Rapport du projet de SIM202

Song Xuanye, Koen Aristote

3/29/2020

Introduction

Dans le cadre de ce projet, nous nous intéressons à la prédiction de la consommation électrique des appareils électroménagers d'une maison à basse consommation dans le cadre d'une compétition Kaggle. Le problème à résoudre est un problème régression.

Afin de pouvoir prédire cette consommation nous travaillons sur un jeu de données contenant des variables quantitatives comme l'humidité, la température, des données météorologiques et des variables qualitatives représentant des facteurs temporels.

Analyse descriptive des données

Jeu de données

On assigne le jeu d'entrainement à une variable **train** et le jeu permettant de tester la prediction à une variable **test** dans le code suivant

```
test = read.csv(file = "Data/test.csv",header=T, sep=",", dec =".")
train = read.csv(file = "Data/train.csv",header = T, sep=",", dec =".")
```

Nous avons commence par un résumé du jeu de données et un **str** du jeux de données afin d'observer les types des variables auxquelles nous avons à faire. Nous n'afficherons pas les résultats qui sont trop longs (43 variables). Une liste exhaustive des variables inclues dans le jeu de données est disponible sur Kaggle/Building-Appliances avec une courte description.

```
str(train)
dim(train)
summary(train)

str(test)
dim(test)
summary(test)
```

- Le jeu de données d'entrainement contient 43 variable et 13964 observations
- Le jeu de données test contient lui aussi 43 variables et 5771 observations

Le jeu de données test ne contient pas la variable **Appliances** qui est notre 'target', c'est à dire la variable dont on veut prédire les valeurs. A la place il y a une variable **ID** correspondant à l'indice de l'observation.

La fonction **str** nous a permis de déterminer de quels types sont chaques variables. La majorité des variables est de type integer ou numérique sauf **WeekStatus**, **Daytype**, **InstantF**,**Day_of_week** et **Date** qui sont de type factor c'et à dire qualitatives. Nous pouvons déjà exclure **InstantF** de nos modèles car elle

est équivalente à la variable **Instant** qui elle est sous forme numérique ce qui sera beaucoup plus facile à manipuler. Nous verrons par la suite comment inclure ces variables qualitatives dans les calculs.

Afin de pouvoir visualiser notre target sous forme de série chronologique on convertit la variable **date** de nos jeux de données en format **POSIXct**. De plus, on observe qu'il y a des données manquantes dans notre jeu de données **Train** et **Test** pour les variables **RH_6** et **Visibility** à travers le summary

```
#conversion des dates
test$date = as.POSIXlt(strptime(test$date,"%Y-%m-%d %H:%M:%S",tz = 'GMT'))
train$date = as.POSIXlt(strptime(train$date,"%Y-%m-%d %H:%M:%S",tz ='GMT'))

cbind(summary(test[,c("RH_6","Visibility")]),summary(train[,c("RH_6","Visibility")]))
```

```
##       RH_6           Visibility          RH_6           Visibility
##  "Min.   :10.07  " "Min.   : 1.0  " "Min.   :10.00  " "Min.   :10.00  "
##  "1st Qu.:37.32  " "1st Qu.:29.0  " "1st Qu.:43.30  " "1st Qu.:29.00  "
##  "Median :55.39  " "Median :40.0  " "Median :64.26  " "Median :40.00  "
##  "Mean   :58.78  " "Mean   :38.3  " "Mean   :63.12  " "Mean   :38.56  "
##  "3rd Qu.:83.10  " "3rd Qu.:40.0  " "3rd Qu.:86.82  " "3rd Qu.:40.00  "
##  "Max.   :99.90  " "Max.   :66.0  " "Max.   :99.90  " "Max.   :66.00  "
##  "NA's   :887  "   "NA's   :7  "    "NA's   :1541  "  "NA's   :84  "
```

Traitement des données manquantes

Afin de pouvoir mener à bien l'analyse descriptive de nos données nous nous interessons avant à traiter les données manquantes.

Nous avons utilisé deux approches:

- Suppression des lignes correspondant aux données manquantes
  L'avantage de cette méthode est sa facilité d'implémentation mais elle n'est pas intéressante car on supprime trop de données des autres variables explicatives. Et nous avons besoin de ces données pour effectuer la prédiction dans le jeu de données test par exemple.
- Foret aléatoire Prediction basée sur le jeu données sans les lignes correspondant aux données manquantes. Nous avons utilisé cette méthode car elle nous permet de conserver toutes les observations des jeux de données. Voici une implémentation pour **RH_6** dans **train**.

```
#completion des données manquantes pour RH6

#données sans les lignes correspondant aux données manquantes
data_wthna_RH6 = train[-which(is.na(train$RH_6)),]

#forêt aléatoire appliquée à RH_6 en fonction de variables explicatives fortement corelées
RF_NA_RH6 = randomForest(formula =  RH_6~RH_5 + RH_2 + RH_9 + RH_7 + RH_8
+ RH_1 + RH_3+RH_4+ RH_out +BE_load_actual_entsoe_transparency+BE_load_forecast_entsoe_transparency+BE_
#prediction de Visibility à partir des variables explicatives pour les données manquantes
predic = predict(RF_NA_RH6,newdata=train[which(is.na(train$RH_6)),])
train[which(is.na(train$RH_6)),'RH_6']= predic
```

Nous nous sommes donc basés sur le jeu de données complété par un modèle forêt aléatoire pour la suite. On peut voir que il n'y a plus de données manquantes

```
which(is.na(train))
```

```
## integer(0)
```

```r
which(is.na(test))
```

```
## integer(0)
```

Traitement des données factorielles

Nous avons observé la présence de 4 variables factorielles sans compter les dates. Ces variables vont poser problème lors des modélisations ainsi nous avons transformé ces variables sous forme d'indicatrices.

On a aussi remarqué en cours de route que les variables **Daytype** et **Day_of week** donnaient la même information et on a donc décidé de ne pas considérer Daytype dans les modèles. De même les variables **Instant** et **InstantF** nous donnent la même information sur l'heure de la journée sous forme numérique et string respectivement, on a donc décidé de ne seulement considérer **Instant** dans nos modèles.

Finalement il y a des variables telles que **Heure** et **Posan** qui ne nous intéressent pas car l'information qu'elles donnent est encore fortement reliée à **Instant** et à **Month**. En effet **Posan** est un indicateur entre 0 et 1 permettant de déterminer à quel moment de lannée l'observation a été effectuée et **Heure** nous donne l'heure, qui est contenue dans la variable **Instant**. Ainsi nous ne les considererons pas. Enfin, **NSM** ne sera pas considéree non plus car elle nous donne de nouveau une information temporelle dans la journée que nous avons à travers Instant, nous ne la considérerons donc pas dans la plupart des modèles.

Voici un exemple de comment nous avons transformé la variable **Day_of_Week** en somme d'indicatrices pondérées.

```r
#Train
DayofWeek = as.vector(train$Day_of_week)
DayofWeek[DayofWeek==c('Monday')]=1
DayofWeek[DayofWeek==c('Tuesday')]=2
DayofWeek[DayofWeek==c('Wednesday')]=3
DayofWeek[DayofWeek==c('Thursday')]=4
DayofWeek[DayofWeek==c('Friday')]=5
DayofWeek[DayofWeek==c('Saturday')]=6
DayofWeek[DayofWeek==c('Sunday')]=7
```

Nous avons eu cette idée après avoir implémenté plusieurs modèles sans transformer les variables, ainsi nous ferons une comparaison des résultats entre le jeu de données sans les transformations de variables et avec ces transformations pour le stepAIC afin de comparer les resultats.

Ci-dessous le jeu données avec les transformations et la suppression des variables obsolètes.

```r
## Jeu de données sans les facteurs et sans les doublons
train_reworked = train[-c(33,34)]
train_reworked$Day_of_week = DayofWeek
train_reworked$WeekStatus = WeekStatus
train_reworked$Day_of_week = as.numeric(train_reworked$Day_of_week)
train_reworked$WeekStatus = as.numeric(train_reworked$WeekStatus)

test_reworked = test[-c(32,33)]
test_reworked$Day_of_week = DayofWeek_test
test_reworked$WeekStatus = WeekStatus_test
test_reworked$Day_of_week = as.numeric(test_reworked$Day_of_week)
test_reworked$WeekStatus = as.numeric(test_reworked$WeekStatus)
```

On observe bien la transformation de notre variable *Day_of_week* par exemple.

```r
table(train_reworked$Day_of_week)
```

