PROJET Régression Pénalisée et Réduction de la dimension

DELWENDE NOAGA DAMIEN MASSIMBO & SARRA TRABELSI

07/12/2020

Chargement des librairies

```
library(data.table)
library(tinytex)
library(tidyverse)
library(ggplot2)
library(ISLR)
library(ggplot2)
library(reshape2)
library(plotly)
library(ggpubr)
library(htmltools)
library(glmnet)
```

Objectif

Dans notre projet, nous allons mettre en place les methodes vues en cours telles que :

- La régression pénalisée (ridge et lasso)
- La réduction de la dimension (régression sur composantes principales et PLS)

Alors, ces methodes seront appliquées sur le fichier **graisse2.txt** dans le but de prédire bel et bien notre variable d'intérêt **y=graisse (teneur en graisse observée des patients)** en prenant en compte toutes les autres variables.

Sans plus tarder, nous ferons une analyse descriptive sur la variable à prédire. **y=graisse**

Analyse Descriptive

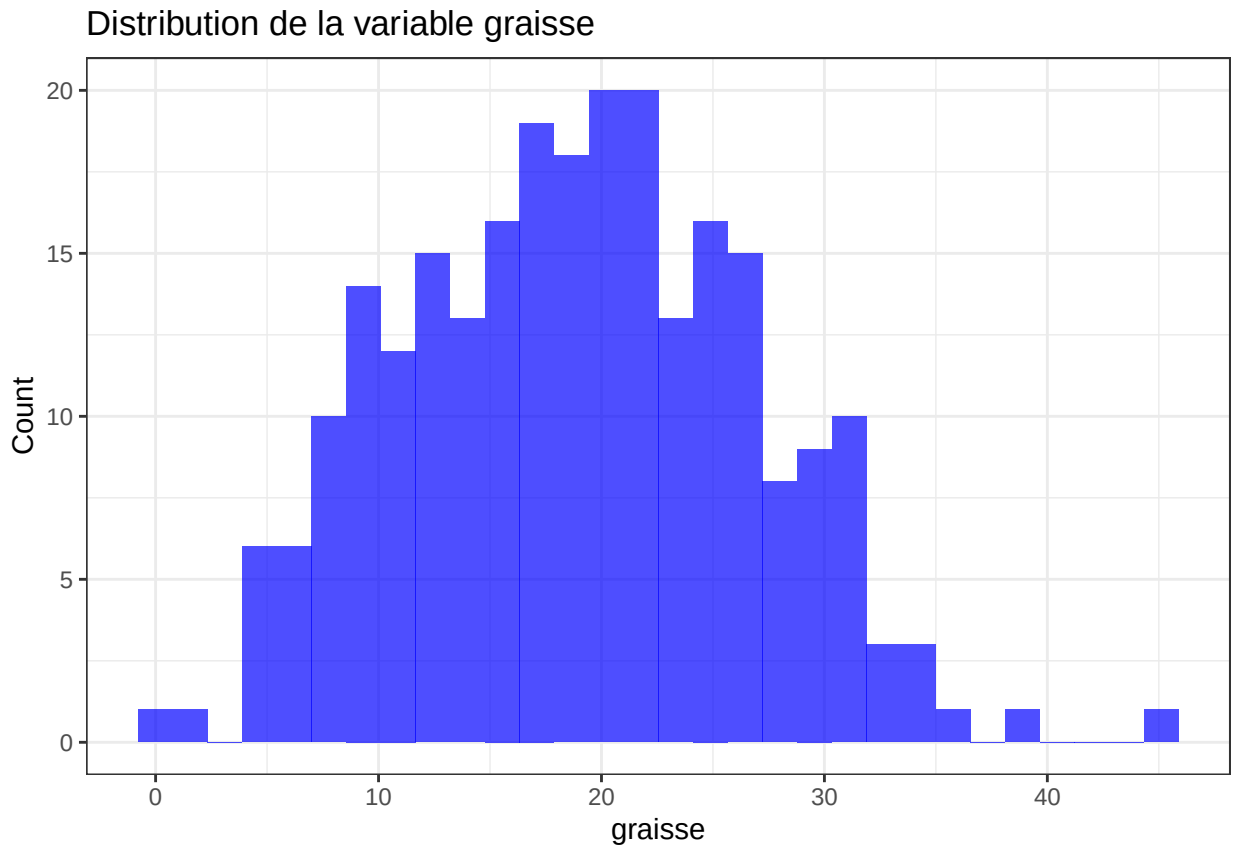
Notre jeu de données est composé de 251 observations de 15 variables toutes quantitatives à partir de la commande **str** de R.

```
tgraisse <- read.table("graisse2.txt", header = T)
dim(tgraisse)
```

```
## [1] 251  15
```

Analyse sur notre variable à prédire : y = graisse

```
ggplot(tgraisse, aes(x = tgraisse$graisse))+
  geom_histogram(fill="blue", alpha=0.7)+
  theme_bw()+
  labs(x="graisse", y= "Count", title = "Distribution de la variable graisse")
```



On constate que notre variable d'intérêt "graisse" a une distribution proche de la loi normale, par ailleurs, en regardant sur l'axe des abscisses, on constate qu'une des observation de notre variable à predire se distingue des autres avec une valeur supérieur à environ 45. Nous chercherons par la suite à savoir quel est est l'individu ayant cette observation élévée, et nous changerons la forme de la distribution de la variable "graisse" pour voir si nous avions un changement où pas en absence de cet individu.

```
ind <- which(tgraisse[,"graisse"]>45)
ind
```

```
## [1] 215
```

```
max(tgraisse[-ind,"graisse"])
```

