Challenge: Prediction de la consommation electrique sur l'ile d'Ouessant

Cedric Bezy, Riwan Mouster (SIdRC)

Table of Contents

Présentation et contexte

Nous sommes étudiants en Master *Statistique et Informatique Décisionnelle* à Toulouse. Afin d’améliorer nos compétences en Machine Learning, nous participons de temps en temps à des compétitions de type *Kaggle*, ou autres.

Dans ce cadre, nous avons choisi de participer au *“Data Challenge”* organisé par le “groupe des Jeunes Statisticien.ne.s”. Ce projet consiste à prévoir la consommation électrique de l’île d’Ouessant pendant 8 jours, à chaque heure. Pour cela, nous disposons des fichiers suivants :

- **conso_train**: un an de données de consommation historiques, à la maille horaire
- **meteo_train**: un an de données météorologiques à la maille tri-horaire, issues de la proche station météorologique de Brest
- **meteo_pred**: une semaine de données météorologiques à la maille tri-horaire, issues de la même station et faisant office de prédiction météorologique. Dans ce projet, on considèrera ces prédictions comme étant exactes.
- **sample_submission**: l’exemple de fichier-type pour la soumission, à la maille horaire sur les 8 jours de prédiction.

Pour répondre aux attentes de ce projet, nous avons utilisé le logiciel R. Ce notebook décrit notre démarche ainsi que le code utilisé pendant ce projet.

Résumé

Notre démarche générale est la suivante :

1. Import des packages / données
2. Nettoyage des données
3. Création de nouvelles variables à partir des données
4. Statistique Descriptive
5. Modélisation et Prédiction

Les points 3, 4 et 5 sont itératifs, afin d’améliorer la prédiction.

Pour la prédiction, nous avons utilisé notamment *randomForest* et *xgboost*. La meilleure prédiction à été trouvée grâce à **xgboost**.

Mise en place de l'environnement

Avant de démarrer l'explication de la démarche, il est utile que le lecteur dispose du même environnement (chemin, packages, …)

Langue de l'environnement

Nous avons fais le choix de travailler dans un enviromment local anglais (notamment pour les dates).

```
# Set Option : Date in English
Sys.setlocale("LC_TIME", "English_United States")
```

```
## [1] "English_United States.1252"
```

Chemin d'accès aux dossiers souhaités

path_file contient une chaine donnant le dossier de localisation des fichiers nécessaires pour cette étude.

```
## A modifier
path_files <- "D:/Documents/PROJETS/Kaggle/ouessant/ouessant_copy/data"
```

Installation des packages

Ces lignes de code permettent d'installer les packages nécessaires.

```
# install.packages(c("dplyr", "tidyr"))
# install.packages(c("ggplot2", "ggpubr"))
# install.packages(c("circular", "lubridate", "magrittr"))
# install.packages(c("xgboost", "FactoMineR", "gsubfn"))
```

Importation des packages

Maintenant que les packages sont installés, importons-les dans l'environnement.

```
## Gestion DataFrames
library(dplyr)
library(tidyr)

## Plots
library(ggplot2)
library(ggpubr)

## Gestion des types particuliers (angle et dates)
library(circular)
library(lubridate)
```

```r
library(magrittr)

## Table disjonctive
library(FactoMineR)

## XGboost
library(xgboost)

## Simplification des résultats de fonctions en listes
library(gsubfn)
```

Fonctions

Les fonctions suivantes ont été programmées et utilisées pendant le projet.