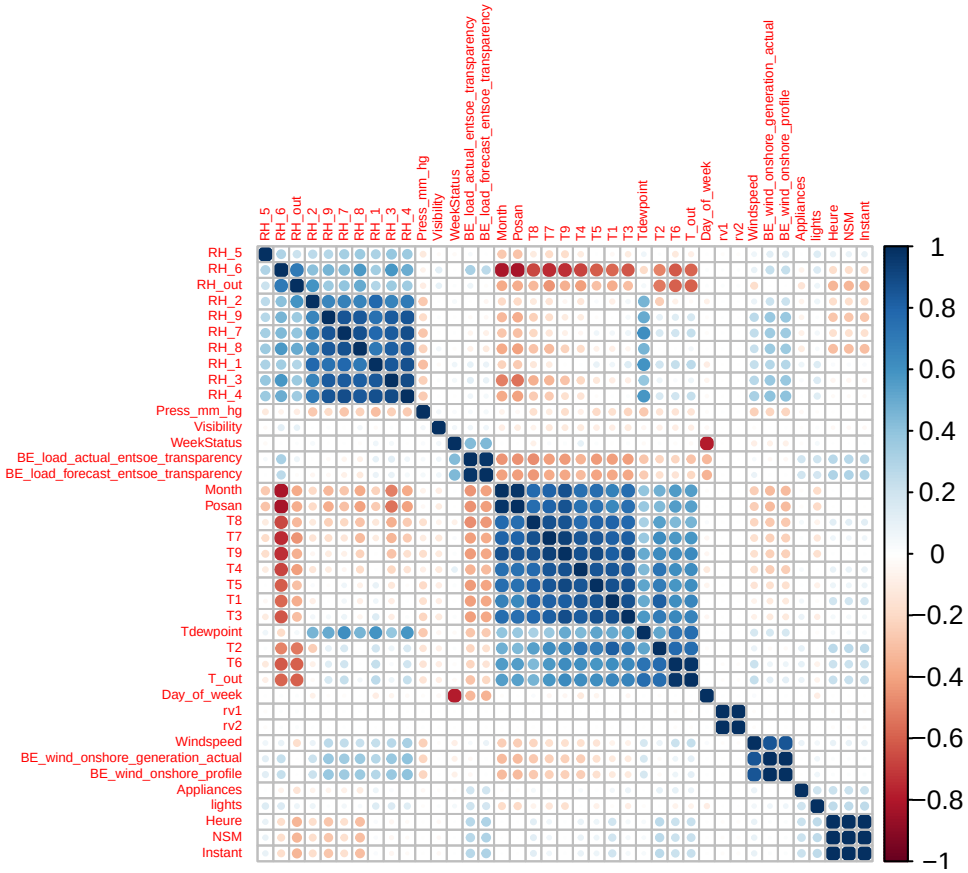
```
##
##    1    2    3    4    5    6    7
## 1993 2061 2059 2023 1957 1918 1953
```

Etude des correlations et ACP

Corrélations

Afin de pouvoir mieux comprendre le comportement des variables entre elles et leur significativité dans le modèle que nous construirons nous avons effectué une analyse des corrélations. Ci dessous voici un plot des correlations entre les variables. Comme la variable **BE_Wind_Onshore_Capacity** est constante et donc de variance nulle, nous ne l'avons pas inclue pour le calcul des corrélations. Nous avons considéré NSM et Posan pour quand même avoir des informations dessus.

```
library(corrplot)
corrplot(cor(train_reworked[,-c(1,39)]),tl.cex=0.4,order="hclust")
```



On observe des clusters de variables fortement corellées entre elles:

- Les variables associées aux températures
- Celles associées à l'humidité dans les pièces et à l'exterieur
- Les variables associées à des indicateurs temporels tel que l'heure, le mois, l'instant, **NSM**, en effet elles transmettent des informations liées entre elles.

Ce qui est cohérent, si il fait chaud à l'exterieur, les pièces de la maison seront plus chaudes, de même pour l'humidité à quelques différences près entre les pièces.

Si on considère la variable **Appliances** on peut voir que de nombreuses variables ne semblent pas corélées à celle-ci. Les variables étant le plus corélées à **Appliances** sont: **RH_2**,**RH_9**,**RH_7**,**RH_8**,**RH_1**,**RH_6**,**RH_out**,**BE_**
T1,**T3**,**T2**,**T6**,**Tout**,**Windspeed**,**lights**,**instant**,**NSM**,**Heure**

Comme **Instant**,**NSM**,**Heure** ont une corrélation de 1 nous ne considererons qu'une seule des variables, Instant.

On observe d'autres relations entre les variables moins évidentes par rapport à la nature des variables.

Par exemple, il semble y avoir une corrélation entre la température, l'humidité et la génération d'électricité par éolienne en Belgique. **BE_wind_onshore_generation_actual**, **BE_wind_onshore_generation_forecast**

De même on observe des corrélations entre la consommation d'électricité et la température et l'humidité.

La vitesse du vent est elle aussi corrélée à la température et l'humidité et les températures et l'humidité sont également corélées.

La consommation d'électricité, les températures, l'humidité sont aussi corélées aux variables temporelles. On verra qu'une analyse en composante principale peut aider à mieux visualiser ces constats.

Enfin, on peut voir que les variables que nous avons décidé de ne pas considérer: **NSM, Instant, Heure** sont positivement corélées et égales à 1, de même pour les valeurs prédites et actuelles des consommations et de la production d'énergie en Belgique.

L'analyse des corrélations nous permet donc d'en dire beaucoup sur les données que nous étudions.

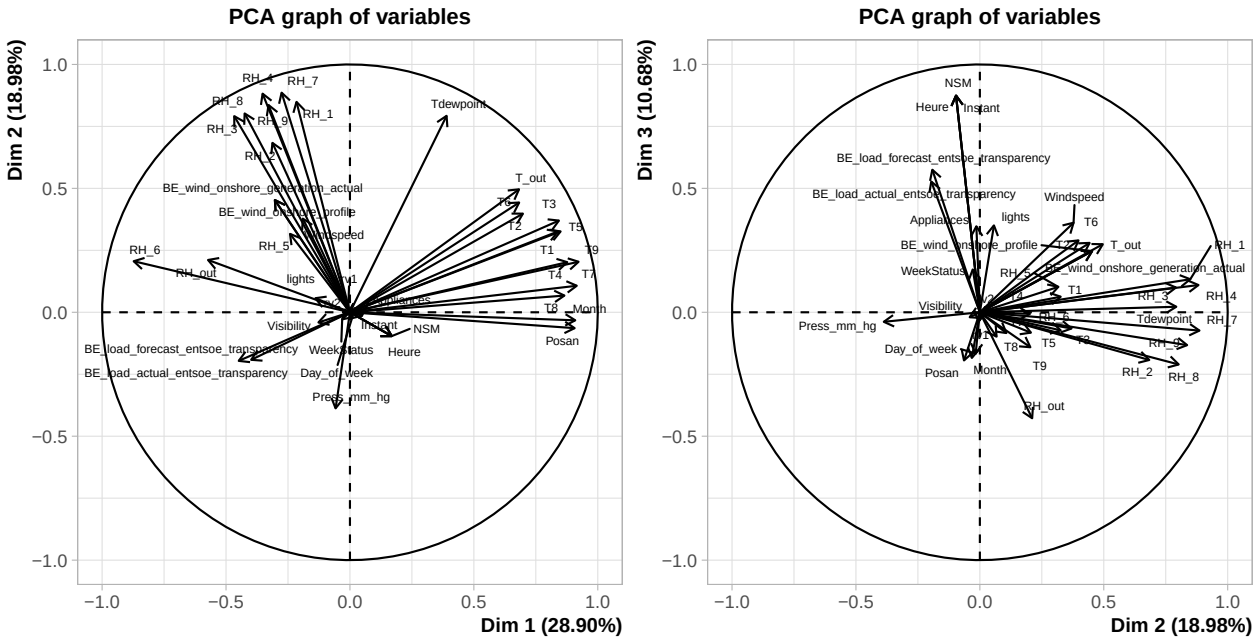
ACP

Afin d'observer des caractéristiques sur nos variables et pousser l'analyse descriptive, nous avons procédé à une analyse en composantes principales.

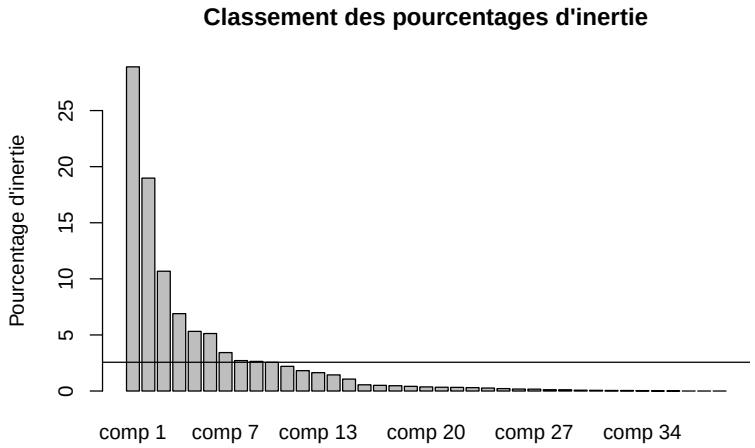
Ayant un jeu de données seulement composé de valeurs numériques nous pouvons donc procéder à une ACP, il suffit de centrer et réduire nos variables car elles ne sont pas homogènes entre elles. Le code ci dessous utilise le package e1071. La fonction PCA nous permet d'effectuer l'ACP.

```
#jeu de données centré réduit
N = length(train$Appliances)
PCA_train = scale(train_reworked[,-c(1,39)])*((N-1)/N)
PCA_train_analyse= PCA(train_reworked[,-c(1,39)],graph = FALSE)
```

```
par(mfrow=c(1,2))
plot.PCA(PCA_train_analyse,choix="var",axes = c(1,2),cex=0.5)
plot.PCA(PCA_train_analyse,choix="var",axes = c(2,3),cex=0.5)
```




```
barplot(PCA_train_analyse$eig[,2],ylab = "Pourcentage d'inertie",main = "Classement des pourcentages d'i
coude = 100/length(names(train_reworked[,-c(1,39)])) #règle du coude
abline(h=coude)
```




```
PCA_train_analyse$eig[1:15,3]
```

```
##   comp 1   comp 2   comp 3   comp 4   comp 5   comp 6   comp 7   comp 8
## 28.89944 47.88010 58.55858 65.45641 70.77394 75.89859 79.32476 82.03784
##   comp 9  comp 10  comp 11  comp 12  comp 13  comp 14  comp 15
## 84.67745 87.23645 89.43627 91.25093 92.88780 94.32483 95.38851
```

L'analyse en composantes principales, consiste à diagonaliser la matrice de correlation associée au jeu de données. En se placant sur la base formée par les vecteurs propres de cette matrice on peut déterminer quelles sont les dimensions qui contribuent le plus à expliquer la variabilité des données. Sur le barplot on à tracé les valeurs propres par ordre décroissant et on affiche le pourcentage d'inertie c'est à dire : $I = \frac{\lambda}{\sum_{i=1}^{N} \lambda_i}$ qui correspond au pourcentage de variabilitié des données. Ainsi on observe que à partir de la 11ème dimension on a expliqué environ 90% de la variabilité des données ce qui peut être intéressant car on peut réduire la dimension de notre jeu de données.