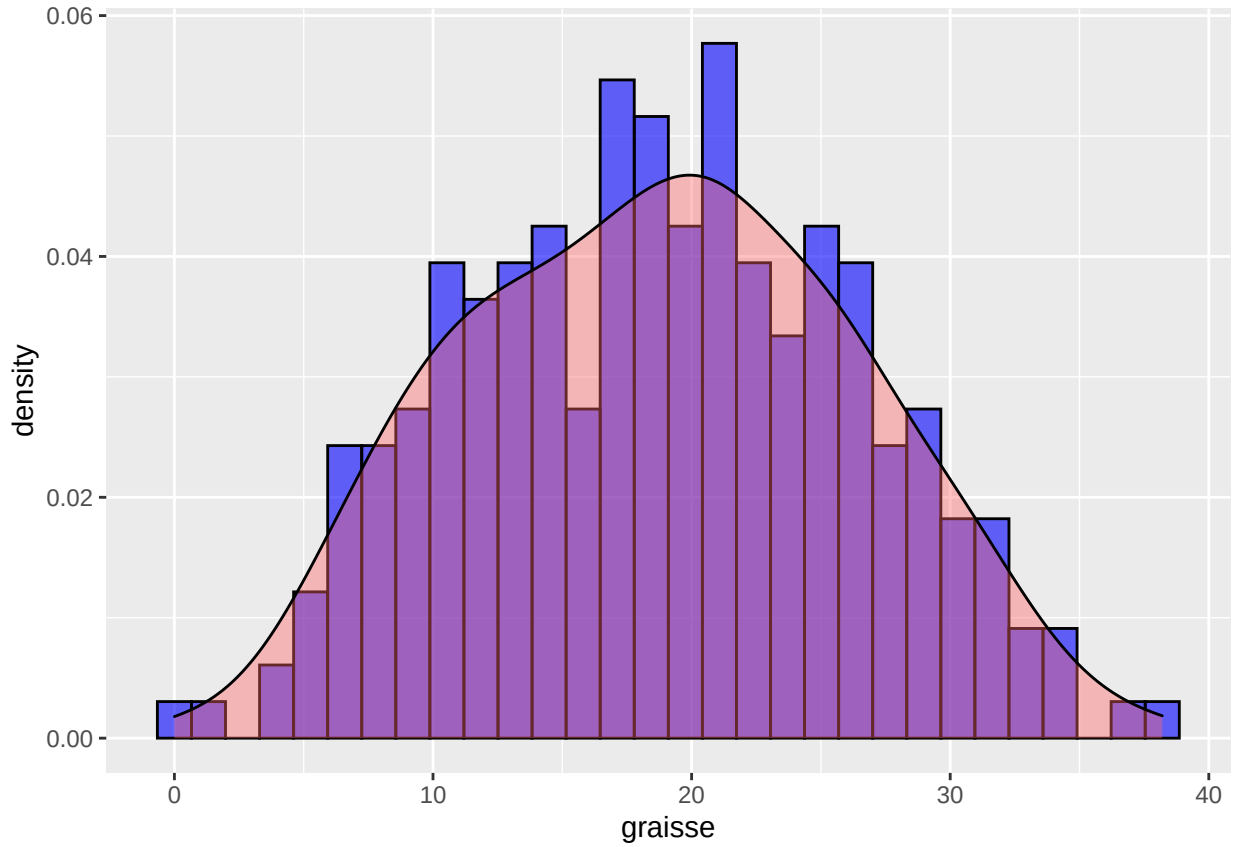
```
## [1] 38.2
```

```
tgraisse1<-tgraisse[-215,]
dim(tgraisse1)
```

```
## [1] 250  15
```

Distribution de notre variable à prédire sans l'individu ayant l'observation extrême

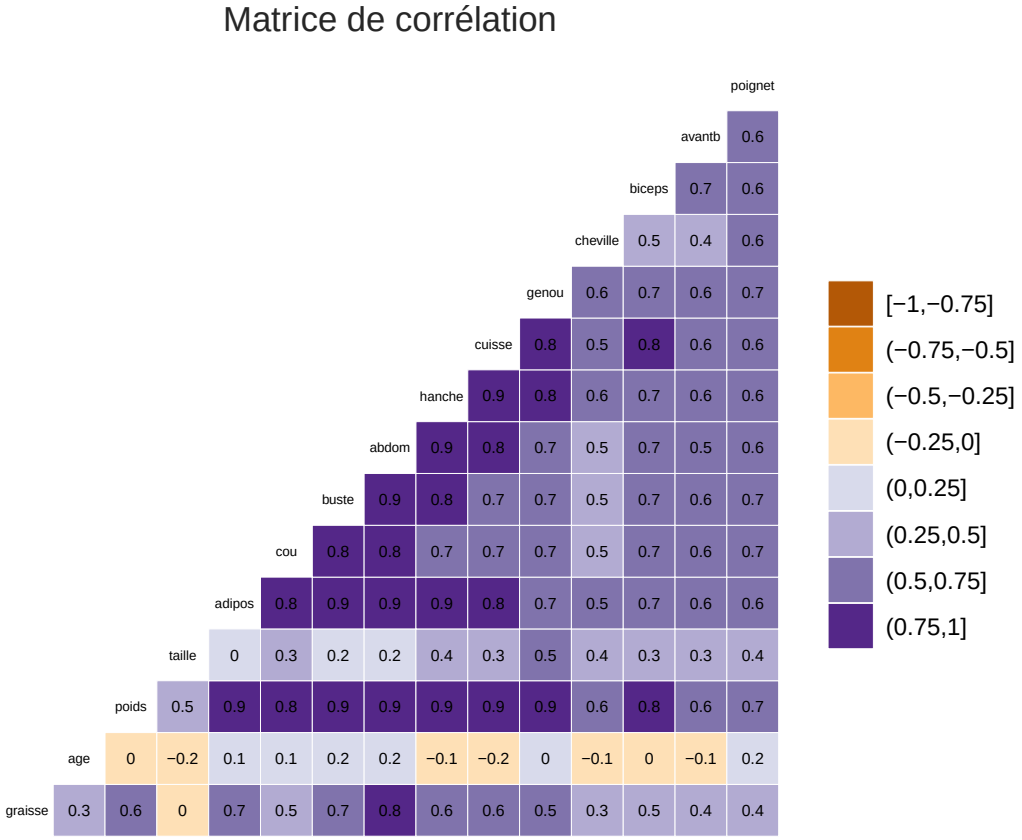
```
ggplot(tgraisse1, aes(x=graisse)) +
geom_histogram(aes(y=..density..), color="black", fill="blue",alpha=.6) +
geom_density(alpha=.4, fill="#FF6666") +
xlab("graisse")
```



On constate que notre variable d'intérêt "graisse" a toujours une distribution proche de la loi normale. Nous gardons nos données de base et nous intéressons maintenant à la corrélation entre nos variables.

Matrice de corrélation

```
library(GGally)
##matrice de correlation
ggcorr(tgraisse[,1:15], nbreaks=8, palette='PuOr', label=TRUE,
label_size=2, size = 1.8, label_color='black') +
ggtitle("Matrice de corrélation") +
theme(plot.title = element_text(hjust = 0.5, color = "grey15"))
```



On constate rapidement à vue d'oeil que nos variables sont bel et bien corrélées.

Comme nous voulons prédire notre variable d'intérêt "graisse" et vu la situation selon laquelle nous faisons face à une corrélation assez importante de variables dûe certainement à la redondance de l'information. Nous nous servirons des méthodes citées au debut de ce rapport pour atteindre nos objectifs. Une réduction en un nombre beaucoup plus pétites de variables est nécessaire vu la corrélation des variables et bien évidemment comme vu en cours, nous ne penserons pas à l'estimateur dans le cas de la régression linéaire car même sans biais, ses propriétés se révèlent défaillantes vu la corrélation importante des variables.