Fonctions de mesure d'écart entre deux vecteurs

Dans ce projet, nos résultats seront évalués selon les critères *"Mean Absolute Percentage Error"* (**MAPE**), et *"Root Mean Square Error"* (**RMSE**). Il existe certes des packages donnant ces fonctions, mais les programmer par nous même nous permettra de mieux comprendre ces critères - en plus d'éviter à avoir à charger d'autres packages.

```r
## Fonction mape
MAPE <- function(y_pred, y_true){
    mape <- mean(abs((y_true - y_pred)/y_true))
    return(mape)
}
## Fonction RMSE
RMSE <- function(x, y){
    rmse <- sqrt(mean((x - y) ^ 2))
    return(rmse)
}
```

Fonctions Descriptives: Voir les données manquantes par variable

```r
# count_na compte le nombre de valeurs manquantes dans un vecteur x
count_na <- function(x){
    sum(is.na(x))
}

# prop_na donne la proportion de valeurs manquantes dans un vecteur x
prop_na <- function(x){
    mean(is.na(x))
}

#------------------------
# plotting missing values
#------------------------
# plot_na renvoie le diagramme en barre donnant la proportion de valeurs
```

```r
#   manquantes (NA) dans chaque colonne d'une data.frame.
# En entrée:
#   "df": data.frame à évaluer
#   pareto: logical
#       si TRUE, les barres sont ordonnées par ordre décroissant
#       si FALSE, alors les barres sont rangées par ordre alphabetique
#   simplify: logical
#       si TRUE, les colonnes sans valeurs manquantes ne sont pas affichés,
#           seules restent les variables avec valeurs manquantes
#       si FALSE, toutes les variables sont affichées
#   legend:  logical ; = TRUE si la legende doit etre affichee, =FALSE sinon
#   maintitle : titre du graphique (voir "label" de l fonction ggtitle)
#   subtitle : soustitre du graphique (voir la fonction ggtitle)
# En sortie :
#   un graphique de type ggplot

plot_na <- function(
    df,
    pareto = TRUE,
    simplify = TRUE,
    legend = FALSE,
    maintitle = "Missing Values",
    subtitle = NULL
){
    ## data : proportion of na
    dfna <- data.frame(
        variable = colnames(df),
        nb_na = sapply(df, count_na),
        p_na = sapply(df, prop_na)
    )
    if(simplify){
        dfna <- dfna %>% dplyr::filter(nb_na >= 1)
    }
    if(pareto){
        dfna <- dfna %>% dplyr::arrange(desc(p_na), variable)
        dfna$variable <- factor(dfna$variable, unique(dfna$variable))
    }
    ## Plot Layers
    resbarplot <- ggplot(data = dfna) +
        geom_bar(
            mapping = aes(variable, p_na, fill = p_na),
            stat = "identity",
            col = "black"
        )
    resbarplot <- resbarplot +
        ## modify x axis
        scale_x_discrete(name = NULL) +
        # modify y axis
        scale_y_continuous(
            name = "Prop NA",
```

```
            breaks = seq(0, 1, 0.1),
            limits = c(0, 1)
        ) +
        ## Modify colors
        scale_fill_continuous(name="Prop NA", high="#333333", low="#CCCCCC")

    ## set title abnd theme
    resbarplot <- resbarplot +
        ggtitle(maintitle, subtitle) +
        theme_bw()

    ## delete legend
    if(!legend){
        resbarplot <- resbarplot + guides(fill = FALSE)
    }
    ## results
    return(resbarplot)
}
```

Fonction pour diviser une table de données en deux échantillons.

```
# La fonction permet de spliter une data.frame en 2 echantillon selon une pro
portion ptrain entre 0 et 1.
# L'échantillon train contient une proportion ptrain de lignes
# l'échantillon test contient une proportion (1-ptrain) de lignes
# En sortie : une liste de deux data.frames "train"" et "test""

train_test_split <- function(data, ptrain = 0.75)
{
    N <- nrow(data)
    n <- floor(ptrain * N)
    s <- sample.int(N, n, replace = FALSE)
    reslist <- list(
        train = data[s,],
        test = data[-s,]
    )
    return(reslist)
}
```

Fonction pour traiter les variables numériques

Comment doivent être traitées les variables numériques ? En général, les valeurs manquantes sont remplacées par la moyenne de la variable dans l'échantillon d'apprentissage (train). Enfin, les variables peuvent être centrées et / ou réduites. Nous avons créé une fonction qui permettait de le faire, par niveau ou non.