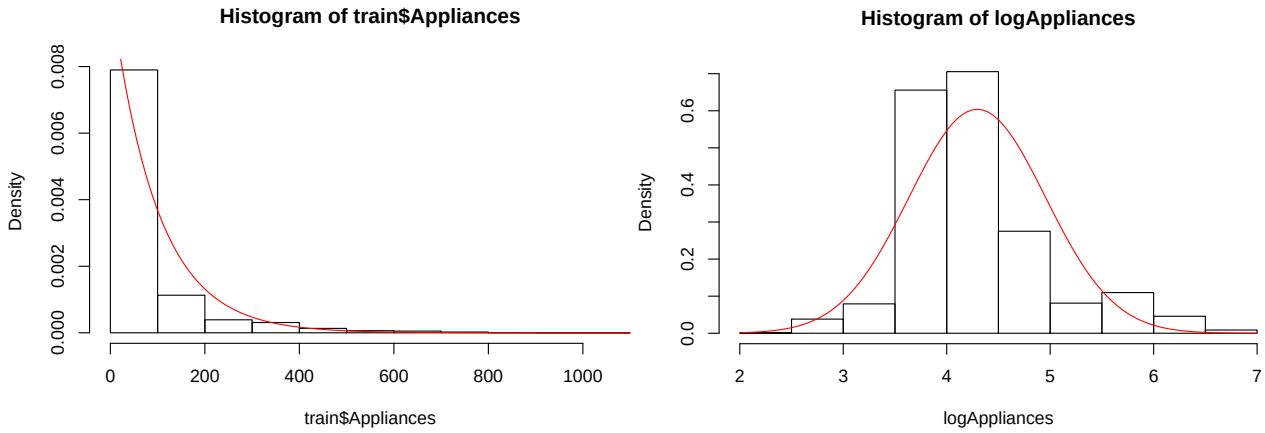
De plus, sur le cercle de corrélation on peut observer les variables qui sont le mieux representées dans le jeu

de données train, ce sont les fleches qui sont le plus proche du cercle unité, nous ne tracerons ces cercles que pour 3 axes car le nombre de combinaisons est trop grand mais on peut déja observer sur ces deux cercles que nous pouvons séparer les valeurs de **Appliances** en fonction de la valeur de ses prédicteurs. Sur le cercle 1 on voit que **Posan**,**Month** et certaines températures sont fortement corélées à la dimension 1 tandis que les humidités elles, sont corrélées à la dimension 2. Ceci signifique que l'on pourra déja séparer les valeurs de **Appliance** en fonction des valeurs de ces variables. Si ce sont des valeurs vers la fin de l'année et que les températures et les humidités sont élevées, ces observations seront sur le quadrant superieur droit. Si elles correspondent à la fin de l'année avec des humidités faibles alors elles seront placées sur le 2nd cadrant. On voit bien que l'on peut donc déja extraire des information sur Appliance en fonction de ses prédicteurs. De même sur le deuxième cercle, l'humidité correspond à la dimension 2 ce qui est cohérent avec notre analyse du premier cercle et la dimension 3 correspond aux facteurs temporels liés à l'heure.

Etude graphique des variables et de leurs lois

Considérons notre target **Appliances** :

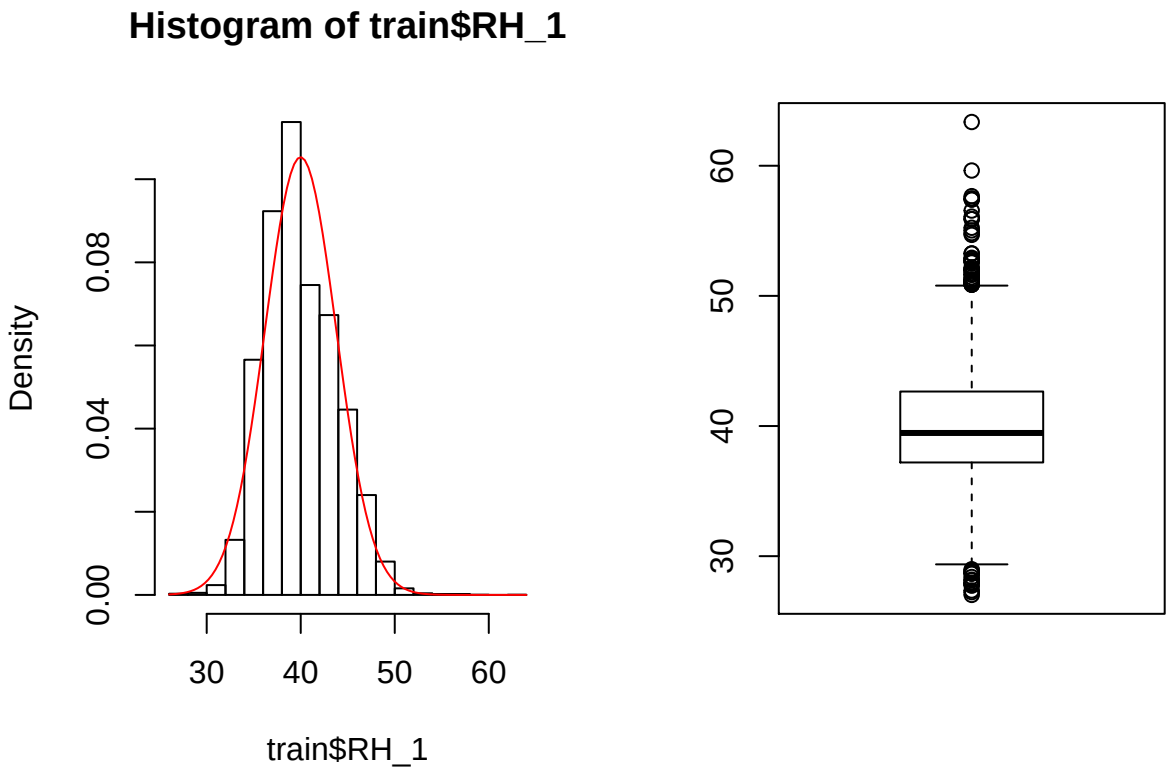
```
hist(train$Appliances,proba=T)
curve(dexp(x,1/mean(train$Appliances)),add=T,col='red')
logAppliances=log(train$Appliances)
hist(logAppliances,proba=T)
curve(dnorm(x,mean(logAppliances),sd(logAppliances)),add=T,col='red')
```



En tracant l'histogramme de notre target, on peut penser que cette variable suit une loi exponentielle. Nous avons callé la densité d'une loi exponentielle de paramètre $\frac{1}{\overline{X}}$ (l'estimateur du maximum de vraisemblance) et on observe que elle se calle bien à l'histogramme. Et en passant au log on semble obtenir une loi normale de paramètre les estimateurs du max de vraisemblance sur la 3eme figure ci-dessus.Ainsi on peut penser que la variable suit une loi lognormale

Considérons **RH_1**

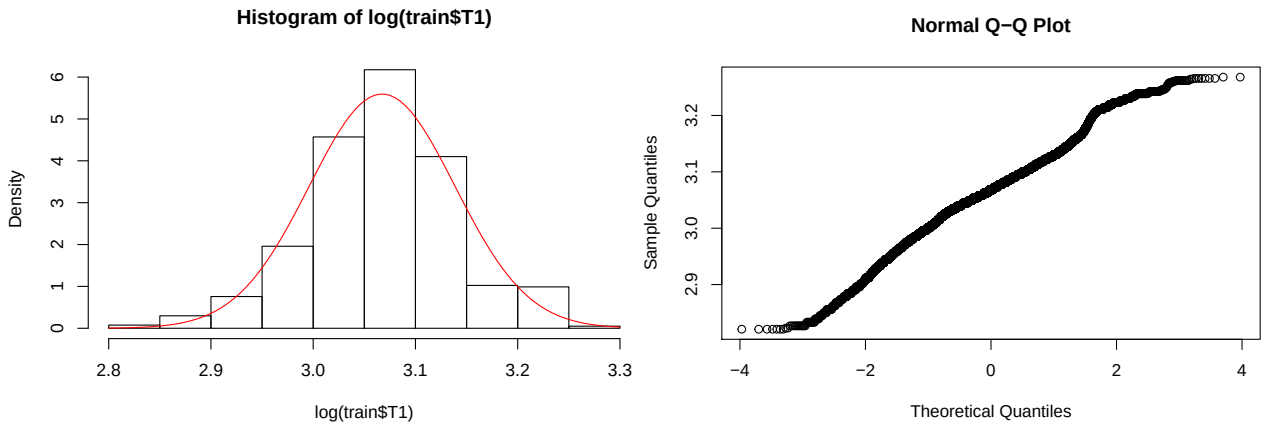
```
par(mfrow=c(1,2))
hist(train$RH_1,proba=T)
curve(dnorm(x,mean(train$RH_1),sd(train$RH_1)),add=T,col='red')
boxplot(train$RH_1)
```



On peut supposer que RH_1 suit une loi normale, car l estimateur de max de vraisemblance des paramètres nous donne une densité qui est bien calée par rapport à l'histogramme. En utilisant les méthodes similaires, on observe que les variables **RH_i** sauf **RH_6** suivent une loi normale.

