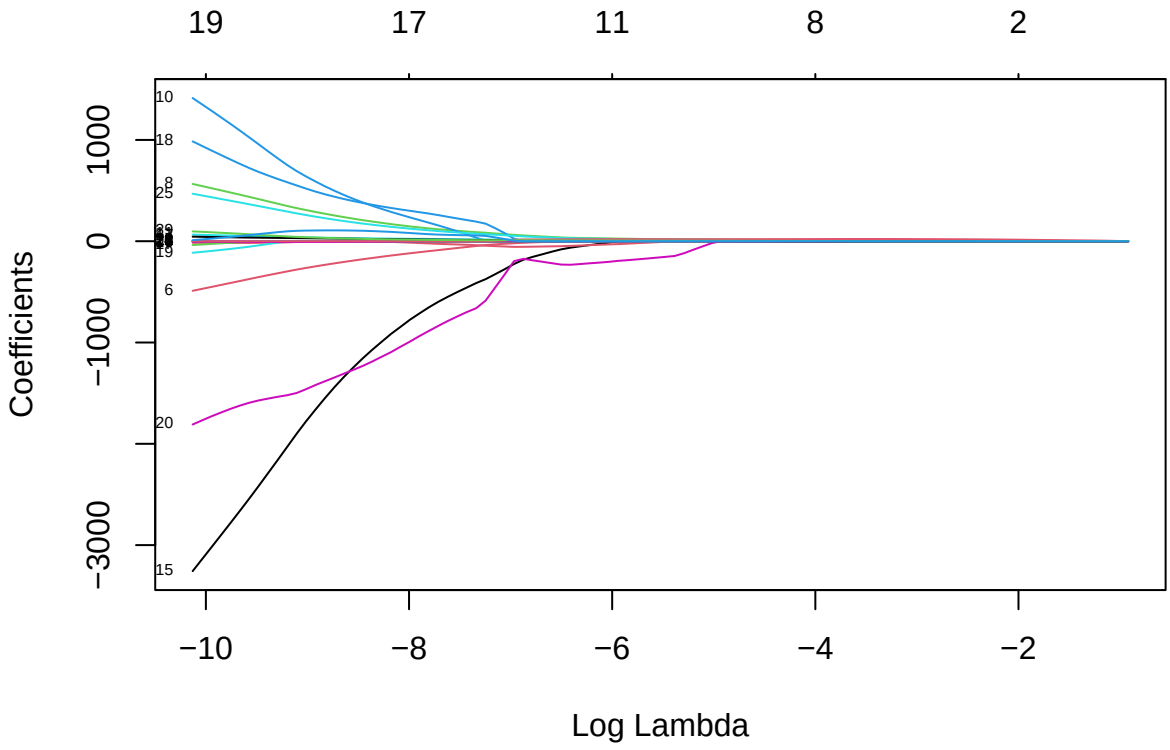
```r
plot(ridge_model, xvar = "lambda", label = TRUE)
```




```
plot(lasso_model, label = TRUE)
```




```
plot(lasso_model, xvar = "lambda", label = TRUE)
```




```
pred_ridge <- predict(ridge_model, s = 0.01, newx = as.matrix(test_data[, -1]), type = "response")
pred_ridge_qual <- ifelse(pred_ridge > 0.5, "M", "B")
table(pred_ridge_qual, test_data$diagnosis)
```

Prédictions et performances

```
##
## pred_ridge_qual  B  M
##               B 81  6
##               M  0 27
```

```
ridge_accuracy <- mean(pred_ridge_qual == test_data$diagnosis) # accuracy
ridge_accuracy
```

```
## [1] 0.9473684
```

```
pred_lasso <- predict(lasso_model, s = 0.01, newx = as.matrix(test_data[, -1]), type = "response")
pred_lasso_qual <- ifelse(pred_lasso > 0.5, "M", "B")
table(pred_lasso_qual, test_data$diagnosis)
```

```
##
```

```
## pred_lasso_qual  B  M
##               B 81  6
##               M  0 27
```

```
lasso_accuracy <- mean(pred_lasso_qual == test_data$diagnosis) # accuracy
lasso_accuracy
```

```
## [1] 0.9473684
```

```
roc_ridge <- roc(test_data$diagnosis, pred_ridge)
```

```
## Setting levels: control = B, case = M
```

```
## Warning in roc.default(test_data$diagnosis, pred_ridge): Deprecated use a
## matrix as predictor. Unexpected results may be produced, please pass a numeric
## vector.
```

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

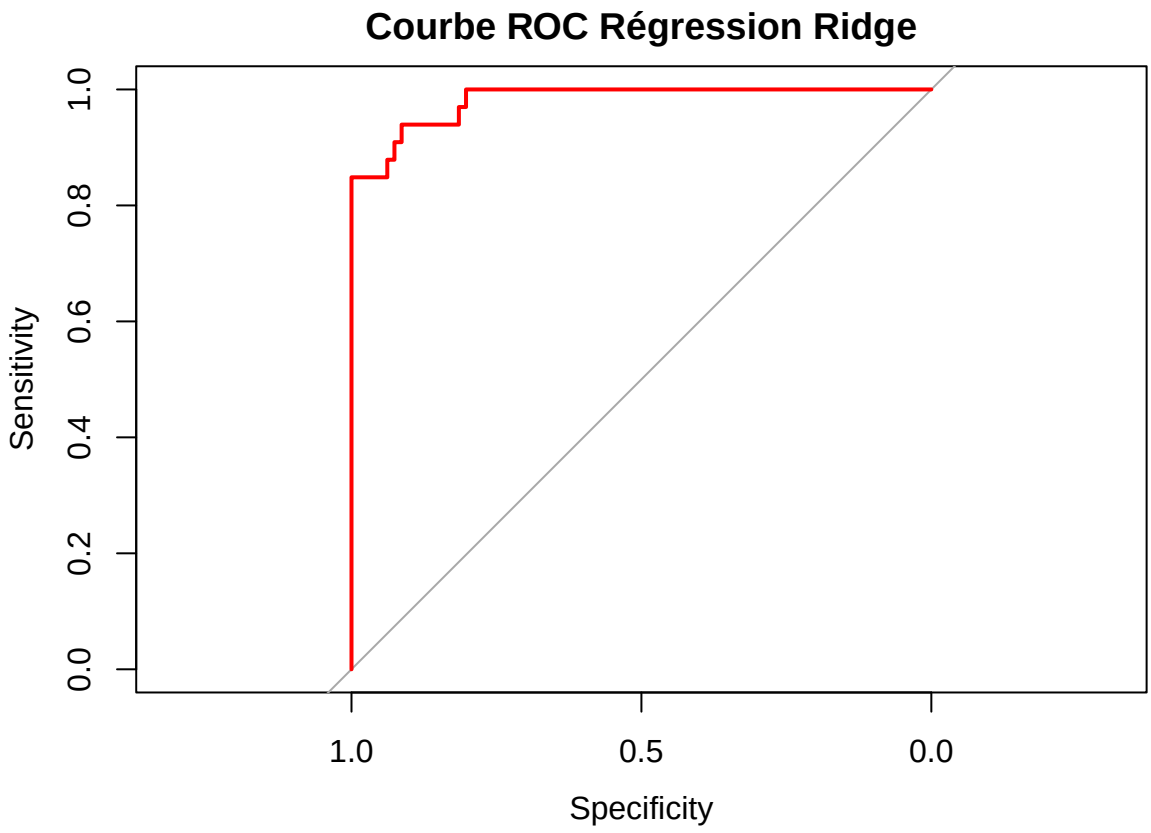
```
roc_lasso <- roc(test_data$diagnosis, pred_lasso)
```

```
## Setting levels: control = B, case = M
```

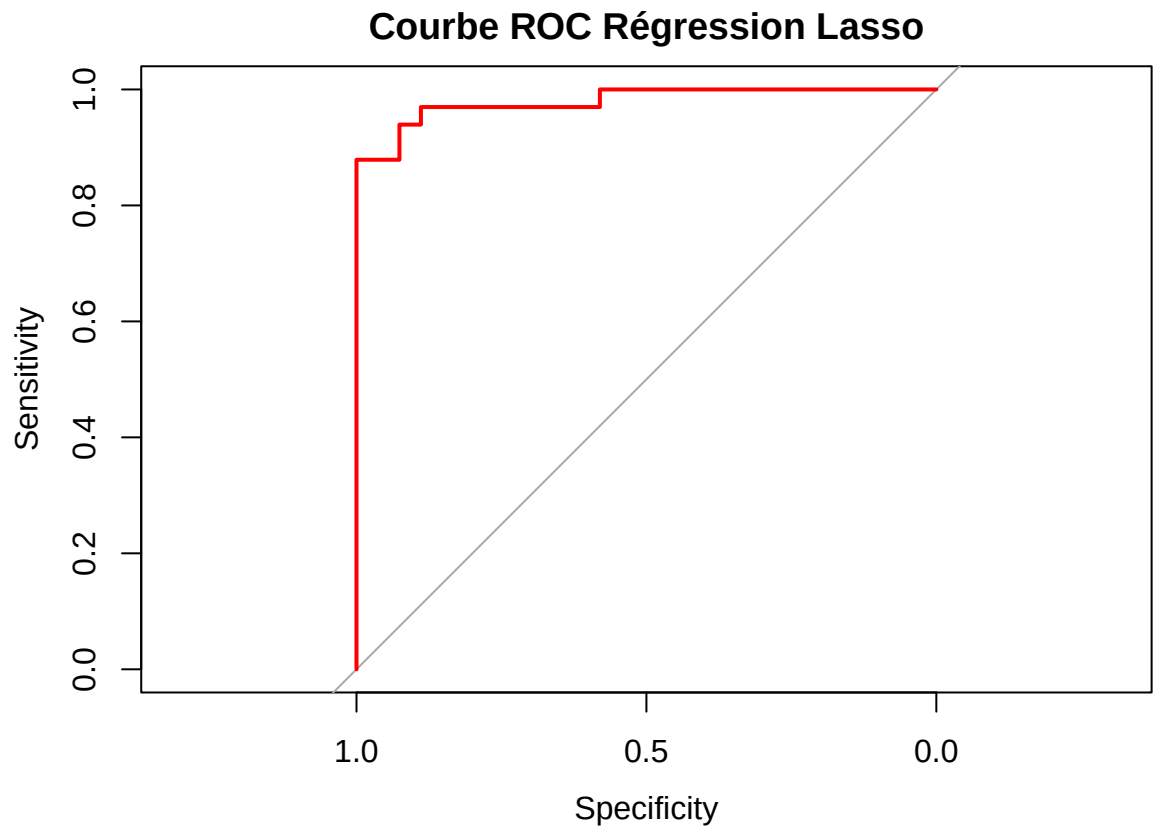
```
## Warning in roc.default(test_data$diagnosis, pred_lasso): Deprecated use a
## matrix as predictor. Unexpected results may be produced, please pass a numeric
## vector.
```