Echantillon d'apprentissage et test

Nous créons un échantillon d'apprentissage contenant la moitié de nos observations pour estimer les modèles et un échantillon test pour le reste de nos observations pour la comparaison des erreurs de prévisions.

```
set.seed(123456)
n <- nrow(tgraisse)
tgraisse_train <- sample(1:n, ceiling(n/2))
tgraisse_test <- tgraisse[-tgraisse_train,]
tgraisse_test_obs=(-tgraisse_train)
y <- tgraisse$graisse
y_test <- tgraisse_test$graisse
```

Méthodes de pénalisation : Ridge

Nous écrivons la regression Ridge sous la forme des carrés des résidus pénalisés sous cette forme : nous minimisons le terme ci dessous:

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{14}\beta_j z_{ij})^2 + \lambda \sum_{j=1}^{14}\beta_j^2$$

.

Nous avons deux termes, le prémier correspond à la RSS et le second terme est la fonction de pénalité.

Nous avons deux termes, le prémier correspond à la RSS et le second terme est la fonction de pénalité.

$\lambda, (\lambda > 0)$ est un paramètre, autrement dit le coefficient de pénalité qui contrôle l'impact de la pénalité.

Faisons d'abord une régréssion linéaire vu que toutes nos variables (explicatives, et à expliquer) sont quantitatives.

```
res <- lm(graisse~., data=tgraisse,subset=tgraisse_train) #modèle linéaire
y_pred_lm <- predict(res,tgraisse_test) #Prédictions
mean((y_pred_lm-y_test)^2) #EQM
```

```
## [1] 21.42071
```

Calculons les coefficients obtenus par les MCO

```
res$coefficients
```

```
##  (Intercept)          age        poids       taille       adipos          cou
## 22.482490338  0.086393575  0.021818347  0.109304252  1.175483861 -0.287530708
##        buste        abdom       hanche       cuisse        genou     cheville
## -0.462964664  0.844253680 -0.425831553  0.008972958 -0.328110111  0.219077009
##       biceps       avantb      poignet
##  0.312622088  0.067780168 -1.754042367
```

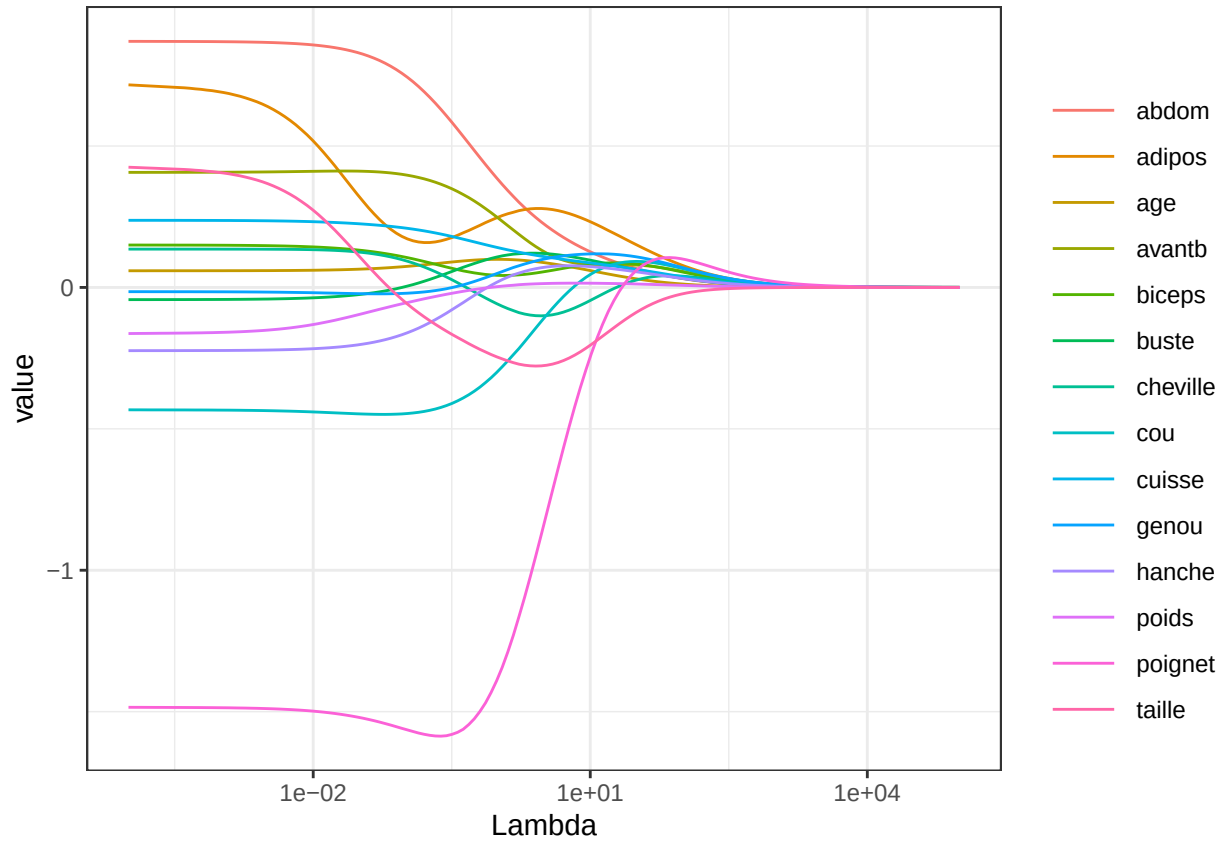
L'erreur obtenue est de 21.4207 dans le cadre de la régression linéaire.

Grille pour 150 valeurs possibles de λ et l'impact des valeurs possibles de ce λ sur les regresseurs

```
x <- model.matrix(tgraisse$graisse~., data=tgraisse)[,-1]
norm_vec <- function(x) sqrt(sum(x^2))
grid <- 10^seq(5,-4,length=150)
ridge.mod <- glmnet(x,y,alpha=0,lambda=grid)
coeff_Ridge <- coef(ridge.mod)
Ridge_data=as.data.frame(as.matrix(coeff_Ridge))
Ridge_data$coef <- row.names(Ridge_data)
Ridge_data <- melt(Ridge_data, id = "coef")
Ridge_data$variable <- as.numeric(gsub("s", "", Ridge_data$variable))
Ridge_data$lambda <- ridge.mod$lambda[Ridge_data$variable+1]
Ridge_data$norm2 <- apply(coeff_Ridge[-1,], 2, norm_vec)[Ridge_data$variable+1]
```