On fait la même chose pour les variables de température.

```
hist(log(train$T1),proba=T)
curve(dnorm(x,mean(log(train$T1)),sd(log(train$T1))),add=T,col='red')
qqnorm(log(train$T1))
```

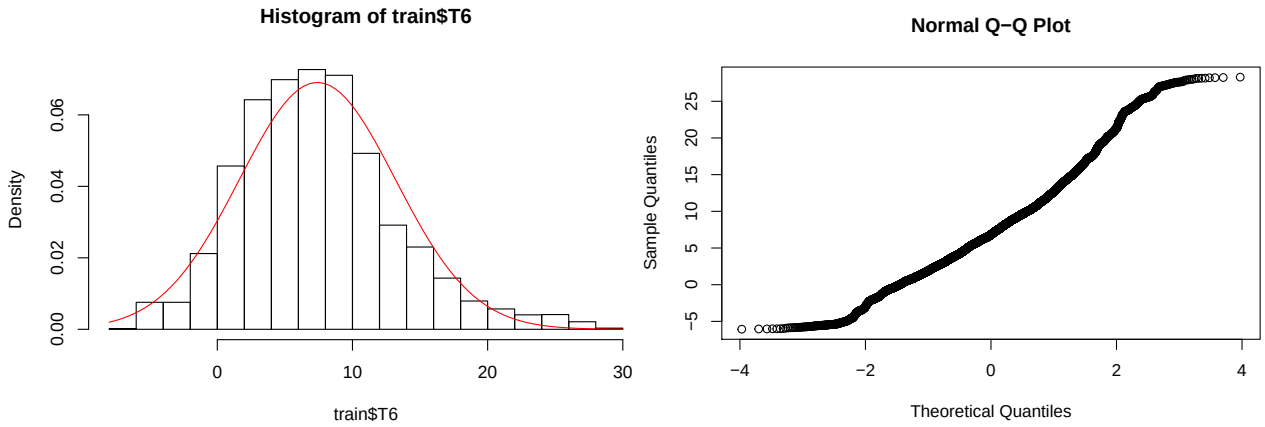


On peut observer que **T1** suit de même une loi lognormale car en passant au log on observe une densité de type gaussienne. On peut aussi vérifier avec un qqplot

On a observé que les variables associées aux températures suivaient ces mêmes lois sauf T6 qui semble suivre une loi normale.

```
hist(train$T6,proba=TRUE)
curve(dnorm(x,mean(train$T6),sd(train$T6)),add=T,col='red')
```

```
qqnorm(train$T6)
```



T6suit une loi normale. On a aussi étudié les lois des autres variables comme **lights** qui suit une loi exponentielle mais ces études n'apportent pas grand chose à la suite et par souci de taille.

Modèles prédictifs

Dans le but d'effectuer une prédiction nous utiliserons plusieurs modèles différents afin de trouver le plus précis.Le critère de qualité de la prédiction utilisé sera le RMSE:

$$\text{RMSE} = \sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

. Voici l'implémentation de la fonction RMSE:

```
RMSE = function (y,y_hat){
  n=length(y)
  return(sqrt((1/n)*sum((y-y_hat)^2)))}
```

Notre but est donc d'obtenir le RMSE le plus faible sur le jeu de données test qui nous est donné par Kaggle lors de la soumission d'une prédiction.

Afin de tester et comparer la qualité de prédictions nous procèderons en général par validation croisée 2-fold en enlevant une partie de nos observations afin de tester la prédiction sur le jeu de données sans ces observations. Nous sélectionnerons un échantillon test de cette manière:

```
n <- nrow(train)
set.seed(101)
s <- sample(c(1:n), size=floor(n*0.1))
donnees_train = train[-s,]
donnees_test = train[s,]
```

Régression linéaire

Nous avons tout d'abord essayé de faire des prédictions en modélisant nos données par régréssion linéaire. Afin de minimiser le RMSE pour la régression linéaire il faudra trouver le meilleur modèle, c'est à dire seulement inclure les variables ayant un impact

Sélection de modèle basée sur les corrélations

Dans ce cas nous choisissons les variables fortement corélées.

Implémentation:

```
reg1 = lm(Appliances~ RH_2+RH_9+RH_7+RH_8+RH_1+RH_6+RH_out+
            BE_load_actual_entsoe_transparency+ T1+T3+T2+T6+T_out+
            Windspeed+lights+Instant, data =donnees_train)

pred_reg1 = predict(reg1,newdata=donnees_test)
RMSE(donnees_test$Appliances,pred_reg1)
```

```
## [1] 94.31994
```

On obtient un RMSE de 94.3 par validation croisée sur les variables choisies à partir de l'analyse des corrélations. Lors de la soumission sur Kaggle le score était dans ces ordres également.

Step Forward

On implémente une méthode de selection step forward, qui consiste à commencer avec une variable dans la régression et à en ajouter successivement. On choisit ensuite le modèle qui minimise le RMSE.

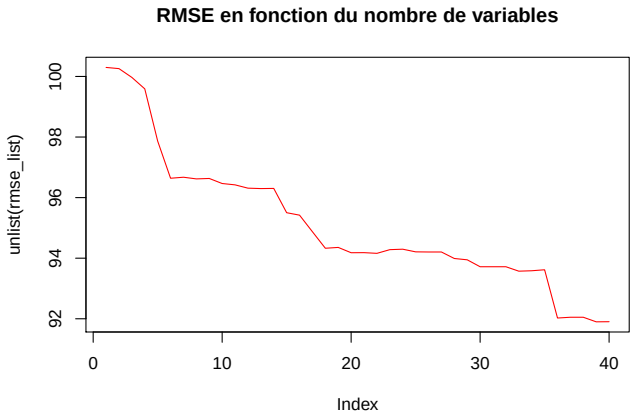
```
cov2 <- head(names(train)[-c(1,2,34)],40)
length(cov2)
```

```
## [1] 40
```

```
eq_list <- list()
eq_list[[1]] <-  paste0("Appliances ~", paste0(cov2[1], collapse='+'))
for(i in c(2:length(cov2)))
{
  eq_list[[i]] <-  paste0("Appliances ~", paste0(cov2[1:i], collapse='+'))
}

fitMod <- function(eq, subset)
{
  reg <- lm(eq, data=train[-subset,])
  return(reg)
}

reg_list <- lapply(eq_list, fitMod, subset=s)
reg_list_forecast <- lapply(reg_list, predict, newdata=train[s,])
rmse_list <- lapply(reg_list_forecast, RMSE, y=train[s,]$Appliances)
length(reg_list)
```

```
## [1] 40
```

```
plot(unlist(rmse_list),type='l',
col='red',main="RMSE en fonction du nombre de variables")
```




```
rmse_list[which.min(rmse_list)]
```

```
## [[1]]
## [1] 91.90055
```

On observe un RMSE de 91.9 pour le modèle prenant toutes les covariables en compte. Meilleur que avec le modèle step forward.

Stepwise AIC

Nous avons ensuite essayé le modèle Stepwise AIC qui consiste à choisir successivement les covariables à prendre en compte dans le modèle en allant dans différentes directions. En effet, le désavantage du step forward et qu'il ne considère pas toutes les combinaisons de covariables. La méthode stepwise sélectionnera le modèle avec les covariables tel que le critère AIC est minimisé. Voici l'implémentation de cette méthode

```
#Stepwise AIC on transofrmed variables
cov <- head(names(donnees_test)[-c(1,2,35)],41)
eq <- paste0("Appliances ~", paste0(cov, collapse='+'))
eq
```

```
## [1] "Appliances ~lights+T1+RH_1+T2+RH_2+T3+RH_3+T4+RH_4+T5+RH_5+T6+RH_6+T7+RH_7+T8+RH_8+T9+RH_9+T_ou
```

```
full.model <- lm(eq, data = donnees_test)

library(MASS)
step.model <- stepAIC(full.model, direction = "both", trace = FALSE, data=donnees_test)

step.model.forecast <- predict(step.model, newdata=donnees_test)
```

```
step.model$call
```

```
## lm(formula = Appliances ~ lights + RH_1 + T2 + RH_2 + T3 + RH_3 +
##     T4 + RH_4 + T6 + RH_6 + RH_7 + T8 + T9 + Windspeed + Day_of_week +
##     Month + BE_load_actual_entsoe_transparency + BE_wind_onshore_generation_actual,
##     data = donnees_test)
```

```
RMSE(donnees_test$Appliances, step.model.forecast)
```

```
## [1] 90.32114
```

On observe une amélioration de RMSE avec le stepwise AIC qui est désormais à 90.30