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

```
plot(roc_ridge, col = "red", AUC = TRUE, main = "Courbe ROC Régression Ridge")
```




```r
plot(roc_lasso, col = "red", AUC = TRUE, main = "Courbe ROC Régression Lasso")
```




```
auc_ridge <- auc(roc_ridge)
auc_lasso <- auc(roc_lasso)
auc_ridge
```

```
## Area under the curve: 0.9817
```

```
auc_lasso
```

```
## Area under the curve: 0.9794
```

```
sum(coef(lasso_model, s=exp(-6))!=0)
```

Nombre de variables sélectionnées

```
## [1] 12
```

```
sum(coef(ridge_model, s=exp(-6))!=0)
```

```
## [1] 31
```

On a sélectionné beaucoup moins de variables pour le modèle Lasso, qui a des performances similaires.

SVM

Nous avons décidé d'expérimenter un modèle SVM pour voir si les performances sont meilleures, comme nous l'avons fait lors de nos projets industriels.

Lancement du modèle

```
# install.packages("e1071")
library(e1071)
```

```
svm_lin_model <- svm(diagnosis ~ ., data = train_data, kernel = "linear",  probability = TRUE )
svm_rbf_model <- svm(diagnosis ~ ., data = train_data, kernel = "radial",  probability = TRUE )
```

Prédiction

```
pred_svm_lin <- predict(svm_lin_model, test_data, probability = TRUE)
pred_svm_lin_prob <- attr(pred_svm_lin, "probabilities")
table(pred_svm_lin, test_data$diagnosis)
```

```
##
## pred_svm_lin  B  M
##            B 79  5
##            M  2 28
```

```
pred_svm_rbf <- predict(svm_rbf_model, test_data, probability = TRUE)
pred_svm_rbf_prob <- attr(pred_svm_rbf, "probabilities")
table(pred_svm_rbf, test_data$diagnosis)
```

```
##
## pred_svm_rbf  B  M
##            B 77  3
##            M  4 30
```

Performance du modèle

```
svm_lin_accuracy <- mean(pred_svm_lin == test_data$diagnosis) # accuracy
svm_lin_accuracy
```

```
## [1] 0.9385965
```

```
svm_rbf_accuracy <- mean(pred_svm_rbf == test_data$diagnosis) # accuracy
svm_rbf_accuracy
```

```
## [1] 0.9385965
```

```r
roc_svm_lin <- roc(test_data$diagnosis, pred_svm_lin_prob[,2])
```

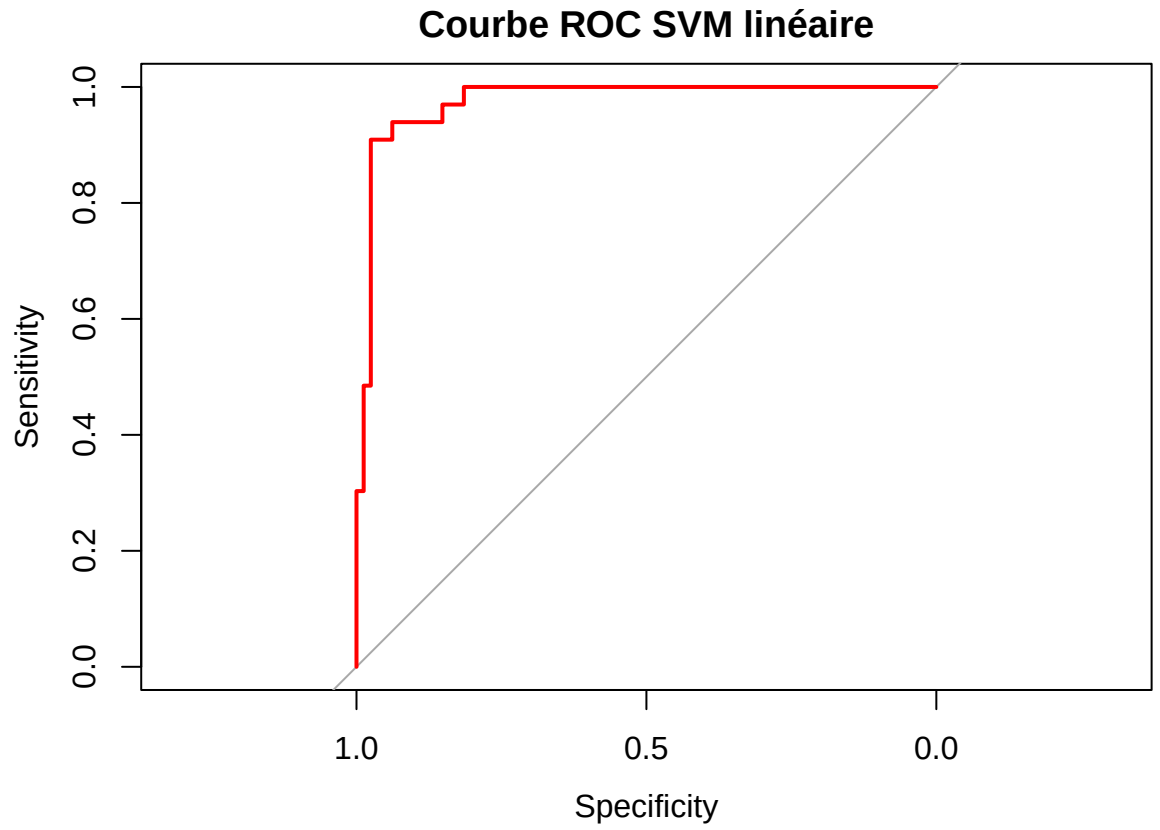
```
## Setting levels: control = B, case = M
```

```
## Setting direction: controls > cases
```

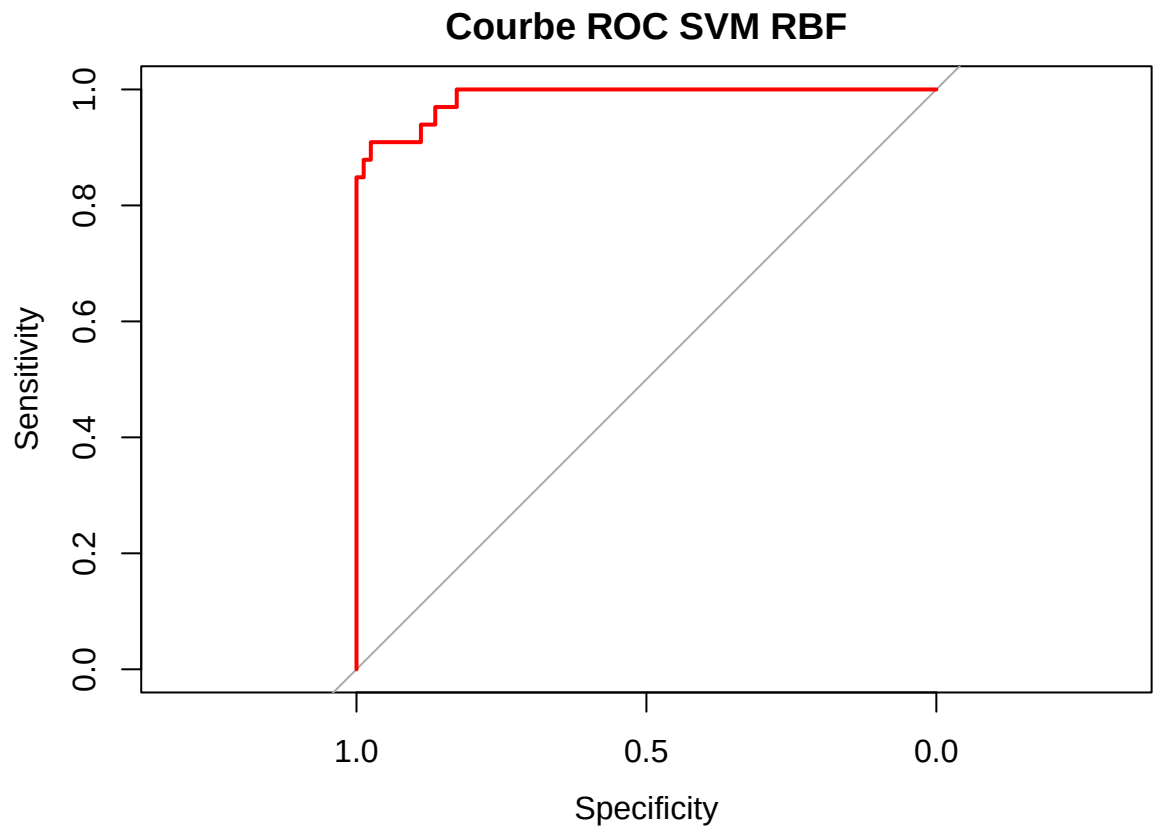
```r
roc_svm_rbf <- roc(test_data$diagnosis, pred_svm_rbf_prob[,2])
```

```
## Setting levels: control = B, case = M
## Setting direction: controls > cases
```

```r
plot(roc_svm_lin, col = "red", AUC = TRUE, main = "Courbe ROC SVM linéaire")
```




```r
plot(roc_svm_rbf, col = "red", AUC = TRUE, main = "Courbe ROC SVM RBF")
```




```
auc_svm_lin <- auc(roc_svm_lin)
auc_svm_rbf <- auc(roc_svm_rbf)
auc_svm_lin
```

```
## Area under the curve: 0.9753
```

```
auc_svm_rbf
```

```
## Area under the curve: 0.9862
```