Dans l'échantillon test, le remplacement des variables se fait sur le modèle de la table d'apprentissage (valeur de la moyenne et de l'écart-type).

Par niveau, cela signifie que la table est divisée en plusieurs morceaux selon des facteurs, le remplacement des valeurs manquantes et centrer / réduire se fait dans chaque morceau indépendamment des autres.

```
## function to replace na in a vector x, and center and scale if needed
## this function :
##      replace na by mean into x, center and scale x
##      then, it applies on y with x values

replace_na_center_scale <- function(x, y = NULL,
                                    na_replace = TRUE,
                                    center = TRUE,
                                    scale = TRUE)
{
    moy <- mean(x, na.rm = TRUE)
    if(na_replace){x <- replace(x, is.na(x), moy)}
    std <- sd(x, na.rm = TRUE)
    cent <- ifelse(center & !is.na(moy), moy, FALSE)
    scal <- ifelse(scale & !is.na(std) & std != 0, std, FALSE)
    x_scaled <- scale(x, cent, scal)[,1]
    ## Do the same on y, with x values
    if(length(y > 0)){
        if(na_replace){y <- replace(y, is.na(y), moy)}
        y_scaled <- scale(y, cent, scal)[,1]
        result <- list(x = x_scaled, y = y_scaled)
    }else{
        result <- x_scaled
    }
    return(result)
}

# cette fonction fait appel à "replace_na_center_scale" ci-dessus

# En entrée:
# train and test: deux data.frames
# variables : liste des colonnes à traiter
# by: liste des facteurs de division
# na_replace,  center, scale : logicals

# En sortie : une liste de taille 3:
#   - train : nouvelle table d'apprentissage
#   - test : nouvelle table de test
#   - columns : correspond aux variables effectivement modifiées

deal_train_test_numerics <- function(
    train,
    test,
    variables,
    by = NULL,
    na_replace = TRUE,
```

```
    center = TRUE,
    scale = TRUE
){
    numerics <- names(which(sapply(train[variables], is.numeric)))
    if(length(numerics) >= 1){
        ## keep only wished values to have the same structure
        train$DATA_ <- "train"
        train$ORDER_ <- 1:nrow(train)
        test$DATA_ <- "test"
        test$ORDER_ <- 1:nrow(test)
        all_variables <- c(numerics, by, "DATA_", "ORDER_")
        subTrain <- train[all_variables]
        subTest <- test[all_variables]
        ## Then, subTrain and subTest have the same structure
        # bind train and test data
        alldata <- rbind(subTrain, subTest)
        alldata$ORDER_ <- 1:nrow(alldata)

        # make a list of subdata
        if(length(by) >= 1){
            alldata_ls <- split(alldata, alldata[by])
        }else{
            alldata_ls <- list(alldata)
        }
        ## function to apply on each data.frame (see "replace_na_center_scale
")
        ## return a new data frame with arrangeddata
        .repl_scale_vars <- function(df){
            ok <- (df$DATA_ == 'train')
            scaled <- mapply(
                replace_na_center_scale,
                x = df[which(ok), numerics],
                y = df[which(!ok), numerics],
                na_replace = na_replace,
                center = center,
                scale = scale
            )
            if(any(!ok)){
                scaled <- bind_rows(apply(scaled, 1, bind_cols))
            }
            others <- setdiff(colnames(df), numerics)
            resDf <- cbind(df[others], scaled)
            return(resDf)
        }
        # applying .repl_scale_numerics on each subdata
        arrangedDf <- bind_rows(lapply(alldata_ls, .repl_scale_vars))

        # then, we can rebuild train and test
        arrangedTrain <- arrangedDf %>%
            dplyr::filter(DATA_ == "train") %>%
```

```
            dplyr::arrange(ORDER_)
        train[numerics] <- arrangedTrain[numerics]
        train$DATA_ <- NULL
        train$ORDER_ <- NULL

        arrangedTest <- arrangedDf %>%
            dplyr::filter(DATA_ == "test") %>%
            dplyr::arrange(ORDER_) %>%
            dplyr::mutate(DATA_ = NULL, ORDER_ = NULL)
        test[numerics] <- arrangedTest[numerics]
        test$DATA_ <- NULL
        test$ORDER_ <- NULL

        ## result list : train / test
        resLs <- list(train = train, test = test, columns = numerics)
    }else{
        resLs <- list(train = train, test = test, columns = NULL)
    }
    return(resLs)
}
```