Representations graphiques avec ggplot

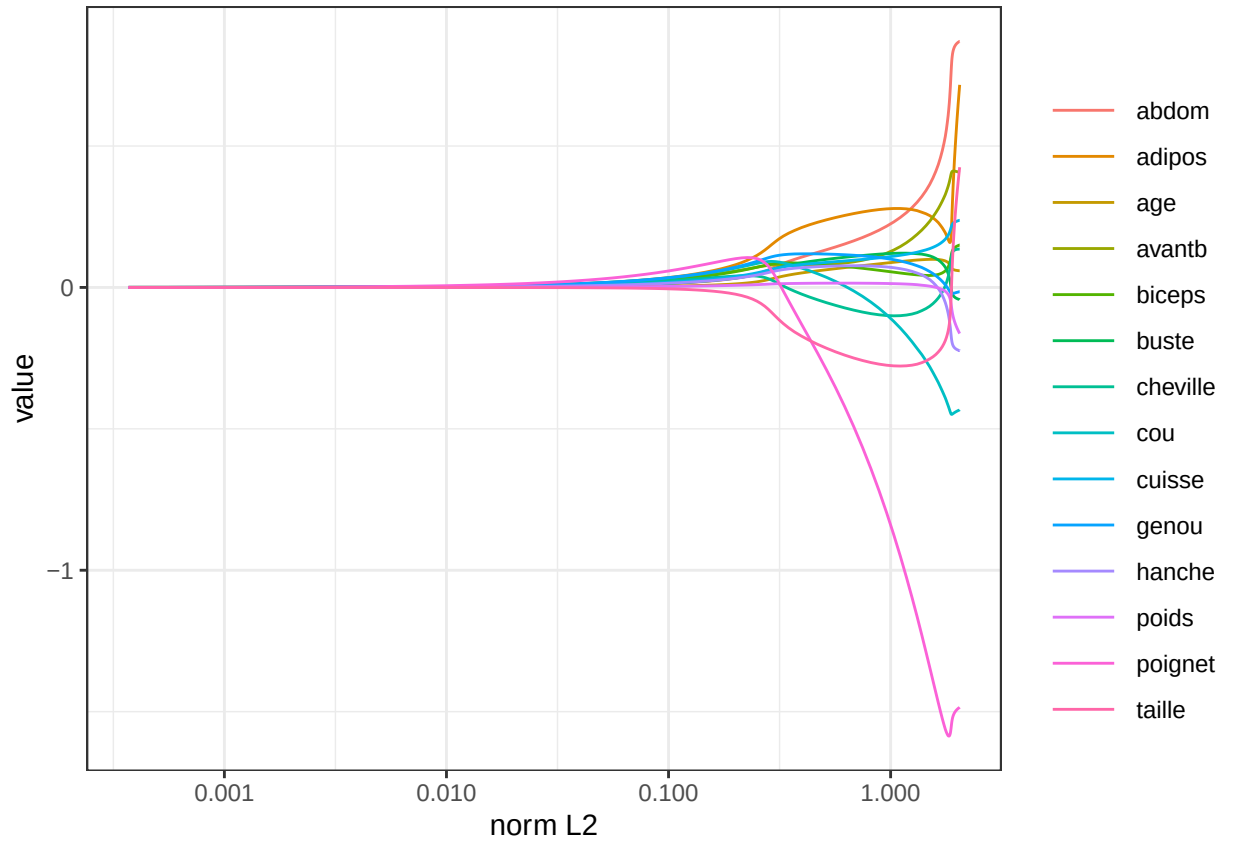
```
ggplot(Ridge_data[Ridge_data$coef != "(Intercept)",], aes(lambda, value, color = coef))+
geom_line()+
scale_x_log10() +
xlab("Lambda") +
guides(color = guide_legend(title = ""),
linetype = guide_legend(title = "")) +
theme_bw() +
theme(legend.key.width = unit(2,"lines"))
```



On obtient des coefficients similaires à ceux fournis par les MCO pour des valeurs de λ proches de 0. En revanche, lorsque λ grandit, on constate que les coefficients tendent vers 0 afin d'équilibrer l'augmentation du coefficient de pénalisation.

Regardons maintenant l'impact des valeurs possibles de ce λ sur les regresseurs en fonction de la norme

```
ggplot(Ridge_data[Ridge_data$coef != "(Intercept)",], aes(norm2, value, color = coef))+
geom_line()+
scale_x_log10() +
xlab("norm L2") +
guides(color = guide_legend(title = ""),
linetype = guide_legend(title = "")) +
theme_bw() +
theme(legend.key.width = unit(2,"lines"))
```



La norme accroît et décroît au même sens que les coefficients. Nous définissons la norme L2 des β_j comme la distance entre les coefficients au point 0. Quand λ augmente, on constate que la norme dimunie plus dans la plus part des cas et quand λ diminue, cette norme est très minime.

Erreur de prediction

Pouvoir predictif d'un modèle Ridge

Nous voulons nous servir de la validation croisée en calculant l'erreur comme précedemment et mesurer l'efficacité du modèle Rigde en terme de prévisions.

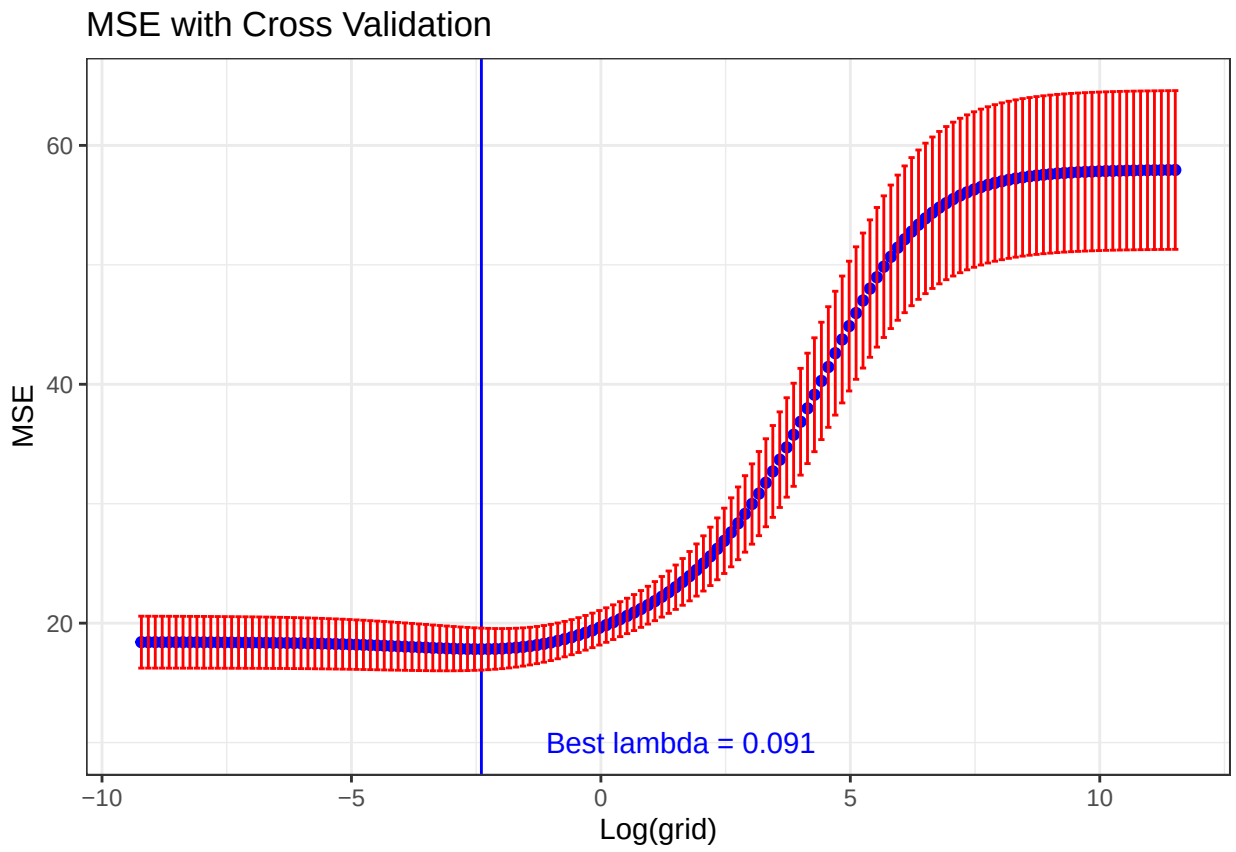
glmnet() est un package R qui peut être utilisé pour ajuster les modèles de Régression Rigde modèle et d'autres. Alpha argument détermine le type de modèle qui est adapté. Quand l'alpha=0, Ridge Modèle est bon.