On obtient des AUC très satisfaisantes.

ADABoost

Nous avons décidé d'expérimenter un modèle ADABoost pour voir si les performances sont meilleures, comme nous l'avons fait lors de nos projets industriels.

Lancement du modèle

```
# install.packages("gbm")
library(gbm)
```

```
## Loaded gbm 2.1.9

## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.c
```

On lance un modèle non-calibré pour voir les performances de base.

```
ada_model <- gbm(as.numeric(diagnosis)-1 ~ ., data = train_data, distribution = "adaboost")
ada_model
```

```
## gbm(formula = as.numeric(diagnosis) - 1 ~ ., distribution = "adaboost",
##     data = train_data)
## A gradient boosted model with adaboost loss function.
## 100 iterations were performed.
## There were 30 predictors of which 15 had non-zero influence.
```

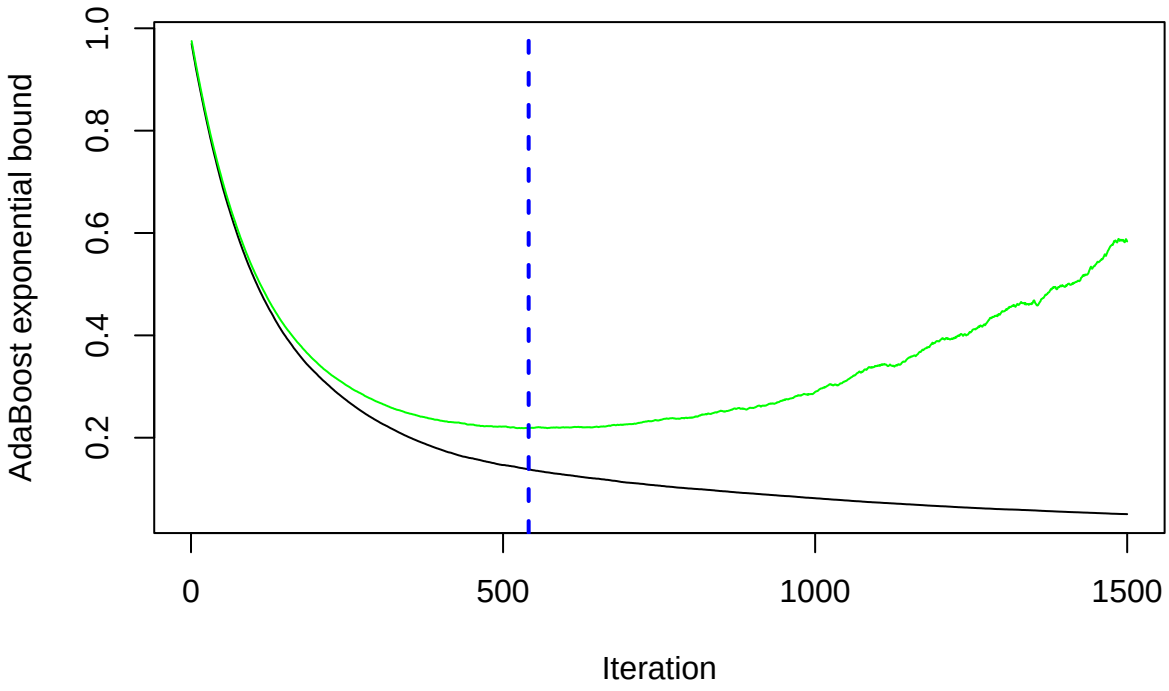
Calibration du modèle

```
ada_cv_adadist_model <- gbm(as.numeric(diagnosis)-1 ~., train_data, distribution = "adaboost",cv.folds =
ada_cv_adadist_model
```

Calibration pour une distribution adaboost

```
## gbm(formula = as.numeric(diagnosis) - 1 ~ ., distribution = "adaboost",
##     data = train_data, n.trees = 1500, shrinkage = 0.01, cv.folds = 5)
## A gradient boosted model with adaboost loss function.
## 1500 iterations were performed.
## The best cross-validation iteration was 541.
## There were 30 predictors of which 16 had non-zero influence.
```

```
B.opt <- gbm.perf(ada_cv_adadist_model, method = "cv")
```



En noir, on a la perte pour le jeu "données", et en vert la perte estimée par validation croisée (c'est l'erreur de généralisation). Le trait vertical en pointillé donne le nombre d'itérations qui minimise la perte estimée par validation croisée. Si on prend un nombre d'itérations plus grand, on va faire du sur-apprentissage : on apprendra très bien sur le jeu de donnees, mais le modèle qu'on aura appris ne va pas bien se généraliser à de nouvelles données.

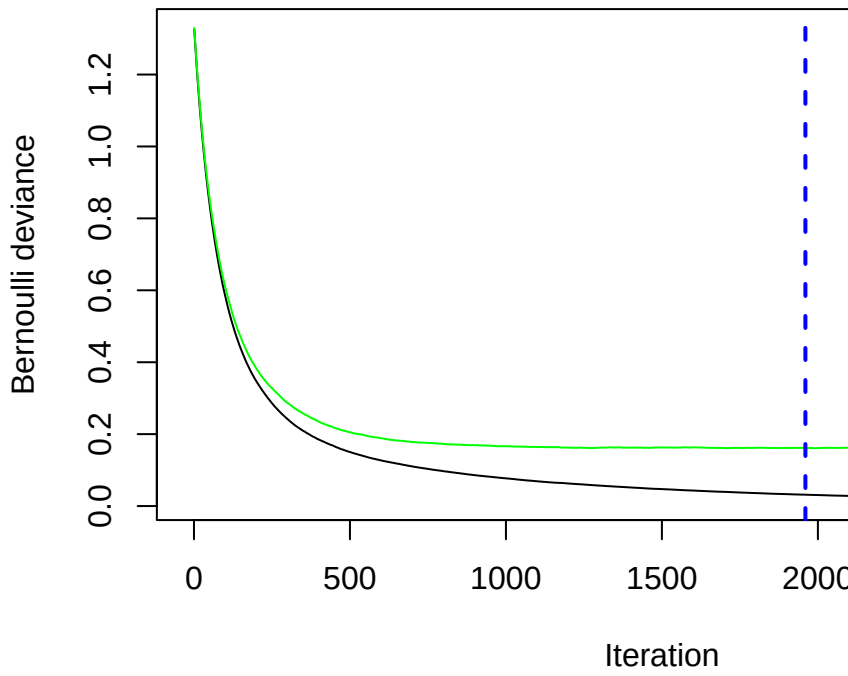
```
B.opt
```

```
## [1] 541
```

On remarque que l'erreur de généralisation tend à augmenter avec le nombre d'itérations. Cela signifie que le modèle fait du sur-apprentissage. La valeur optimale obtenue pour le nombre d'itérations est de 548.

```
ada_cv_berndist_model <- gbm(as.numeric(diagnosis)-1 ~., train_data, distribution = "bernoulli",cv.folds
```

```
B.opt <- gbm.perf(ada_cv_berndist_model, method = "cv")
```



Calibration pour une distribution bernoulli

```
B.opt
```

```
## [1] 1960
```

On obtient une valeur optimale de 1035 pour le nombre d'itérations.