Fonction pour transformer des facteurs en variables disjonctives

Nous avons également programmé une fonction permettant de remplacer les facteurs en variables disjonctives (1 ou 0), afin de pouvoir les traiter comme des variables numériques.

```
# cette fonction fait appel à "tab.disjonctif" du package FactoMineR

# En entrée:
# train and test: deux data.frames
# variables : liste des colonnes à transformer
# remove : = TRUE si les variables de bases sont à supprimer, = FALSe sinon

# En sortie : une liste de taille 3:
#   - train : nouvelle table d'apprentissage
#   - test : nouvelle table de test
#   - columns : correspond aux noms des nouvelles colonnes

deal_train_test_factors <- function(train, test, variables, remove = TRUE, ..
.)
{
    if(length(variables) >= 1){
        tabTrain <- FactoMineR::tab.disjonctif(train[variables])
        tabTest <- FactoMineR::tab.disjonctif(test[variables])
        columns <- unname(colnames(tabTrain))
        if(remove){
            train <- train[setdiff(colnames(train), variables)]
            test <- test[setdiff(colnames(test), variables)]
        }
        train <- cbind(train, tabTrain)
```

```
        test <- cbind(test, tabTest)
    }else{
        columns <- NULL
    }
    reslist <- list(train = train, test = test, columns = columns)
}

## cette fonction realise un xgboost sur l'ensemble des vecteurs contenues da
ns la list Ytrain par les données contenus dans Xtrain
## et renvoie une prediction pour chaque vecteur de Ytrain selon les donnees
de Xtest

xgboost_predict <- function(Ytrain, Xtrain, Xtest, ...)
{
    ## prediction sur une colonne, par xgboost
    ## ... = arguments of Xgboost
    .subpredict <- function(y){
        w <- which(!is.na(y))
        bst <- xgboost(
            data = as.matrix(Xtrain[w,]),
            label = y[w],
            ...
        )
        pred <- predict(bst, as.matrix(Xtest))
        return(list(pred = pred, bst = bst))
    }
    predict_list <- lapply(Ytrain, .subpredict)
    predDf <- as.data.frame(lapply(predict_list, function(x){x$pred}))
    predBst <- lapply(predict_list, function(x){x$bst})
    reslist <- list(predictions = predDf, bst = predBst)
    return(reslist)
}
```

Gestion des Données

Maintenant que l'environnement de travail a été initialisé, nous pouvons nous occuper des données

Importation des données

```
# Importation de conso_train
conso_train <- read.csv2(
    file = sprintf("%s/conso_train.csv", path_files),
    dec = ".",
    stringsAsFactors = FALSE
)

## Importation de meteo_train
meteo_train <- read.csv2(
    file = sprintf("%s/meteo_train.csv", path_files),
```

```
    dec = ".",
    stringsAsFactors = FALSE
)

## Importation de meteo_prev
meteo_prev <- read.csv2(
    file = sprintf("%s/meteo_prev.csv", path_files),
    dec = ".",
    stringsAsFactors = FALSE
)
```