Afin de comparer le modèle prenant en compte les variables qualitatives sous forme numérique nous allons l'appliquer au jeux de données transformé.

```r
#Validation croisée sur le nouveau jeu de données (2 fold)
donnees_train_reworked = train_reworked[-s,]
donnees_test_reworked = train_reworked[s,]


#Stepwise AIC on transofrmed variables
cov2 <- head(names(donnees_train_reworked)[-c(1,2)],39)
eq2 <- paste0("Appliances ~", paste0(cov2, collapse='+'))
full.model2 <- lm(eq2, data = donnees_train_reworked)

library(MASS)
step.model2 <- stepAIC(full.model2, direction = "both", trace = FALSE, data=donnees_train_reworked)

step.model2$call
```

```
## lm(formula = Appliances ~ lights + T1 + RH_1 + T2 + RH_2 + T3 +
##     RH_3 + T4 + RH_4 + RH_5 + T6 + T7 + RH_7 + T8 + RH_8 + T9 +
##     RH_9 + T_out + Press_mm_hg + RH_out + Windspeed + Visibility +
##     Tdewpoint + NSM + WeekStatus + Heure + BE_load_actual_entsoe_transparency +
##     BE_load_forecast_entsoe_transparency + BE_wind_onshore_profile,
##     data = donnees_train_reworked)
```

```r
step.model.forecast2 <- predict(step.model2, newdata=donnees_test_reworked)
```

```r
RMSE(donnees_test_reworked$Appliances, step.model.forecast2)
```

```
## [1] 92.26027
```

On observe que le RMSE est légèrement plus élevé que avec le jeu de données initial mais très proche, ceci peut être du à un facteur aléatoire lors de la selection des variables du modèle AIC. Dans tous les cas nous utiliserons le jeu de données initial lorsque c'est possible et utiliserons le second si nous avons besoins de valeurs numériques seulement car il reste efficace pour prédire les valeurs du target.

Régression Ridge

Dans cette partie, on essaye d'utiliser la méthode de régression Ridge. Etant donnée λ, on cherche β qui minimise

$$\sum_{i=1}^{n}(y_i - x_i)^2 + \lambda\|\beta\|^2$$

. Par conséquent

$$\hat{\beta} = (X'X + \lambda I)^{-1}X'Y$$

et la prévision

$$\hat{Y} = X\hat{\beta}$$

.
Ayant besoin de variables numériques pour cette méthode nous utiliserons le jeu de données avec les transformations de variables quantitatives.

La tache est de trouver le paramètre λ minimisant le RMSE. On utilise la fonction *glmnet*.

```r
library(glmnet)

lambdas = seq(0,1,0.01)
ridge_mod = cv.glmnet(as.matrix(donnees_train_reworked[-c(1,2)]),
```

```
                      donnees_train_reworked$Appliances,alpha = 0,family = "gaussian",lambda = lambdas)
ridge_mod$lambda.min
```

```
## [1] 0.05
```

```
fit = ridge_mod$glmnet.fit
pred_ridge = predict(fit,s=ridge_mod$lambda.min,
                     newx=as.matrix(donnees_test_reworked[,-c(1,2)]))
RMSE(donnees_test_reworked$Appliances,pred_ridge)
```

```
## [1] 92.2035
```

Le RMSE n'est pas amélioré de cette manière et ceci est sûrement du au fait que nous avons beaucoup de variables dont nombreuses sont coréllées entre elles. Nous allons essayer d'ameliorer le modèle afin de ne garder que les variables expliquant la variabilité du modèle à travers les composantes principales du jeu de données. Nous avons vu que en gardant les 11 premières composantes principales nous expliquions 90% de la variance du modèle.

```
##On normalise le jeu de données et on effectue l'analyse en composante principales sans Appliances

scaled_train_reworked = scale(train_reworked[,-c(1,2,39)])*((N-1)/N)
PCA2 = PCA(scaled_train_reworked,ncp = 11,graph=FALSE)
barplot(PCA2$eig[,2])
PCA2$eig[11,3]
```

```
## [1] 90.93215
```

```
## On crée un jeu de données avec les 11 composantes principales

PCA_data = as.data.frame(unlist(PCA2$ind$coord))
#On ajoute Appliances au jeu de données

PCA_data$Appliances = train$Appliances

##On effectue la régression ridge sur ce jeu de données en CV

Xtrain_PCA = subset(PCA_data,select=-Appliances)
Xtrain_PCA = Xtrain_PCA[-s,]
Xtest_PCA = subset(PCA_data,select=-Appliances)
Xtest_PCA= Xtest_PCA[s,]

Ytrain_PCA = PCA_data$Appliances[-s]
Ytest_PCA = PCA_data$Appliances[s]

lambdas <- seq(0,1000,0.1)
ridge_mod = cv.glmnet(as.matrix(Xtrain_PCA),Ytrain_PCA,alpha = 0,family = "gaussian",lambda = lambdas)
ridge_mod$lambda.min
```

```
## [1] 0.7
```

```
fit = ridge_mod$glmnet.fit
pred_ridge2 = predict(fit,s=ridge_mod$lambda.min,newx=as.matrix(Xtest_PCA))
RMSE(Ytest_PCA,pred_ridge2)
```

```
## [1] 96.66012
```

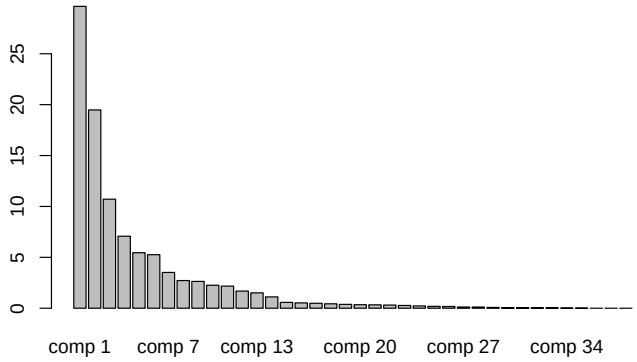
```
ridge_mod = cv.glmnet(as.matrix(donnees_train_reworked[-c(1,2)]),
                      donnees_train_reworked$Appliances,alpha = 0,family = "gaussian",lambda = lambdas)
```

```
ridge_mod$lambda.min
```

```
## [1] 0.1
```

```
fit = ridge_mod$glmnet.fit
pred_ridge = predict(fit,s=ridge_mod$lambda.min,
                     newx=as.matrix(donnees_test_reworked[,-c(1,2)]))
RMSE(donnees_test_reworked$Appliances,pred_ridge)
```

```
## [1] 92.2072
```



Il apparait que la régression ridge sur les composantes principales n'est pas aussi bonne que la régression ridge sur toutes les variables du jeu de données malgré un choix optimal du paramètre lambda (0.7) qui minimise le RMSE. On essayera alors d'autres méthodes

Regression LASSO

Nous avons essayé la régression LASSO,très proche de la régression ridge, en effet cette méthode consiste cette fois, étant donnée λ, à chercher β qui minimise

$$\sum_{i=1}^{n}(y_i - x_i)^2 + \lambda|\beta|$$

.

Ayant besoin de variables numériques pour cette méthode nous utiliserons le jeu de données avec les transformations de variables quantitatives.

```
library(glmnet)

lambdas = seq(0.001,1,0.01)

#On donne le vecteur avec plusieurs valeurs de lambda afin d'obtenir le lambda minimisant l'expression
ridge_mod_lasso = cv.glmnet(as.matrix(donnees_train_reworked[-c(1,2)]),
                            donnees_train_reworked$Appliances,alpha = 1,
                            family = "gaussian",lambda = lambdas)
ridge_mod_lasso$lambda.min
```

```
## [1] 0.001
```

```
fit = ridge_mod_lasso$glmnet.fit
pred_ridge = predict(fit,s=ridge_mod$lambda.min,
                     newx=as.matrix(donnees_test_reworked[,-c(1,2)]))
RMSE(donnees_test_reworked$Appliances,pred_ridge)
```

```
## [1] 92.19784
```

On obtient un RMSE similaire que avec la régression Ridge mais il semble qu'il n'y ait pas de lambda optimal dans les deux cas, en effet en commencant notre vecteur à 0 le paramètre optimal est 0, ceci est peut-être du au fait que de nombreuses sont corélées entre elles. Ainsi ces méthodes pourraient surement être plus efficaces après des transformations de variables et des réductions de dimension.

Afin de réduire la dimension du modèle nous avons tenté l'analyse en composantes principales qui n'a pas été saisfsaisante. Une autre méthode que nous avons tenté est de regrouper les températures et les humidités sous une variables car nous avons vu dans l'analyse descriptive qu'elles sont fortement corélées mais nous n'avons pas réussi à déterminer les poids optimaux pour exprimer chaque variable comme combinaison linéaire des autres.

Foret aléatoire

Nous nous sommes ensuite intéressé aux forêts aléatoires. Ces méthodes reposent sur les arbres de décision qui consistent à faire un découpage succéssif des données en plusieurs zones selon différentes variables afin de minimiser l'erreur entre la valeur de la réponse et celle de la prediction. Une forêt aléatoire construit plusieurs de ces arbres aléatoirement puis les moyenne.

Implémentation:

```
library(randomForest)
model_rf = randomForest(formula =  Appliances~lights+T1+ T2+ T3 +RH_1 +RH_2+RH_3+RH_5+T8+RH_8+T9+Windsp
predic1_test = predict(model_rf,newdata=test)

#Ici nous avons considéré NSM,Month et Posan tout de même
#car on obtenait de meilleurs résultats
full.model_rf <- randomForest(formula=Appliances ~lights+T1+RH_1+T2+RH_2+T3+RH_3+T4+RH_4+T5+RH_5+T6+RH_
                +Tdewpoint                  +rv1+rv2+NSM+WeekStatus+Day_of_week+DayType+Instant+Month+P
          +Heure+BE_load_actual_entsoe_transparency+
          BE_load_forecast_entsoe_transparency+
          BE_wind_onshore_generation_actual+BE_wind_onshore_profile,
          data = donnees_train)

predic1_test = predict(full.model_rf,newdata=donnees_test)

RMSE(donnees_test$Appliances,predic1_test)
```

```
## [1] 67.20859
```

On observe un RMSE bien meilleur que dans les autres méthodes 67.2. Cette méthode nous a permis d'obtenir un score d'environ 82 sur Kaggle. Nous n'avons pas essayé d'ajuster les hyperparamètres de la forêt tel que le nombre d'arbres et la profondeur des arbres car nous avons obtenus de bien meilleurs résultats avec le gradient boosting, que les calculs étaient très longs et que nous avons préféré consacrer ce temps à régler les hyperparamètres de cette dernière.