Prédiction

```
pred_ada <- predict(ada_model, test_data, n.trees = 548)
```

ADABoost sans cross-validation

```
## Warning in predict.gbm(ada_model, test_data, n.trees = 548): Number of trees
## not specified or exceeded number fit so far. Using 100.
```

```
pred_ada_qual <- ifelse(pred_ada > 0.5, "M", "B")
table(pred_ada_qual, test_data$diagnosis)
```

```
##
## pred_ada_qual  B  M
##             B 79  8
##             M  2 25
```

```r
ada_accuracy <- mean(pred_ada_qual == test_data$diagnosis) # accuracy
ada_accuracy
```

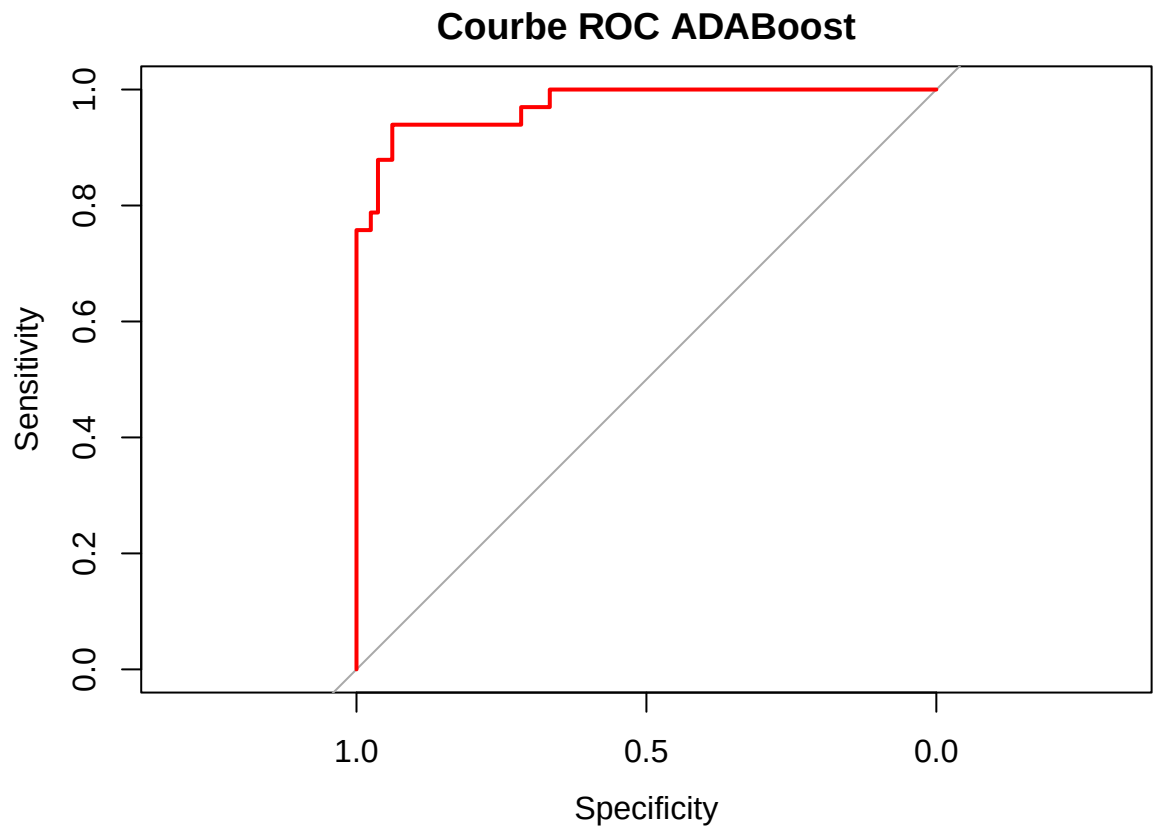
```
## [1] 0.9122807
```

```r
roc_ada <- roc(test_data$diagnosis, pred_ada)
```

```
## Setting levels: control = B, case = M
```

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

```r
plot(roc_ada, col = "red", AUC = TRUE, main = "Courbe ROC ADABoost")
```




```r
auc_ada <- auc(roc_ada)
auc_ada
```

```
## Area under the curve: 0.9734
```

```r
pred_ada_cv_adadist <- predict(ada_cv_adadist_model, test_data, n.trees = 548)
pred_ada_cv_adadist_qual <- ifelse(pred_ada_cv_adadist > 0.5, "M", "B")
table(pred_ada_cv_adadist_qual, test_data$diagnosis)
```

ADABoost avec cross-validation, distribution adaboost

```
##
## pred_ada_cv_adadist_qual  B  M
##                        B 79  8
##                        M  2 25
```

```
ada_cv_adadist_accuracy <- mean(pred_ada_cv_adadist_qual == test_data$diagnosis) # accuracy
ada_cv_adadist_accuracy
```

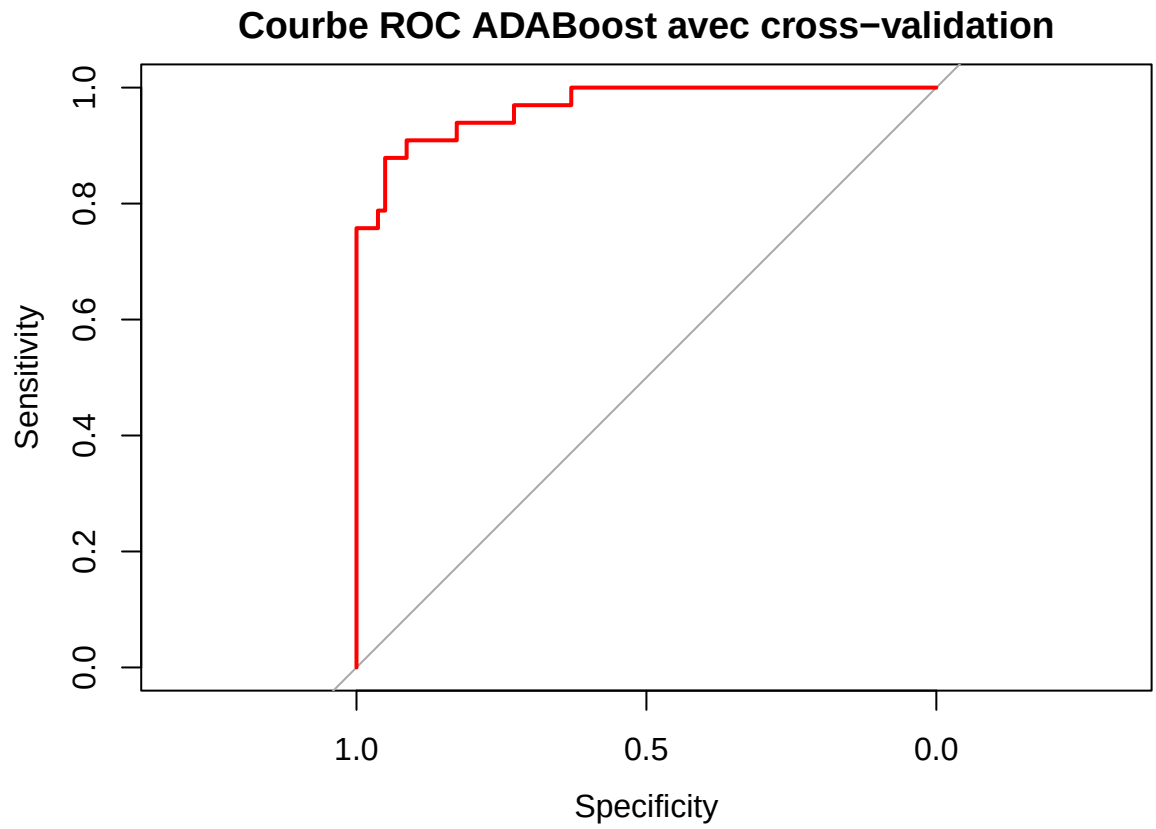
```
## [1] 0.9122807
```

```
roc_ada_cv_adadist <- roc(test_data$diagnosis, pred_ada_cv_adadist)
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_ada_cv_adadist, col = "red", AUC = TRUE, main = "Courbe ROC ADABoost avec cross-validation")
```




```
auc_ada_cv_adadist <- auc(roc_ada_cv_adadist)
auc_ada_cv_adadist
```

```
## Area under the curve: 0.9671
```

```
pred_ada_cv_berndist <- predict(ada_cv_berndist_model, test_data, n.trees = 1035)
pred_ada_cv_berndist_qual <- ifelse(pred_ada_cv_berndist > 0.5, "M", "B")
table(pred_ada_cv_berndist_qual, test_data$diagnosis)
```

ADABoost avec cross-validation, distribution Bernoulli

```
##
## pred_ada_cv_berndist_qual  B  M
##                         B 79  7
##                         M  2 26
```

```
ada_cv_berndist_accuracy <- mean(pred_ada_cv_berndist_qual == test_data$diagnosis) # accuracy
ada_cv_berndist_accuracy
```