cv.glmnet() effectue la validation croisée, par défaut de 10 fois ce qui peut être ajustée à l'aide de nfolds.

```
set.seed(123456)
cv.out.ridge <- cv.glmnet(x[tgraisse_train,],y[tgraisse_train],alpha=0,lambda=grid)
best.lambda.ridge <- cv.out.ridge$lambda.min
eq_cross=paste0("Best lambda = ",round(best.lambda.ridge,3))
cv_data=data.frame(log_lambda=log(grid),MSE=cv.out.ridge$cvm,cvup=cv.out.ridge$cvup,
cvlo=cv.out.ridge$cvl)

ggplot(cv_data,aes(log_lambda,MSE)) +
geom_point(col='blue')+
labs(x= "Log(grid)",y='MSE')+
geom_vline(xintercept = log(best.lambda.ridge),col="blue")+
```

```
geom_errorbar(aes(ymin=cvlo,ymax=cvup),col='red')+
theme_bw()+ ggtitle("MSE with Cross Validation")+
annotate("text", x=log(best.lambda.ridge)+4, y=10, label= eq_cross,col='blue')
```




```
min(cv.out.ridge$cvm)
```

```
## [1] 17.8272
```

L'erreur obtenue est de 17.8272 avec le meuilleur λ qui est de 0.091.

Quand Log(λ) tend vers 1, λ tend vers 0 et on constate une augmentation de plus en plus croissante de l'erreur à partir de cette valeur.

Donnons maintenant les coefficients estimés avec le meilleur λ

Nous obtenons une meilleure prédiction de la variable graisse car toutes nos variables explicatives de départ étaient très corrélées à partir de l'estimateur β_{Ridge} ($\hat{\beta}_r$).

```
coef(glmnet(x,y,alpha=0,lambda=best.lambda.ridge))
```

```
## 15 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept) -13.215546998
## age           0.072804756
## poids        -0.060345561
## taille       -0.039596526
## adipos        0.179557998
## cou          -0.446489833
## buste         0.000818533
```

```
## abdom          0.758417784
## hanche        -0.163972663
## cuisse         0.212137417
## genou         -0.019555799
## cheville       0.099836277
## biceps         0.111682689
## avantb         0.400627193
## poignet       -1.551686261
```

Méthodes de pénalisation : Lasso

Cette méthode est très similaire à Ridge.

Nous écrivons la regression Ridge sous la forme de la minimisation des carrés des résidus sous contraite

$$\sum_{j=1}^{14} |\beta_j| \leq \tau$$

: Regardons plus clairement la minimisation du terme ci de sous avec la contrainte :

$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{14} \beta_j z_{ij})^2 \sum_{j=1}^{14} |\beta_j| \leq \tau$

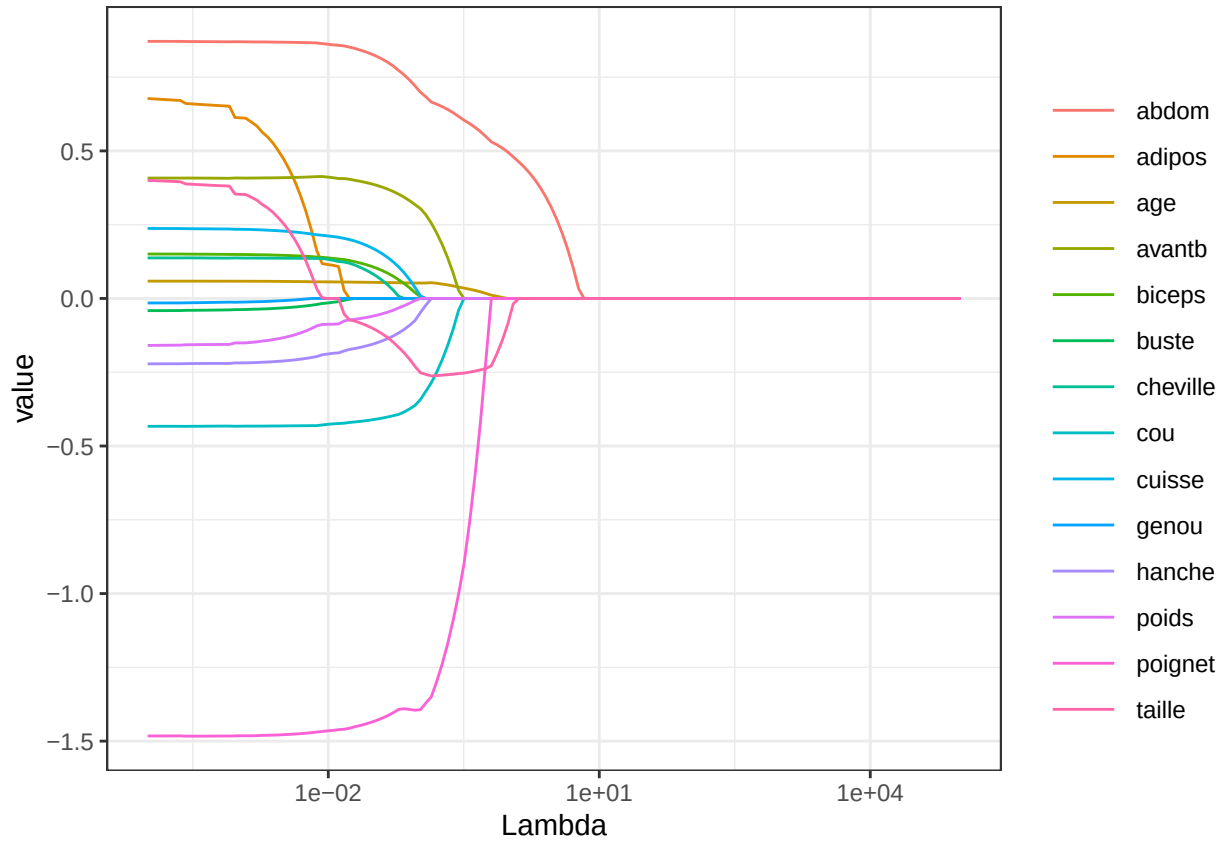
Quand τ est faible, les coefficients sont rétrécis.