Un problème rencontré au cours de l'import est que R ne reconnait pas l'encodage des noms de colonnes. Aussi avons-nous renommé les colonnes afin de faciliter l'utilisation des méthodes de machine learning.

```
colnames(conso_train)

## [1] "ï..date"   "puissance"

colnames(meteo_train)

##  [1] "Date.UTC"        "T....C."         "P..hPa."
##  [4] "HR...."          "P.ros..e....C."  "Visi..km."
##  [7] "Vt..moy...km.h." "Vt..raf...km.h." "Vt..dir....."
## [10] "RR.3h..mm."      "Neige..cm."      "Nebul...octats."

## rename conso_train
colnames(conso_train) <- c('datetime', 'puissance')

## rename meteo_train
newnames <- c(
    "datetime", "temp", "pression", "hr", "p_rosee",
    "visi", "vt_moy", "vt_raf", "vt_dir", "rr_3h",
    "neige", "nebul"
)
colnames(meteo_train) <- newnames
colnames(meteo_prev) <- newnames
```

Description des données

Les données météo sont décrites comme suit :

- **datetime**: dates des observations (sera convertie en Date)
- **temp** : température (°C)
- **pression** : pression (hPa)
- **hr** : hmidité relative (%), mesure la quantité de vapeur d'eau présente dans l'air
- **p_rosee** : point de rosée (°C)
- **visi** : visibilité (km)
- **vt_moy** : vitesse moyen du vent (km/h)
- **vt_raf** : vitesse des rafales de vent (km/h)

- **vt_dir** : direction du vent (degrés, de 0 à 359)
- **rr_3h** : précipitations sur 3h (mm)
- **neige** : neige (cm)
- **nebul** : nébulosité, ou couverture nuageuse (octats)

```
summary(conso_train)
```

```
##    datetime           puissance
##  Length:8760        Min.   : 294.2
##  Class :character   1st Qu.: 527.0
##  Mode  :character   Median : 719.2
##                     Mean   : 752.6
##                     3rd Qu.: 950.2
##                     Max.   :1732.2
```

```
summary(meteo_train)
```

```
##    datetime              temp          pression            hr
##  Length:2928        Min.   :-0.70   Min.   : 976.4   Min.   : 29.00
##  Class :character   1st Qu.: 9.30   1st Qu.:1012.6   1st Qu.: 72.75
##  Mode  :character   Median :12.40   Median :1018.4   Median : 84.00
##                     Mean   :12.37   Mean   :1016.8   Mean   : 81.31
##                     3rd Qu.:15.30   3rd Qu.:1023.2   3rd Qu.: 92.00
##                     Max.   :32.10   Max.   :1037.7   Max.   :100.00
##                     NA's   :64      NA's   :64       NA's   :64
##     p_rosee            visi          vt_moy          vt_raf
##  Min.   :-2.800   Min.   : 0.00   Min.   : 0.00   Min.   :  3.704
##  1st Qu.: 5.700   1st Qu.:10.00   1st Qu.: 9.26   1st Qu.: 18.520
##  Median : 9.500   Median :16.00   Median :16.67   Median : 29.632
##  Mean   : 9.043   Mean   :19.27   Mean   :17.59   Mean   : 31.210
##  3rd Qu.:12.300   3rd Qu.:25.00   3rd Qu.:24.08   3rd Qu.: 42.596
##  Max.   :19.200   Max.   :60.00   Max.   :66.67   Max.   :105.564
##  NA's   :64       NA's   :64      NA's   :64      NA's   :67
##      vt_dir           rr_3h            neige          nebul
##  Min.   :  0.0   Min.   : 0.000   Min.   :0      Min.   :0.000
##  1st Qu.:150.0   1st Qu.: 0.000   1st Qu.:0      1st Qu.:6.000
##  Median :220.0   Median : 0.000   Median :0      Median :7.000
##  Mean   :202.5   Mean   : 0.409   Mean   :0      Mean   :6.416
##  3rd Qu.:270.0   3rd Qu.: 0.000   3rd Qu.:0      3rd Qu.:8.000
##  Max.   :360.0   Max.   :18.000   Max.   :0      Max.   :8.000
##  NA's   :64      NA's   :336      NA's   :1957   NA's   :409
```

```
summary(meteo_prev)
```

```
##    datetime              temp          pression          hr
##  Length:65          Min.   : 9.80   Min.   :1004   Min.   :62.00
##  Class :character   1st Qu.:13.90   1st Qu.:1010   1st Qu.:82.00
##  Mode  :character   Median :15.70   Median :1020   Median :93.00
##                     Mean   :15.55   Mean   :1017   Mean   :88.14
##                     3rd Qu.:17.10   3rd Qu.:1023   3rd Qu.:97.00
##                     Max.   :23.30   Max.   :1025   Max.   :99.00
```

```
##
##     p_rosee           visi          vt_moy          vt_raf
##  Min.   : 9.30   Min.   : 0.10   Min.   : 0.00   Min.   : 3.60
##  1st Qu.:11.60   1st Qu.: 6.00   1st Qu.: 7.20   1st Qu.:12.96
##  Median :13.60   Median :18.00   Median :10.80   Median :18.00
##  Mean   :13.45   Mean   :21.57   Mean   :11.76   Mean   :21.36
##  3rd Qu.:14.80   3rd Qu.:35.00   3rd Qu.:14.82   3rd Qu.:25.93
##  Max.   :18.80   Max.   :60.00   Max.   :32.40   Max.   :54.00
##
##      vt_dir          rr_3h           neige            nebul
##  Min.   :  0.0   Min.   :0.0000   Min.   :  0.00   Min.   :6.000
##  1st Qu.:100.0   1st Qu.:0.0000   1st Qu.:  0.00   1st Qu.:7.750
##  Median :300.0   Median :0.0000   Median :  0.00   Median :8.000
##  Mean   :218.9   Mean   :0.2586   Mean   :112.33   Mean   :7.729
##  3rd Qu.:330.0   3rd Qu.:0.0000   3rd Qu.: 37.75   3rd Qu.:8.000
##  Max.   :360.0   Max.   :9.0000   Max.   :826.00   Max.   :8.000
##                  NA's   :7        NA's   :1        NA's   :17
```