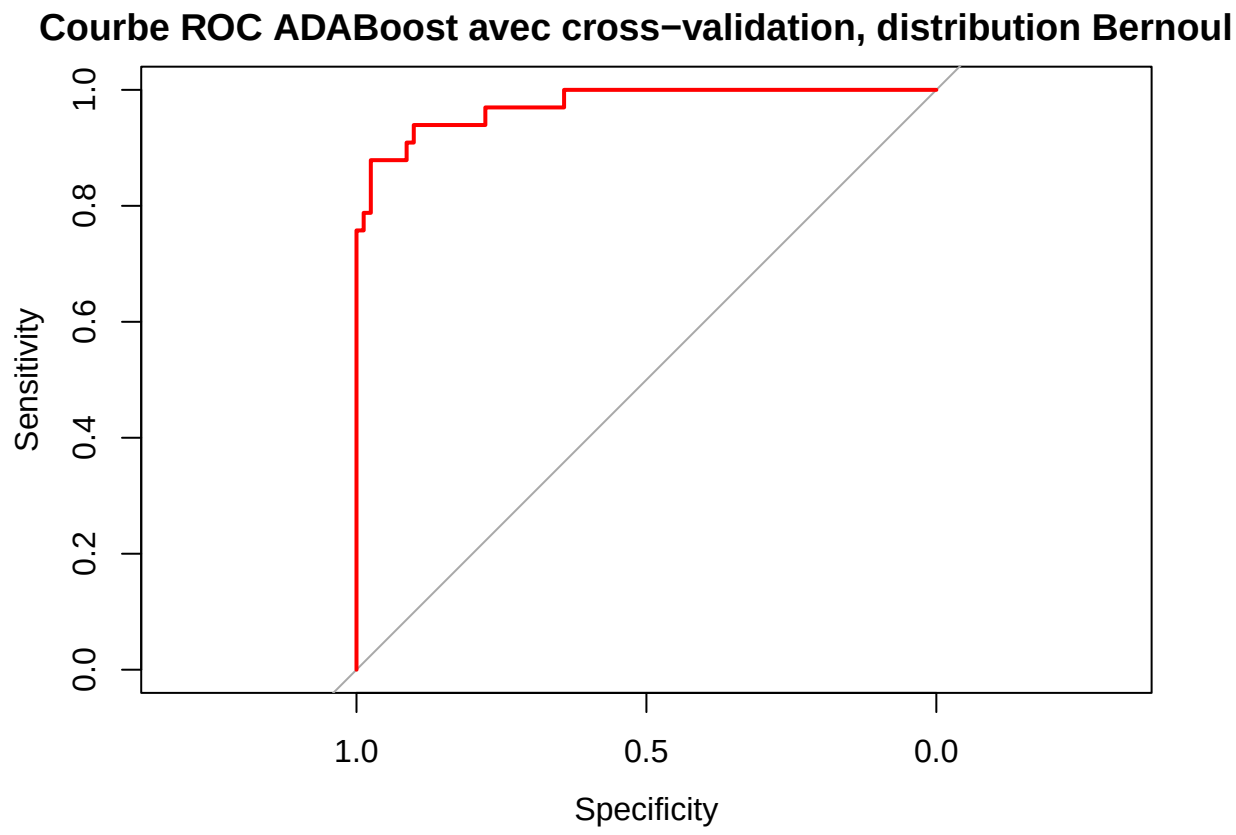
```
## [1] 0.9210526
```

```
roc_ada_cv_berndist <- roc(test_data$diagnosis, pred_ada_cv_berndist)
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_ada_cv_berndist, col = "red", AUC = TRUE, main = "Courbe ROC ADABoost avec cross-validation, di
```



```r
auc_ada_cv_berndist <- auc(roc_ada_cv_berndist)
auc_ada_cv_berndist
```

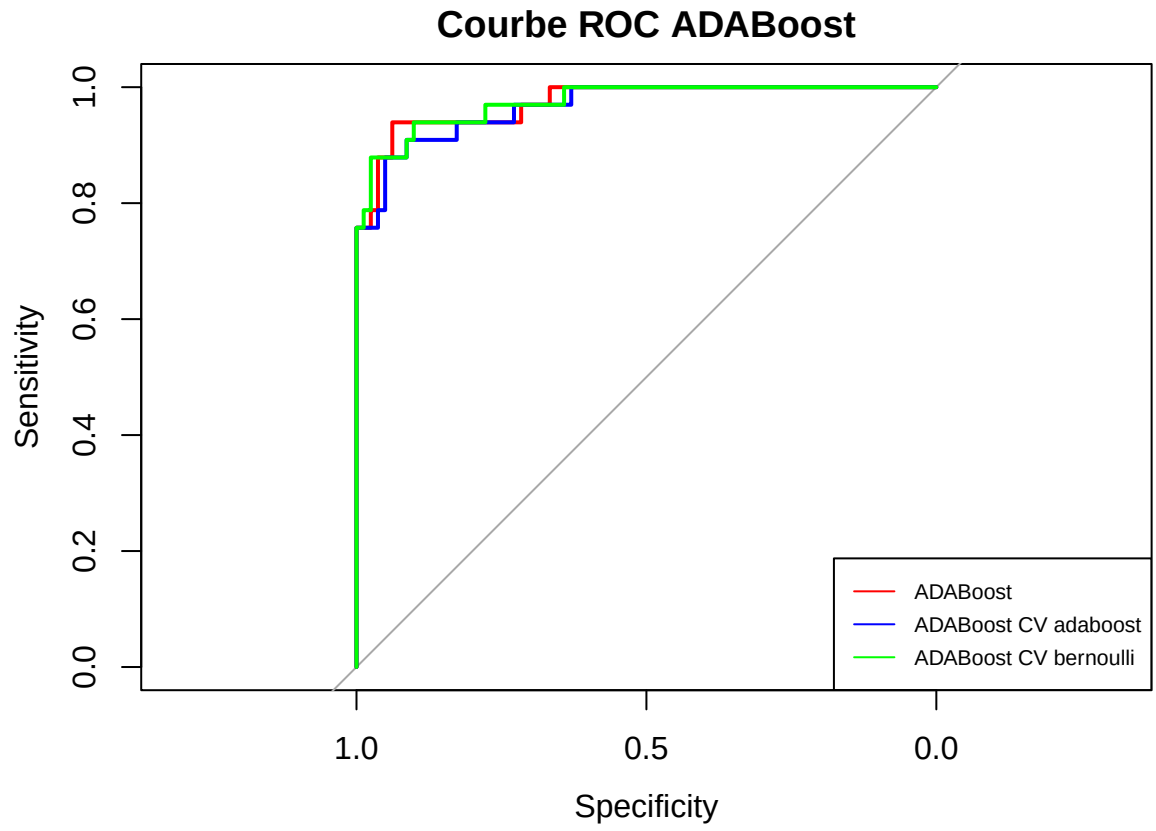
```
## Area under the curve: 0.9742
```

Comparaison des modèles ADABoost

```r
df_ada <- data.frame(model = c("ADABoost", "ADABoost CV adaboost", "ADABoost CV bernoulli"), accuracy =
df_ada
```

```
##                   model  accuracy       auc
## 1              ADABoost 0.9122807 0.9734381
## 2  ADABoost CV adaboost 0.9122807 0.9670782
## 3 ADABoost CV bernoulli 0.9210526 0.9741863
```

```r
plot(roc_ada, col = "red", AUC = TRUE, main = "Courbe ROC ADABoost")
plot(roc_ada_cv_adadist, col = "blue", AUC = TRUE, add = TRUE)
plot(roc_ada_cv_berndist, col = "green", AUC = TRUE, add = TRUE)
legend("bottomright", legend = c("ADABoost", "ADABoost CV adaboost", "ADABoost CV bernoulli"), col = c(
```



On obtient la même accuracy pour les trois modèles. Cependant, le modèle ADABoost avec cross-validation et distribution Bernoulli a une AUC légèrement supérieure, et les trois modèles ont des AUC très proches et satisfaisantes.

Evaluation des modèles

Courbes ROC

```
# install.packages("pROC")
library("pROC")
```

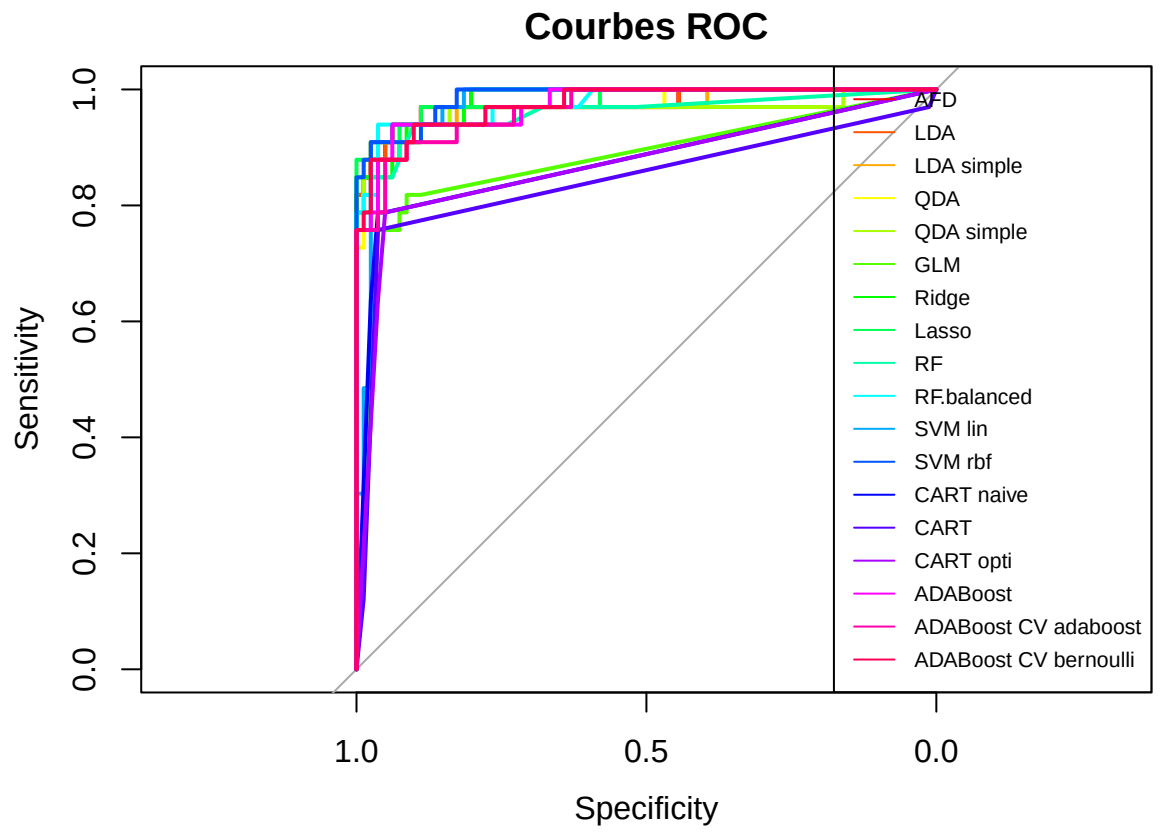
Calcul des courbes ROC

```
roc_list <- list(roc_afd, roc_lda, roc_lda_simple, roc_qda, roc_qda_simple, roc_glm, roc_ridge, roc_las
```

```
legends_list <- c("AFD", "LDA", "LDA simple", "QDA", "QDA simple", "GLM", "Ridge", "Lasso", "RF", "RF.b
```