Intérêt : LASSO peut faire office de dispositif de sélection de variables en annulant certains coefficients β_j. Alors, les variables associées à $(\beta_j = 0)$ sont exclues du modèle prédictif. C'est l'intérêt de Lasso par rapport à Ridge. Lasso essaie de forcer quelques coefficients à être exactement égal à 0.

Pour cela, nous rajouterons à la fonction glmnet l'argument alpha=1 en utilisant mêmes étapes précédemment.

```
lasso.mod <- glmnet(x,y,alpha=1,lambda=grid)
coeff_lasso <- coef(lasso.mod)
Lasso_data=as.data.frame(as.matrix(coeff_lasso))
Lasso_data$coef <- row.names(Lasso_data)
Lasso_data <- melt(Lasso_data, id = "coef")
Lasso_data$variable <- as.numeric(gsub("s", "", Lasso_data$variable))
Lasso_data$lambda <- lasso.mod$lambda[Lasso_data$variable+1]
Lasso_data$norm <- apply(abs(coeff_lasso[-1,]), 2, sum)[Lasso_data$variable+1]

ggplot(Lasso_data[Lasso_data$coef != "(Intercept)",], aes(lambda, value, color = coef))+
geom_line()+
scale_x_log10() +
xlab("Lambda") +
guides(color = guide_legend(title = ""),
linetype = guide_legend(title = "")) +
theme_bw() +
theme(legend.key.width = unit(2,"lines"))
```



On constate qu'effectivement certains coefficients des variables sont réduits à 0 à partir d'une certaine valeur de λ avec λ élevé.

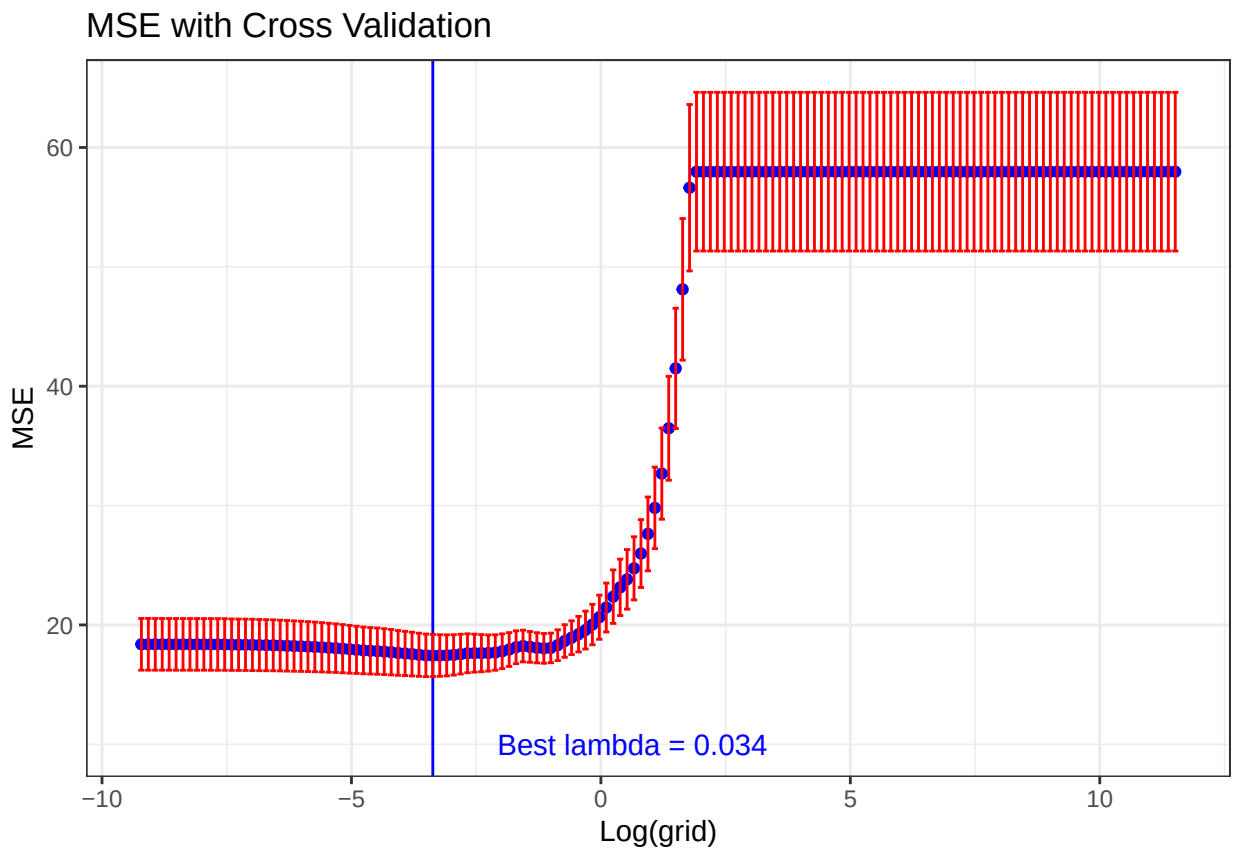
On veut connaitre comme dans Ridge, l'efficacité de lasso en terme de prédiction et pour cela on procède par validation croisée.

```
set.seed(123456)
 cv.out.lasso <- cv.glmnet(x[tgraisse_train,],y[tgraisse_train],alpha=1,lambda=grid)
best.lambda.lasso <- cv.out.lasso$lambda.min
eq_cross=paste0("Best lambda = ",round(best.lambda.lasso,3))



cv_data=data.frame(log_lambda=log(grid),MSE=cv.out.lasso$cvm,cvup=cv.out.lasso$cvup,
cvlo=cv.out.lasso$cvl)

ggplot(cv_data,aes(log_lambda,MSE)) +
geom_point(col='blue')+
labs(x= "Log(grid)",y='MSE')+
geom_vline(xintercept = log(best.lambda.lasso),col="blue")+
geom_errorbar(aes(ymin=cvlo,ymax=cvup),col='red')+
theme_bw()+ ggtitle("MSE with Cross Validation")+
annotate("text", x=log(best.lambda.lasso)+4, y=10, label= eq_cross,col='blue')
```




```
min(cv.out.lasso$cvm)
```

```
## [1] 17.43521
```

Avec un lambda de 0.034 on obtient une erreur de seulement 17.4352 qui est inférieur, disons un peu, légèrement à celle obtenue par Ridge qui est de 17.8272 . On s'intéresse maintenant Ci-dessous aux coefficients correspondant à ce λ.

```
coef(glmnet(x,y,alpha=1,lambda=best.lambda.lasso))
```

```
## 15 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) -10.46043740
## age           0.05442392
## poids        -0.05627710
## taille       -0.10915841
## adipos        .
## cou          -0.40950451
## buste         .
## abdom         0.82021051
## hanche       -0.15322169
## cuisse        0.16376440
## genou         .
## cheville      0.07413114
## biceps        0.10256386
## avantb        0.38235312
## poignet      -1.43026225
```