GAM

Comme le modèle de régression linéaire, on suppose que chaque composant du vecteur Y peut s'écrire comme

$$y_i = f_1(x_{1,i}) + f_2(x_{2,i}) + f_3(x_{3,i}, x_{4,i}) ... + \varepsilon_i$$

.
Chaque fonction f_j peut être estimée par une projection sur une base de spline, c'est-a-dire que

$$f_j(x) = \sum_{q=1}^{k_j} a_{j,q}(x)\beta_{j,q}$$

, où * k_j le nombre de la base de spline * $a_{j,q}(x)$ sont les fonctions de spline * $\beta_{j,q}$ sont les coefficients de regression.
Puisque nous ne savons pas quelles variables ont des effets linéaires, nous devons explorer plusieurs modèles de GAM et puis choisir celui qui minimise le RMSE le plus possible. Ci dessous une implémentation d'un modèle GAM

Nous avons regroupé les variables fortement corrélées entre elles sous une fonction à chaque fois et considéré lights et windspeed comme des effets linéaires.

```
g5<-gam(Appliances~lights+Windspeed+s(T2,T3,T7,T8,T9) +s(RH_1,RH_2,RH_3,RH_7,RH_9)+s(Day_of_week,Instan
        +s(BE_load_actual_entsoe_transparency
           ,BE_load_forecast_entsoe_transparency)
        +s(BE_wind_onshore_generation_actual)
        +s(BE_wind_onshore_profile),data=donnees_train_reworked)
gam5.forecast<-predict(g5,newdata = donnees_test_reworked)
RMSE(donnees_test_reworked$Appliances,gam5.forecast)
```

```
## [1] 84.44051
```

On observe une nette amélioration du RMSE qui est d'environ 84 pour la prédiction par rapport aux modèles précédents.

Série Chronologique

Dans cette partie nous présenterons une approche de modélisation de la variable *Appliance* en série chronologique.

En effet nous avons remarqué que nous pouvions séparer le jeu de données test en deux parties. En effet certaines dates correspondent à des instants manquant dans le jeu de données d'entraînement tandis que d'autres correspondent au futur. Afin de pouvoir modéliser la série chronologique nous avons donc prédit les valeurs d'appliance correspondant aux mesures manquantes à travers une régression linéaire afin de pouvoir modéliser la série et effectuer une prédiction de la deuxième partie qui correspond aux valeurs de *Appliance* dans le futur.

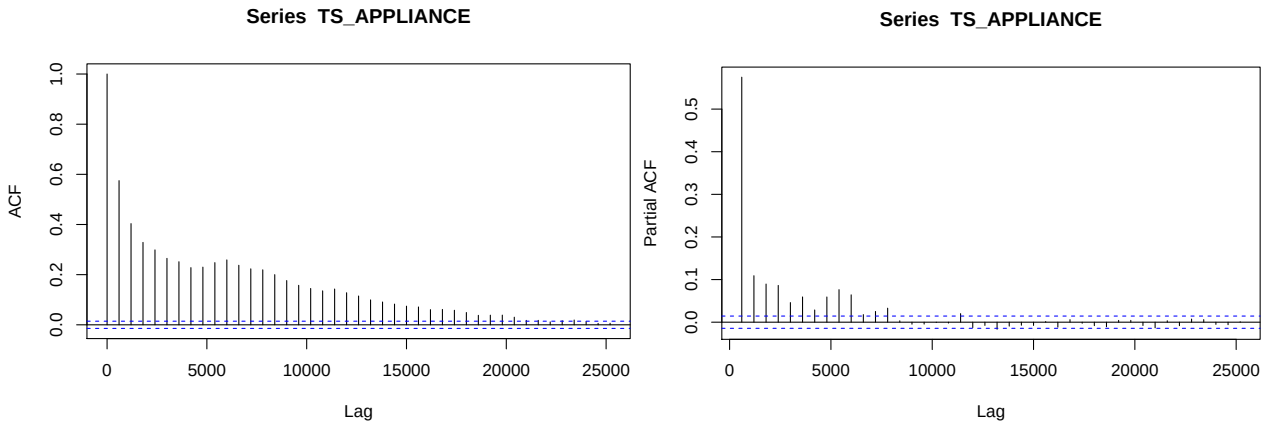
Voici comment nous avons procédé à séparer le jeu de donées test, insérer les dates des mesures manquantes dans le jeu de données d'entrainement, compléter les données données de *Appliance* pour ces dates puis transformé appliances en série chronologique:

```
test_chrono = test #Jeu de données test
test_chrono$Appliances = NA #on crée une colonne Appliance avec des NA
#on place Appliance à la même position que dans le jeu de données train
test_chrono = test_chrono[,c(1,44,c(2:42))]
#On ajoute les lignes de ce nouveau jeu au jeu d'entrainement
dataset_complet = rbind(train,test_chrono)
#On ordonne le jeu par date
dataset_complet= dataset_complet[order(dataset_complet$date),]
#indices des valeurs pour lesquelles Appliance = NA
indicesna = which(is.na(dataset_complet$Appliances))
#Indice des valeurs de Appliance sans NA
indicenonna = which(is.na(dataset_complet$Appliances)==F)
```

```r
#jeu de données sans les NA
dataset_complet_sans_NA = dataset_complet[-c(indicesna),]
#Que les NA
datatasetcomplet_NA = dataset_complet[c(indicesna),]
#valeurs de Appliances pour les dates sans NA
Appli = dataset_complet$Appliances[indicenonna]
#jeu de données d'entrainement avec les indices et Appliance connues
df2train = data.frame(indicenonna,Appli)
 #jeu de données test avec les indices correspondant aux valeurs manquantes
 #sans le futur
testset = as.data.frame(indicesna[1:4654])
#On change le nom de la colonne des indices du jeu de données
#test pour être le même que pour le jeu d'entrainement
colnames(testset)[1] = "indicenonna"
#on fait une régression linéaire pour prédire ces valeurs
linmod = lm(Appli~indicenonna,data=df2train)
vari= dataset_complet[indicesna[1:4654],]
predi = predict(linmod,newdata=testset) #prédiction
#on remplace par les valeurs de Appliance prédites
dataset_complet[indicesna[1:4654],2]= predi
#indices correspondant au futur
newindicena = which(is.na(dataset_complet$Appliances))
N_ = length(newindicena)
#on enleve les indices correspondant au futur de notre série
APLLIANCE = dataset_complet$Appliances[-newindicena]
#dates correspondant aux indices sans le futur
DATES_APPLI = dataset_complet$date[-newindicena]
nahead = length(newindicena)
#nous obtenons donc une série temporelle et le but et de la modéliser
#afin de prédire ses valeurs pour les dates correspondant au futur
TS_APPLIANCE = xts(APLLIANCE,order.by = DATES_APPLI)
```

Modélisation de la série

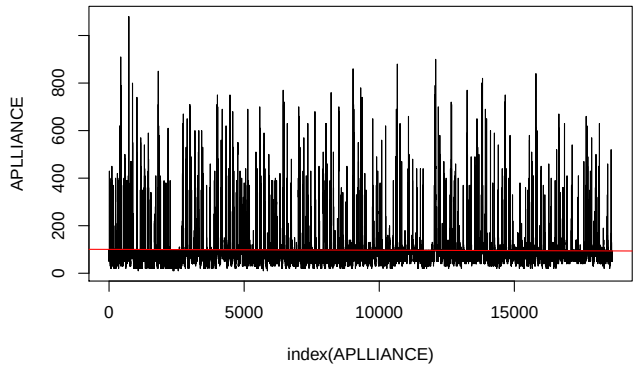
Nous avons ensuite essayé d'appliquerdes méthodes de lissage afin de modéliser la série. Tout d'abord nous étudions les autocorrélogrammes de la série :



On observe une décroissance très lente et il semble y avoir une ou deux composantes saisonnières ainsi qu'une tendance. Afin de savoir si nous pouvons faire une prédiction nous devons vérifier que la série sans sa tendance et sa composante saisonnière est stationnaire.