Affichage des courbes ROC

```
plot(roc_afd, col = "red", main = "Courbes ROC")
cols <- rainbow(length(roc_list))
j <- 1
for (i in roc_list) {
  plot(i, add = TRUE, col = cols[j], label = legends_list[j])
  j <- j + 1
}
legend("bottomright", legend = legends_list, col = cols, lty = 1, cex = 0.7)
```



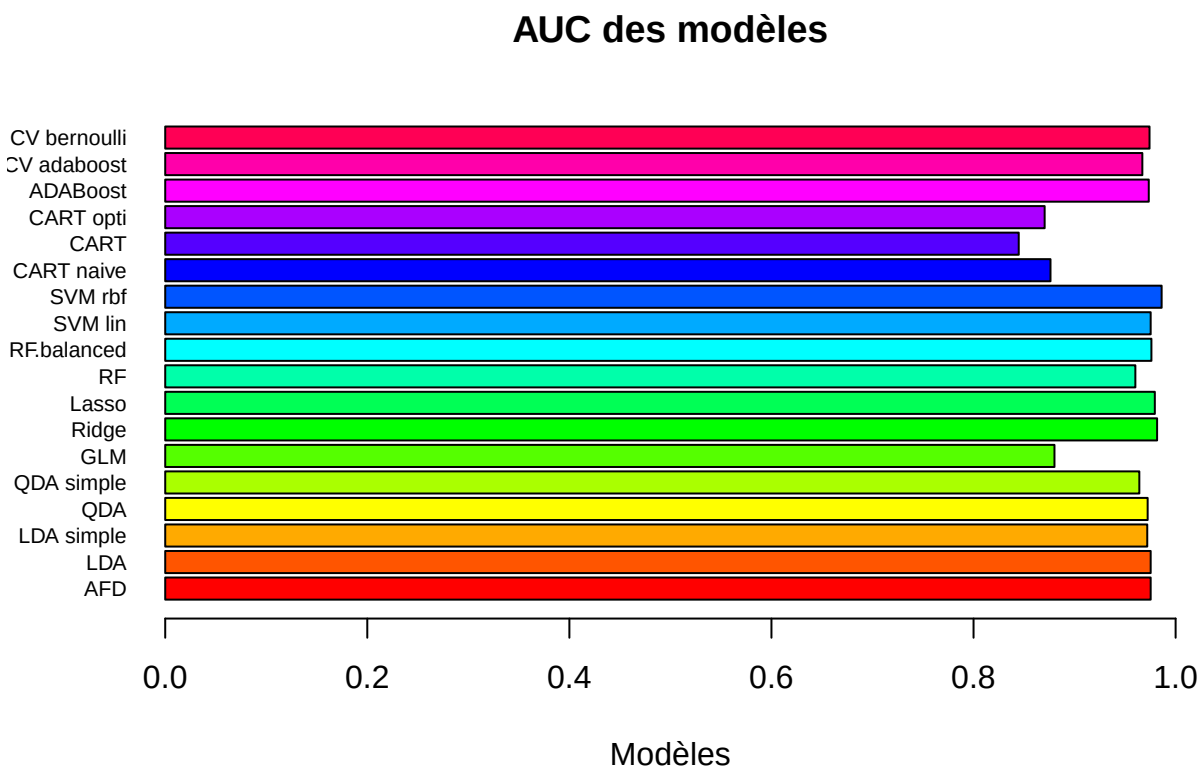
AUC

```
df.auc <- data.frame(model = legends_list, auc = sapply(roc_list, function(x) auc(x)))
df.auc
```

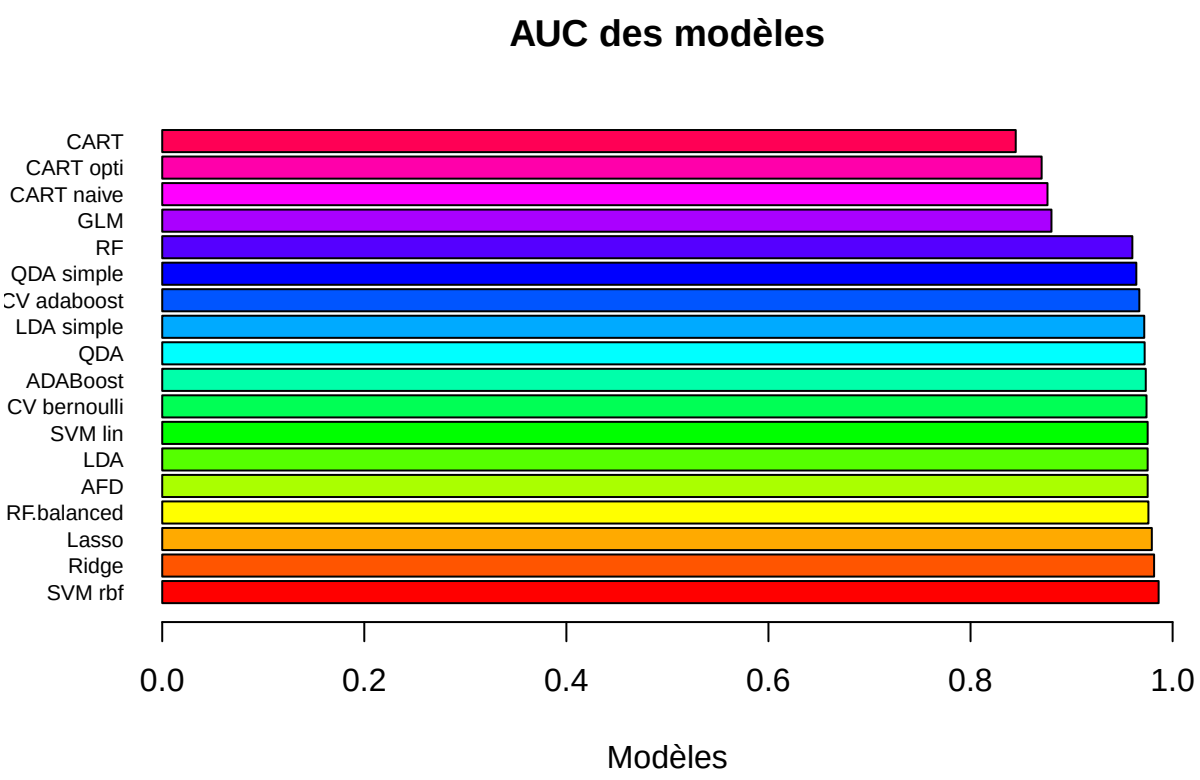
```
##                    model       auc
## 1                    AFD 0.9753086
## 2                    LDA 0.9753086
## 3             LDA simple 0.9719416
## 4                    QDA 0.9723158
## 5             QDA simple 0.9640853
## 6                    GLM 0.8800973
## 7                  Ridge 0.9816685
## 8                  Lasso 0.9794239
## 9                     RF 0.9601571
## 10           RF.balanced 0.9760569
## 11               SVM lin 0.9753086
## 12               SVM rbf 0.9861579
## 13            CART naive 0.8761691
## 14                  CART 0.8447437
## 15             CART opti 0.8703704
## 16              ADABoost 0.9734381
## 17  ADABoost CV adaboost 0.9670782
## 18 ADABoost CV bernoulli 0.9741863
```

```r
# Sauvegarde des AUC dans un fichier csv
write.csv(df.auc, "data/auc.csv")
```