La réduction de la dimension (régression sur composantes principales et PLS)

Régression sur composantes principales

```
library(pls)#chargement de la librairie
```

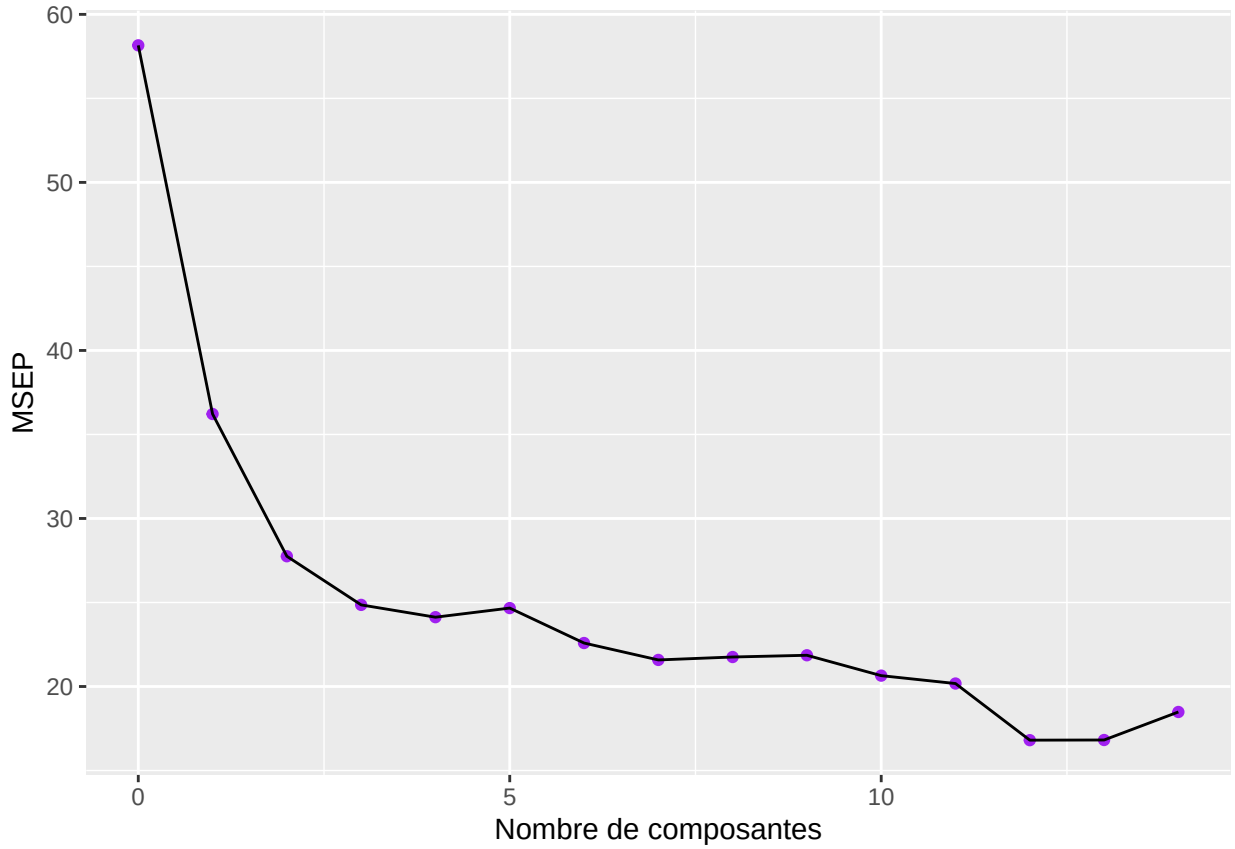
Nous allons traiter deux méthodes qui aident dans la réduction de la dimension. L'idée ici est toute simple : nous cherchons à projeter nos 14 prédicteurs dans un espace de dimension r beaucoup plus petite pour enfin prédire Y par un modèle linéaire utilisant les r projections : Comme son nom l'indique nous voulons réduire le nombre de variables explicatives à l'aide d'une analyse en composante principale (ACP) qui permettra de conserver le maximum d'inertie. La fonction pcr du package **pls** nous sera utile pour notre analyse.

```
set.seed(123456)
pcr.train=pcr(graisse~., data=tgraisse, subset=tgraisse_train, scale=TRUE,validation="CV")
mse.pcr <- MSEP(pcr.train, estimate="CV")
mse.pcr
```

```
## (Intercept)      1 comps      2 comps      3 comps      4 comps      5 comps
##       58.16        36.22        27.75        24.86        24.13        24.67
##     6 comps      7 comps      8 comps      9 comps     10 comps     11 comps
##       22.59        21.58        21.75        21.86        20.65        20.18
##    12 comps     13 comps     14 comps
##       16.81        16.82        18.48
```

```
erreurs_cv_pcr <- cbind(cv=mse.pcr[["val"]], nb=mse.pcr[["comps"]])
erreurss_cv_pcr <- as.data.frame(erreurs_cv_pcr)
```

```
df<-ggplot(erreurss_cv_pcr,aes(x=nb,y=cv))+geom_point(color="purple")+
  geom_line(color="black")+labs(x= "Nombre de composantes",y='MSEP')
df
```



On observe une décroissance marquée pour l'erreur en validation croisée (MSEP) en fonction du nombre de composantes . Le nombre optimal de composantes correspond à la valeur pour laquelle MSEP est minimum :

```
ncomp.pcr <- which.min(mse.pcr$val["CV",,])-1
ncomp.pcr
```

```
## 12 comps
##       12
```

Donc la plus petite erreur quadratique moyenne est atteinte pour p = 12 composantes. et on ajuste les données avec un modèle linéaire avec $p = 12$ composantes principales:

```
new.pcr<-plsr(graisse~., data=tgraisse, subset=tgraisse_train, scale=TRUE, ncomp=12)
summary(new.pcr)
```

```
## Data:    X dimension: 126 14
##  Y dimension: 126 1
## Fit method: kernelpls
## Number of components considered: 12
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X          60.62     71.7    77.24    81.69    85.48    88.45    91.27    93.90
## graisse    45.79     64.9    69.48    72.22    73.99    75.33    75.67    75.85
##          9 comps  10 comps  11 comps  12 comps
## X          95.48     96.54     97.77     98.55
## graisse    75.90     75.95     75.97     75.97
```

On voit par exemple que pour la 12ème composante,on a un taux de variance expliquée qui vaut 98.55% (contribution de ces 12 composantes)

Prévision

On calcule la valeur de notre erreur EQM pour les données précédantes.

```
pcr.pred12 <- predict(new.pcr, newdata=tgraisse_test[,-1], ncomp=12)
EQM12<- mean((pcr.pred12-tgraisse_test$graisse)^2)
EQM12
```

```
## [1] 21.5624
```

On obtient une erreure de 21.56.