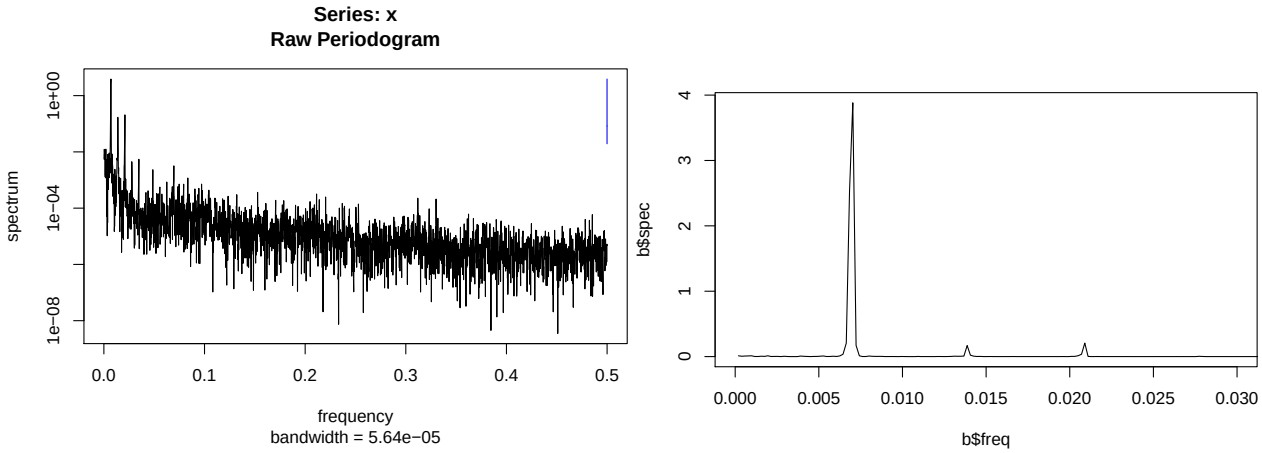
Pour la tendance nous avons effectué une régression linéaire puis nous avons extrait cette tendance du jeu de données

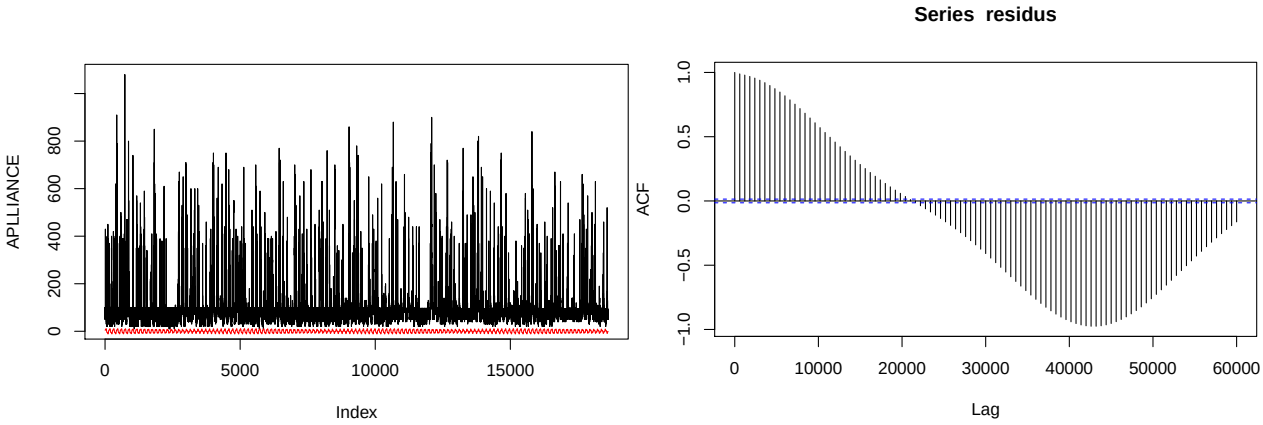
```
tendancelm = lm(APLLIANCE~index(DATES_APPLI))
plot(index(APLLIANCE),APLLIANCE,type='l')
abline(a=tendancelm$coefficients[1],b=tendancelm$coefficients[2],col='red')
```



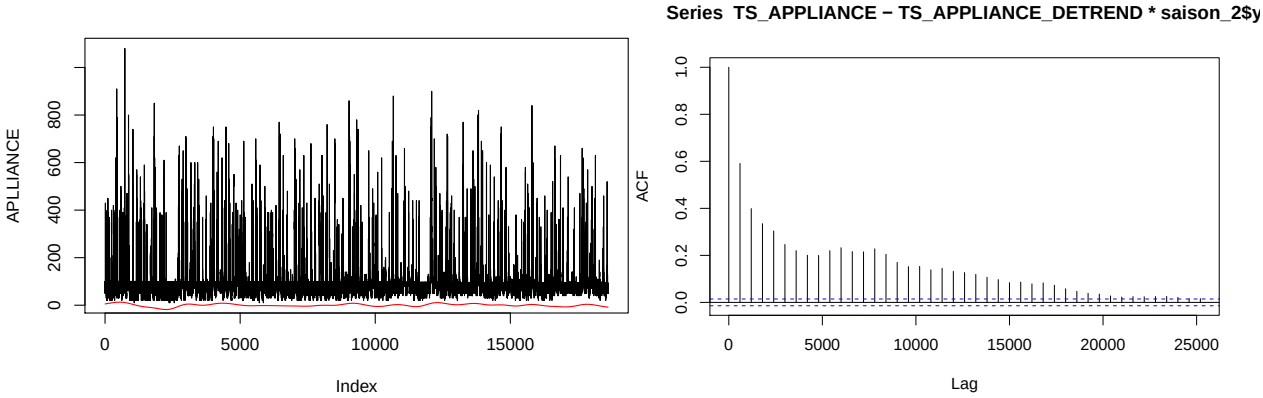

```
TS_APPLIANCE_DETREND = TS_APPLIANCE - tendancelm$fitted.values
```

Puis avons procédé par une décomposition en série de fourier et par noyau gaussien afin d'extraire la composante saisonnière de la série sans sa tendance.Nous avons considéré que le modèle était multiplicatif de la forme $X_t = T_t S_t + \varepsilon_t$ mais avons rencontré un problème lors de l'extraction de la composante saisonnière. En effet, afin d'évaluer les fréquences les plus importantes dans la série nous avons appliqué la fonction *spectrum* à l'autocorrélogramme afin d'extraire les fréquences ayant le plus de poids, mais il susbsiste à chaque fois une composante saisonnière. Ainsi nous n'avons pas réussi à stationnariser la série et n'avons donc pas pu faire de prédiction.Ceci est peut être du à une mauvaise hypothèse sur le modèle de la série mais un modèle additif ne fonctionnait pas non plus. Voici tout de même les résultats:



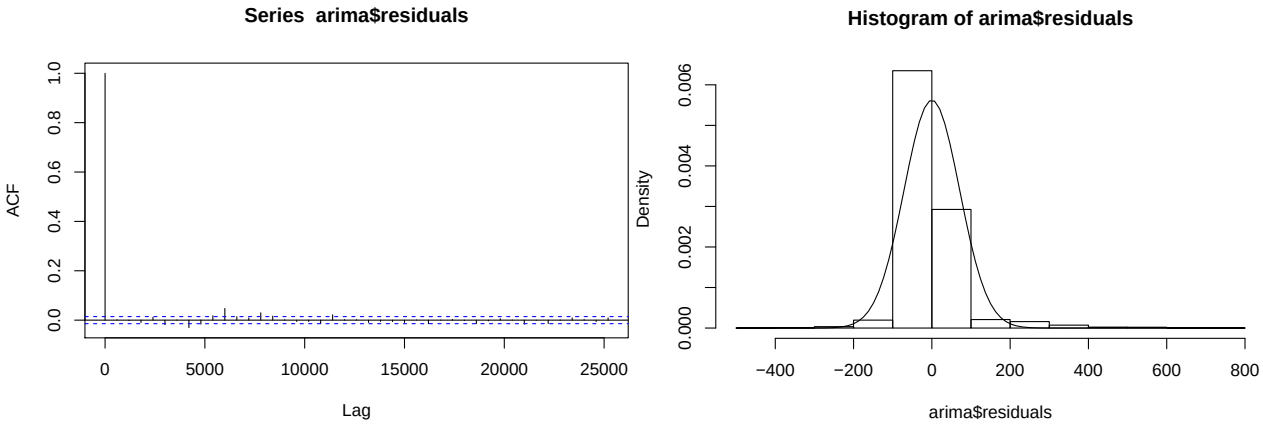


On observe aucune stationnarité sur l'autocorrélogramme des résidus et une composante saisonière persiste malgré l'extraction des trois fréquences observées sur le périodogramme. Ci dessous les résultats obtenus avec la composante saisonnière obtenue par méthode à noyau gaussien de fenêtre 1000. On observe toujours une composante saisonnière sur l'acf des résidus et une décroissance très lente.



On a finalement essayé une modélisation ARIMA afin de déterminer l'ordre optimal du processus avec le package forecast car les ordres p,q max des autocorrélogrammes étaient trop élevés.