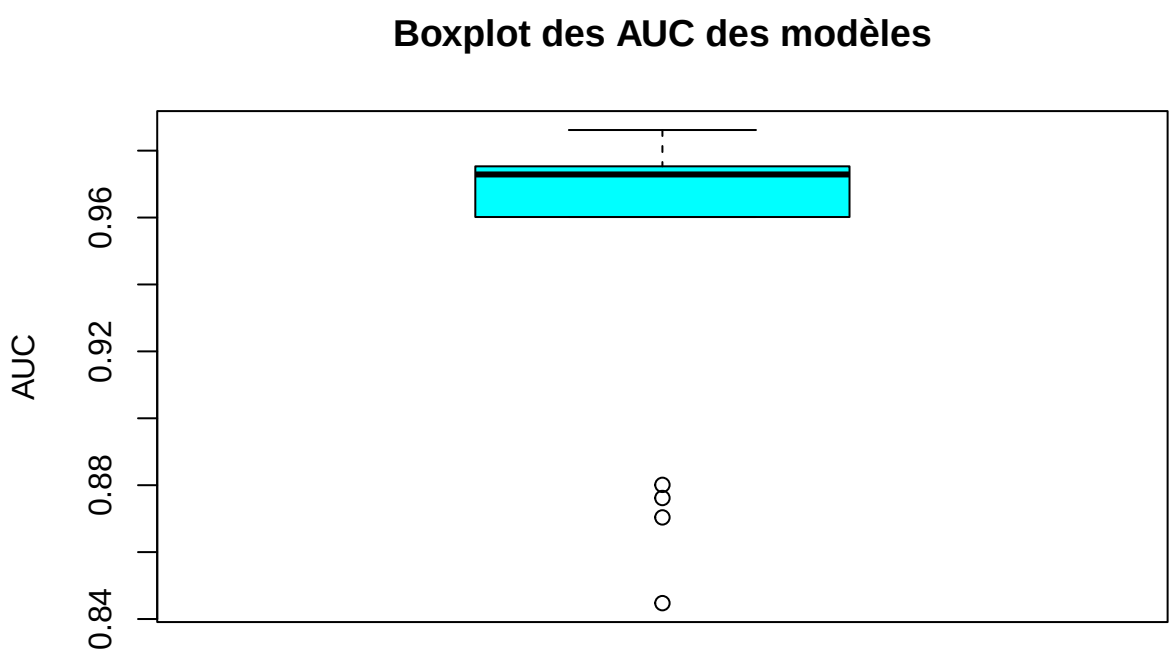
```r
barplot(df.auc$auc, names.arg = df.auc$model, col = rainbow(length(df.auc$auc)), main = "AUC des modèles
```




```r
df.auc_sorted <- df.auc[order(df.auc$auc, decreasing = TRUE),]
barplot(df.auc_sorted$auc, names.arg = df.auc_sorted$model, col = rainbow(length(df.auc_sorted$auc)), ma
```




```r
boxplot(df.auc$auc, main = "Boxplot des AUC des modèles", ylab = "AUC", col = "cyan")
```



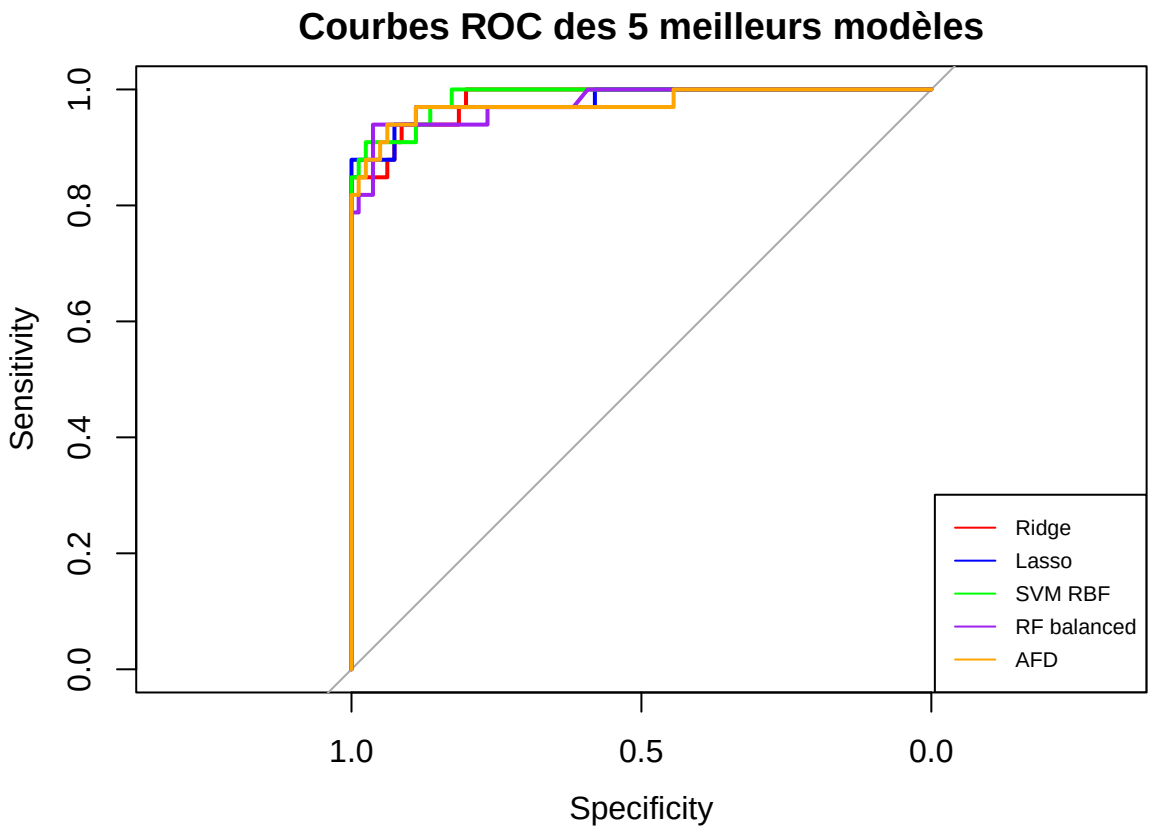

```
summary(df.auc$auc)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8447  0.9611  0.9729  0.9502  0.9753  0.9862
```

On obtient que le *meilleur modèle* relativement à l'AUC trouvé est le SVM avec noyau RBF (non-linéaire). Le meilleur modèle trouvé que l'on a vu dans ce cours est le modèle de régression logistique avec pénalisation de Ridge, avec une AUC de 0.981. De plus, le troisième meilleur modèle, à savoir la régression Lasso, a une AUC assez proche de celle de la régression Ridge, tout en dépendant de moins de variables.

Les *modèles les moins bons* sont les modèles CART, qui ont des AUC inférieures à 0.9, et de régression logistique non pénalisée, qui ont des performances très inférieures aux autres.

```
# Courbes ROC des 5 meilleurs modèles

plot(roc_ridge, col = "red", AUC = TRUE, main = "Courbes ROC des 5 meilleurs modèles")
plot(roc_lasso, col = "blue", AUC = TRUE, add = TRUE)
plot(roc_svm_rbf, col = "green", AUC = TRUE, add = TRUE)
plot(roc_rf_balanced, col = "purple", AUC = TRUE, add = TRUE)
plot(roc_afd, col = "orange", AUC = TRUE, add = TRUE)
legend("bottomright", legend = c("Ridge", "Lasso", "SVM RBF", "RF balanced", "AFD"), col = c("red", "blu
```



Comparaison de l'accuarcy des modèles

```
df_accuracy <- data.frame(model = c("AFD", "LDA", "LDA.simple", "QDA", "QDA.simple", "GLM", "Ridge", "La
```

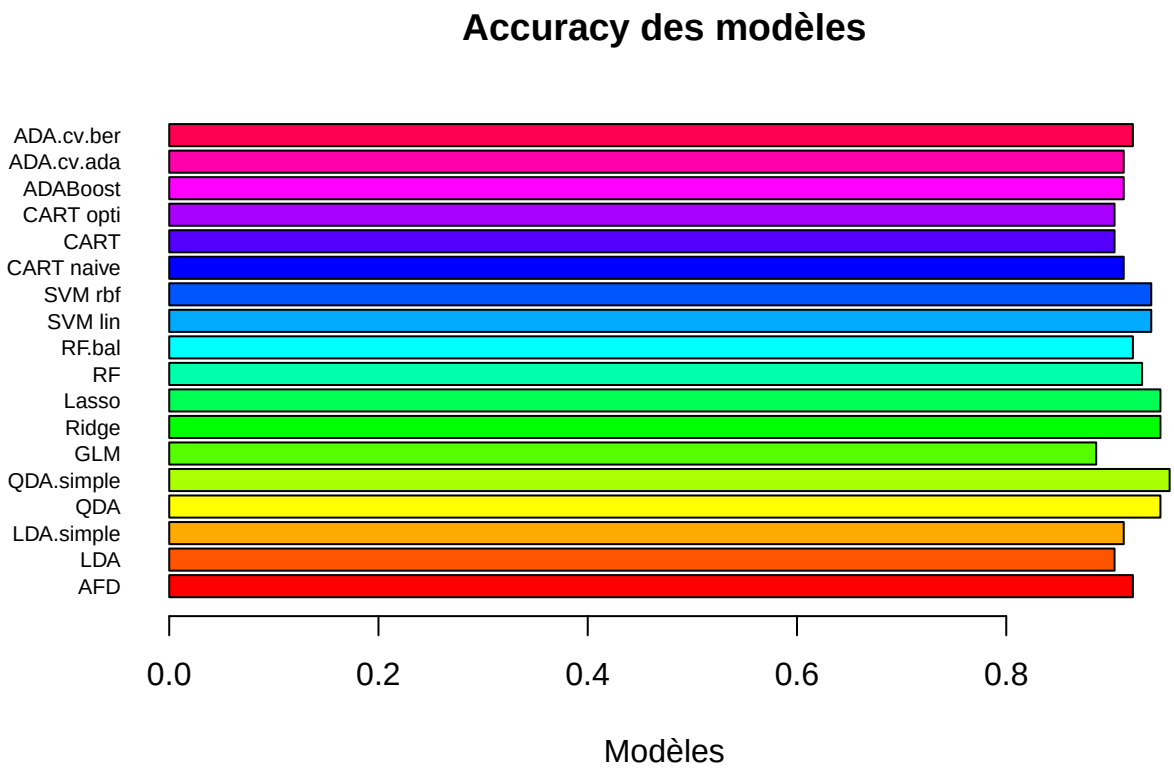
```
df_accuracy
```

```
##         model  accuracy
## 1         AFD 0.9210526
## 2         LDA 0.9035088
## 3  LDA.simple 0.9122807
## 4         QDA 0.9473684
## 5  QDA.simple 0.9561404
## 6         GLM 0.8859649
## 7       Ridge 0.9473684
## 8       Lasso 0.9473684
## 9          RF 0.9298246
## 10     RF.bal 0.9210526
## 11    SVM lin 0.9385965
## 12    SVM rbf 0.9385965
## 13 CART naive 0.9122807
## 14       CART 0.9035088
## 15  CART opti 0.9035088
## 16   ADABoost 0.9122807
```