Régression Partial Least Square (PLS)

La régression PLS doit être utilisée lorsque la régression linéaire multiple ne peut pas s'appliquer. En particulier en cas de forte multicolinéarité ou lorsqu'on a plus de variables que d'individus.

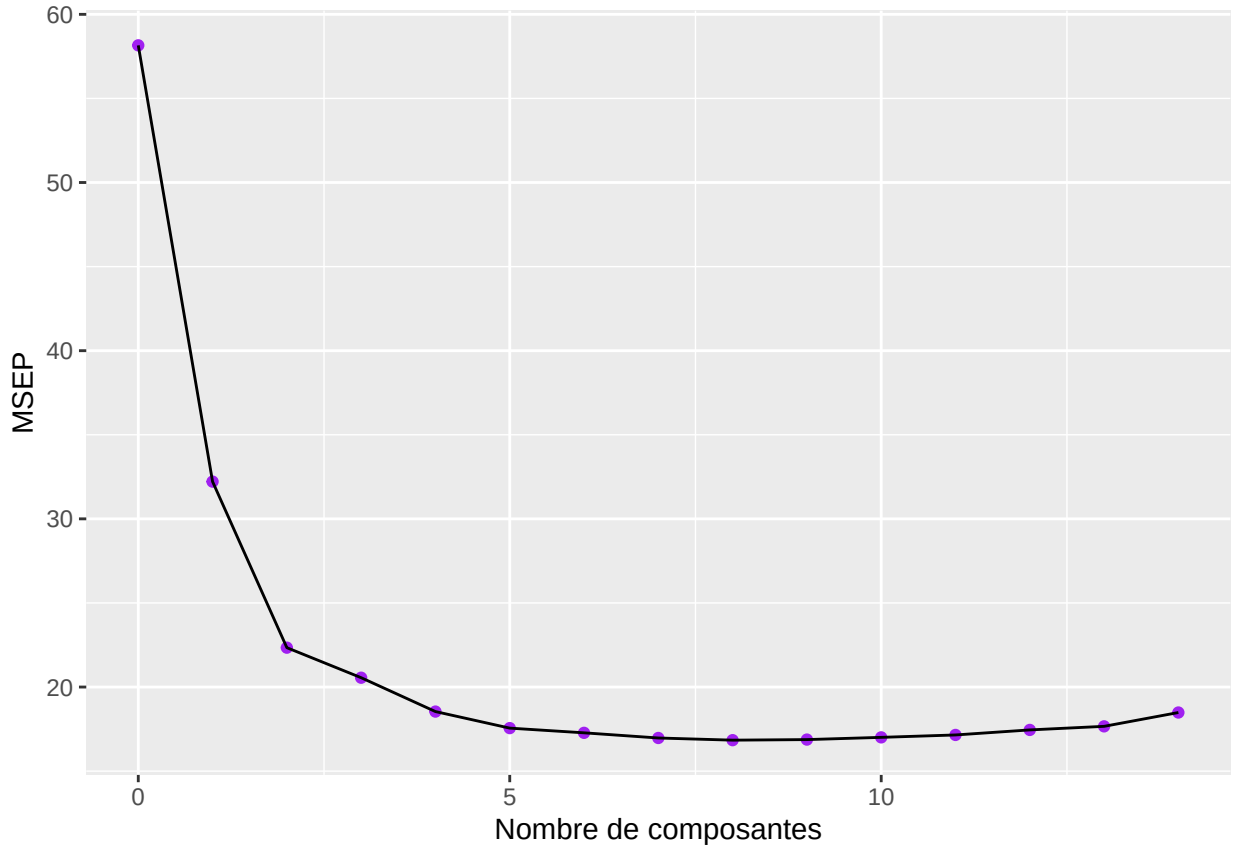
La régression Partial Least Square (PLS) est simple par son principe et sa mise en œuvre. L'objectif est de résumer l'ensemble des variables $X^1 ... X^p$ par un sous-ensemble de variables $Z^1 ... Z^r$ deux à deux orthogonales et combinaisons linéaires des variables $X^1 ... X^p$.

```
set.seed(123456)
plsr.train <- plsr(graisse~., data=tgraisse, subset=tgraisse_train, scale=TRUE,validation="CV")
msep.pls <- MSEP(plsr.train, estimate="CV")
msep.pls
```

```
## (Intercept)      1 comps      2 comps      3 comps      4 comps      5 comps
##       58.16        32.22        22.34        20.55        18.54        17.56
##     6 comps      7 comps      8 comps      9 comps     10 comps     11 comps
##       17.28        16.97        16.84        16.87        17.01        17.15
##    12 comps     13 comps     14 comps
##       17.45        17.66        18.48
```

```
error_pls <- cbind(cv=msep.pls[["val"]], nb=msep.pls[["comps"]])
error_pls <- as.data.frame(error_pls)
```

```
df<-ggplot(error_pls,aes(x=nb,y=cv))+geom_point(color="purple")+
  geom_line(color="black")+labs(x= "Nombre de composantes",y='MSEP')
df
```




```
#library(plotly)
#ggplotly(df)
```

```
ncomp.pls <- which.min(msep.pls$val["CV",,])-1
ncomp.pls
```

```
## 8 comps
##       8
```

Donc la plus petite erreur quadratique moyenne est atteinte pour p = 8 composantes. et on ajuste les données avec un modèle linéaire avec $p = 8$ composantes principales:

```
plsr8<- plsr(graisse~., data=tgraisse, subset=tgraisse_train, scale=TRUE, ncomp=8)
summary(plsr8)
```

```
## Data:    X dimension: 126 14
##  Y dimension: 126 1
## Fit method: kernelpls
## Number of components considered: 8
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X          60.62     71.7    77.24    81.69    85.48    88.45    91.27    93.90
## graisse    45.79     64.9    69.48    72.22    73.99    75.33    75.67    75.85
```

Prévision

```
pred8 <- predict(plsr8, newdata=tgraisse_test[,-1], ncomp=8)
EQM.pls8 <- mean((pred8-tgraisse_test$graisse)^2)
```

```
EQM.pls8
```

```
## [1] 20.56963
```

On obtient une erreur de 20.57, légèrement inférieure à celle obtenue avec la méthode PCR.

Conclusion

Le projet est composé de 2 parties:

- D'une part,nous avons comparé les 2 méthodes de régressions pénalisées Ridge et Lasso.Effectivement ces 2 méthodes sont très proches l'une de l'autre.On peut voir que l'EQM de Ridge est très légèrement supérieur à l'EQM de Lasso.
- D'autre part,nous avons réalisé les méthodes de PCR et PLS.On observe que la méthode de PLS performe mieux que la méthode PCR à cause de sa faible valeur de l'EQM.

Méthode	EQM
Ridge	17.83
Lasso	17.45
PCR	21.56
PLS	20.57