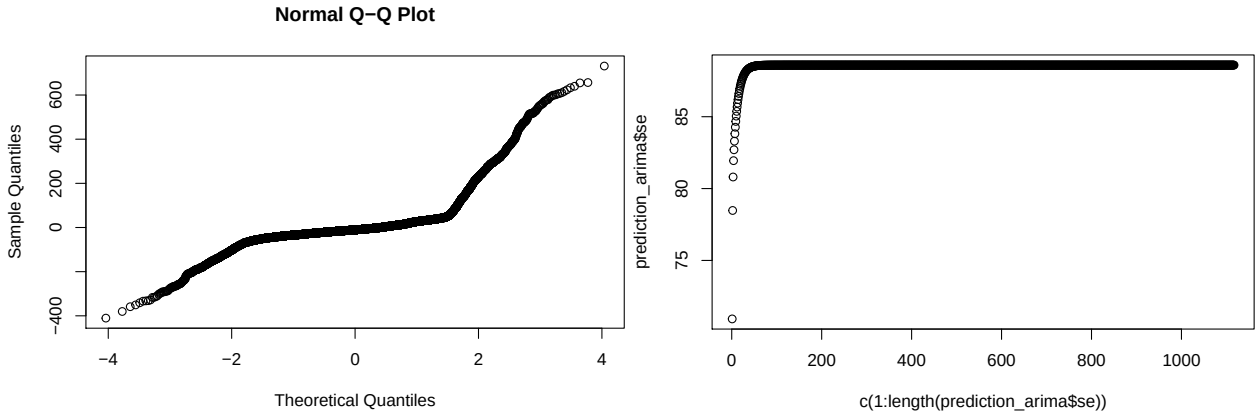
```
##Modélisation ARIMA

library(forecast)
arima = auto.arima(TS_APPLIANCE,ic="aic")
acf(arima$residuals)
hist(arima$residuals,proba=T)
curve(dnorm(x,0,sd(arima$residuals)),add = T)
```



```
Box.test(arima$residuals)
```

```
##
##  Box-Pierce test
##
## data:  arima$residuals
## X-squared = 0.068293, df = 1, p-value = 0.7938
```

```
qqnorm(arima$residuals)
prediction_arima=predict(arima,n.ahead = nahead)
plot(c(1:length(prediction_arima$se)),prediction_arima$se)
```



On obtient un arima d'ordre (2,0,1) et on observe que les résidus suivent une loi normale centrée, de plus le test de Box Pierce ne rejette pas l'hypothèse nulle d'indépendance des résidus et donc on peut considérer que c'est un bruit blanc, malheureusement la prediction devient constante à partir d'un certain rang ce qui n'est pas cohérent avec les prédictions que nous avons obtenues précédemment.

Régression et random forest en composantes principales

La régression ridge en composante principales n'ayant pas donné de résultats satisfaisant nous avons obtenu de meilleurs résultats en appliquant une simple régression sur les composantes principales obtenues de la même manière que précédemment.

```
library(pls)
set.seed (1000)
pcr_model <- pcr(Appliances~., data = donnees_train_reworked[,-c(1,39)], scale = TRUE, validation = "CV
yy=predict(pcr_model,donnees_test_reworked[,-c(1,39)],ncomp=11)
RMSE(donnees_test_reworked$Appliances,yy)
```

```
## [1] 96.65938
```

Nous avons aussi essayé une forêt aléatoire sur les composantes principales

```
PCA_rf=randomForest(x=Xtrain_PCA,y=Ytrain_PCA,mtry= 10000)
predict_PCA = predict(PCA_rf,Xtest_PCA)
RMSE(Ytest_PCA,predict_PCA)
```

Le RMSE obtenu est proche de celui de la forêt aléatoire simple (environ 68) donc peu moins bon, la réduction de la dimension du modèle n'a pas permis d'améliorer le score. Nous n'avons pas executé le code lors de la génération de ce rapport car le temps d'execution est très long.

Support Vector Machines

Les machines à vecteurs support sont très utilisés pour la classification. C'est une technique consistant à transformer le problème en un problème d'optimisation: celui de déterminer une frontière séparant les données de sorte à maximiser la distance des points les plus proches de cette frontière. Il est clair que cette méthode est pertinente pour les problèmes de classification mais il est possible de résoudre des problèmes de regression à travers cette méthode. Nous avons donc essayé de l'implémenter à travers le package e1071:

```
modelsvm = svm(Appliances~.,as.matrix(donnees_train_reworked[-c(1,31,32,34,35,39)]))
library(e1071)
predictions = predict(modelsvm,donnees_test_reworked[-c(1,31,32,34,35,39)])
RMSE(donnees_test_reworked$Appliances,predictions)
```

```
## [1] 87.51281
```

Nous ontenons un RMSE correct, meilleur que GAM et moins bon que les forêts aléatoires. Nous aurions pu l'améliorer à travers une 'grid search' sur les hyperparamètres de cet algorithme comme nous le ferons dans la méthode de gradient boosting ci-dessous. Cependant les temps de calculs sont longs et nous n'avons pas eu le temps. Il aurait fallu utiliser la fonction **tune** du package **e1071** et donner plusieurs valeurs pour les paramètres **epsilon** et **cost** (paramètre de la pénalisation dans le problème d'optimisation quadratique) afin de déterminer le modèle minimisant le RMSE. Cette méthode semble donc très éfficace.

Gradient boosting

Nous nous sommes ensuites intéressés aux méthodes de Gradient Boosting reposant aussi sur des arbres de décision. Ici l'algorithme évalue plusieurs modèles séquentiellement et chaque modèle se voit attribuer un poids en fonction de la précision de celui ci (basé sur une validation croisée dans notre cas). Le choix du modèle suivant est determiné en fonction du poids attribué. Dans le cas du gradient boosting, le poids est determiné à chaque itération par une descente de gradient.

Ces méthodes sont très utilisées aujourd'hui du à leur performances et à leur efficacité en terme de temps de calcul. Mais ces modèles comprenant une multitude d'hyperparamètres, il est indispensable de les régler afin d'obtenir des bons résultats. Ceci se fait manuellement et prend beaucoup de temps car les paramètres sont interdépendants donc il faut trouver une combinaison d'hyperparamètres optimaux. Ceci se fait à travers le package *Caret*, en effet nous donnons plusieurs valeurs possibles de chaque hyperparamètre tel que le nombres d'arbres, le profondeur maximale de l'arbre *Max_Depth*, le taux d'apprentissage *eta* associé au pas de l'optimisation de la fonction de perte à chaque itération, la proportion de variables sélectionnées pour la construction de chaque arbre *colsample_bytree*, la proportion du jeu d'apprentissage utilisée pour éffectuer la construction d'un arbre *subsample* ainsi que *min_child_weight* la somme des poids minimale à considérer pour le fils d'un noeud et *gamma* un paramètre de régularisation, une haute valeur de gamma réduira le sur apprentissage. Il existe d'autres paramètres mais nous avons réglés ceux là qui d'après la littérature ont le plus d'impact sur la précision de la prédiction.

Chacun de ces paramètres affecte le sur ou le sous apprentissage du modèle. Par exemple, une trop grande profondeur maximale des arbres peut générer du sur apprentissage et donc ceci peut être compensé par l'ajustement d'un des autres hyperparamètres. Il est donc difficile de trouver une combinaison optimale car si il fallait essayer toutes les combinaisons possibles le temps de calcul serait trop élevé malgré la possibilité d'effectuer les calculs en parallèle dans la fonction xgboost. Nous avons procédé comme suit:

- Nous avons donné plusieurs valeurs possibles du nombre d'arbres (entre 200 et 10 000) et du du taux d'apprentissage (entre 0.01 et 0.3) en fixant les autres paramètres et avons gardé les valeurs optimales
- Nous avons ensuite utilisé ces valeurs optimales, donné plusieurs valeurs possibles de max_depth (entre 6 et 20) et de min_child_weight (entre 0 et 2) en gardant les autres paramètres fixés
- Puis avons fait de même pour déterminer le reste des paramètres

Voici un exemple de réglage des hyperparamètres, puis l'apprentissage de ce modèle suivi de la prédiction:

```
CV_reworked_test = train_reworked[s,-c(1,2,39)]
CV_reworked_train = train_reworked[-s,-c(1,2,39)]
CV_Y_test = train_reworked[s,2]
CV_Y_train = train_reworked[-s,2]
##On donne la méthode de détermination des valeurs optimales à la fonction
##validation croisée 5-fold
xgb_trcontrol = trainControl(method = "cv", number = 5, allowParallel = TRUE,
                             verboseIter = FALSE, returnData = FALSE)
## On donne les combinaisons de paramètres à essayer:
xgbGrid =expand.grid(nrounds = c(20000),
                       max_depth = c(15),
                       colsample_bytree = c(0.3,0.4,0.5),
                       ## valeurs par défaut :
                       eta = c(0.1),
                       gamma=c(0,0.2,0.3,0.4),
                       min_child_weight = c(0,1),
                       subsample = c(1)
)
## On entraine le modèle pour chaque combinaison
set.seed(0)
xgb_model = train(CV_reworked_train, CV_Y_train, trControl = xgb_trcontrol,
tuneGrid = xgbGrid,
                  method = "xgbTree")

#On affiche le modèle avec les hyperparamètres optimaux
xgb_model$bestTune

#on effectue la prédiction
predicted = predict(xgb_model,CV_reworked_test)
RMSE(predicted,CV_Y_test)
```

Nous avons obtenus d'excellents résultats (RMSE de 64) avec cette méthode qui de loin a le mieux fonctionné, en effet nous avons réussi à dépasser le benchmark et la précision obtenue par forêt aléatoire qui était déjà très bonne.

Conclusion

En comparant toutes ces méthodes, les méthode de Gradient Boosting et de forêts aléatoires se sont clairement distinguées des autres suivies par les machines à vecteurs support. Ensuite les modèles GAM nous ont aussi donnés de bons résultats et nous pensons qu'avec une méthode plus pousséesur la sélection de variables et sur la considération des éffets linéaires et non linéaires (à travers différentes combinaisons et validation croisée de sorte à minimiser le RMSE) nous pouvons aussi obtenir de très bons résultats. Les techniques de régression pénalisées Ridge et LASSO ont étées décevantes et la méthode de régression classique aussi. Nous regrettons ne pas avoir pu générer une prédiction par la considération du target comme une série chronologique. Enfin, nous pensons que nous aurions pu améliorer le score des régressions en transformant plus nos variables, par exemple regrouper les températures sous une seule variable exprimée comme combinaison linéaire des autres et de même pour les humidités. L'ACP nous a aussi donné des résultats intéressants mais qui restent moins bon qu'en considérant les dimensions initiales du jeu de données.