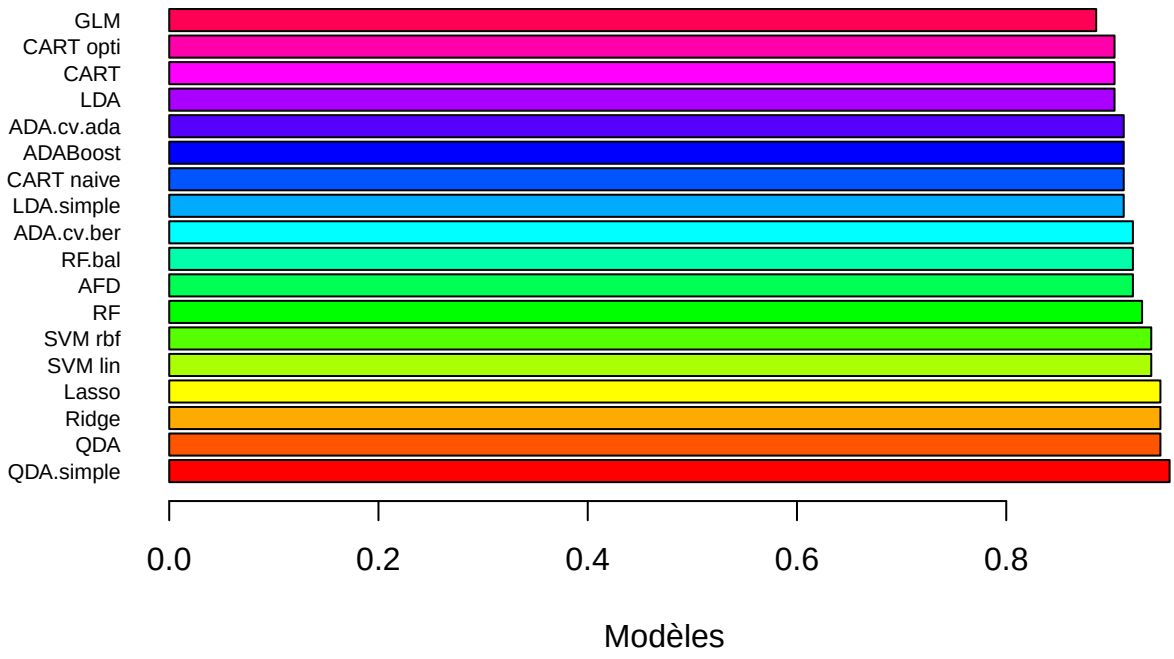
```
## 17 ADA.cv.ada 0.9122807
## 18 ADA.cv.ber 0.9210526
```

```r
# Sauvegarde des accuracy dans un fichier csv
write.csv(df_accuracy, "data/accuracy.csv")
```

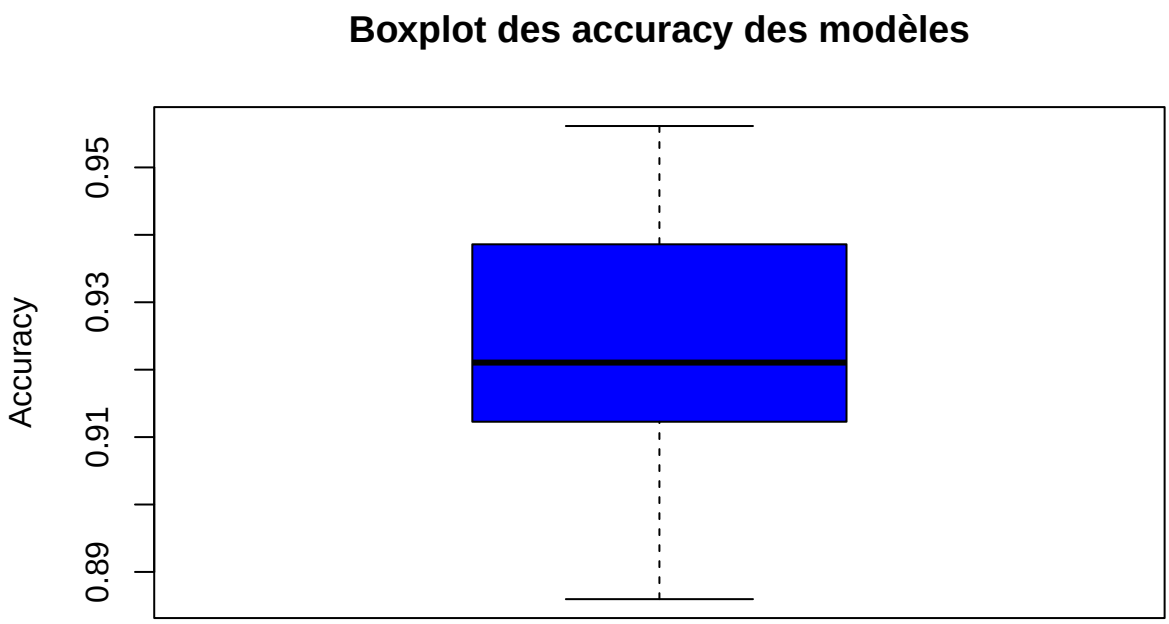
```r
barplot(df_accuracy$accuracy, names.arg = df_accuracy$model, col = rainbow(length(df_accuracy$accuracy)
```




```r
df_accuracy_sorted <- df_accuracy[order(df_accuracy$accuracy, decreasing = TRUE),]
barplot(df_accuracy_sorted$accuracy, names.arg = df_accuracy_sorted$model, col = rainbow(length(df_accu
```




```
boxplot(df_accuracy$accuracy, main = "Boxplot des accuracy des modèles", ylab = "Accuracy", col = "blue
```




```
summary(df_accuracy$accuracy)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8860  0.9123  0.9211  0.9230  0.9386  0.9561
```

- Meilleurs Modèles* : Les modèles avec les meilleures accuracy sont les modèles de QDA, de régression logistique avec pénalisation de Ridge et de régression Lasso. Comme la QDA n'est pas dans les meilleurs modèles en terme d'AUC, on peut dire que les modèles de régression logistique avec pénalisation de Ridge et de régression Lasso sont les meilleurs modèles de statistiques prédictives que l'on a trouvé. Une très bonne accuracy pour la QDA peut être un signe de sur-apprentissage.
- Modèles moins efficaces* : Le modèle qui a le moins fonctionné est le modèle de régression logistique sans pénalisation. En effet, lorsque nous lancions l'analyse, nous obtenions des warnings ce qui signifiait que le modèle n'était pas bien calibré. Le modèle CART a également eu des performances moins intéressantes que l'on a pu observer avec son accuracy plus basse que les autres, tout comme le LDA.

DataFrame du récapitulatif

```
df <- data.frame(
  model = c("AFD", "LDA", "LDA.simple", "QDA", "QDA.simple", "GLM", "Ridge", "Lasso", "RF", "RF.bal", "
  accuracy = c(afd_accuracy, lda_accuracy, lda_simple_accuracy, qda_accuracy, qda_simple_accuracy, glm_
  auc = c(auc_afd, auc_lda, auc_lda_simple, auc_qda, auc_qda_simple, auc_glm, auc_ridge, auc_lasso, auc
```

```r
write.csv(df, "data/summary.csv")
```

```r
summary(df)
```

```
##     model              accuracy           auc        
##  Length:18          Min.   :0.8860   Min.   :0.8447  
##  Class :character   1st Qu.:0.9123   1st Qu.:0.9611  
##  Mode  :character   Median :0.9211   Median :0.9729  
##                     Mean   :0.9230   Mean   :0.9502  
##                     3rd Qu.:0.9386   3rd Qu.:0.9753  
##                     Max.   :0.9561   Max.   :0.9862
```

Interprétation des résultats

Les modèles de régression logistique avec pénalisation de Ridge et de régression Lasso sont les meilleurs modèles de statistiques prédictives que l'on a trouvé. En effet, ils ont les meilleures AUC et les meilleures accuracy. De plus, le modèle de régression Lasso dépend de moins de variables que le modèle de régression Ridge, ce qui peut être un avantage.

Le modèle de SVM avec noyau RBF est le meilleur modèle en terme d'AUC, mais il n'est pas le meilleur en terme d'accuracy.

Le modèle de régression logistique sans pénalisation et le modèle CART sont les moins bons modèles que l'on a testés Cela peut s'expliquer par une mauvaise calibration pour le modèle de régression logistique.

Conclusion

Le data-set est exploitable en machine learning, et nous avons pu obtenir des modèles de statistiques prédictives qui ont de bonnes performances. Les modèles de régression logistique avec pénalisation de Ridge et de régression Lasso sont les meilleurs modèles que l'on a trouvé. Ces modèles pourraient permettre de prédire le diagnostic de patientes atteintes de cancer du sein, dans une certaine mesure.