Nettoyage des données : suppression des doublons

L'étape suivante de la gestion des données consiste à nettoyer les données avant tout. C'est-à-dire de supprimer les lignes présentes plusieurs fois, et convertir les colonnes au bon format. "Le nettoyage des données est la base d'une bonne prédiction."

```
## supprimer les doublons dans conso_train
dup_conso <- which(duplicated(conso_train))
conso_train <- conso_train[-dup_conso,]
print(sprintf("conso_train : %i lignes supprimées !", length(dup_conso)))

## [1] "conso_train : 1 lignes supprimées !"

## supprimer les doublons dans meteo_train
dup_meteo <- which(duplicated(meteo_train))
meteo_train <- meteo_train[-dup_meteo,]
print(sprintf("meteo_train : %i lignes supprimées !", length(dup_meteo)))

## [1] "meteo_train : 8 lignes supprimées !"
```

Formattage des variables

Dans la table *conso_train*, la date doit être formatée, afin d'être convertie en date - sous R, il s'agit du type *"POSIXt"*. Nous conserverons également le fuseau horaire.

De plus, certaines puissances sont données aux temps hh:59:59 à chaque heure. Considérant la seconde négligeable dans notre échelle de travail, et pour le besoin de l'étude, nous avons décidé d'arrondir ces dates à l'heure (soit une seconde plus tard).

```
## extraire la date
conso_dt <- strsplit(conso_train$datetime, split = "\\+")

## Extraire la date
```

```
conso_dt_format <- as.POSIXct(strptime(
    sapply(conso_dt, function(x){x[1]}),
    format = "%Y-%m-%dT%H:%M:%S"
))

## Enfin, on arrondit la date
conso_train$datetime <- round_date(conso_dt_format, "hour")
## fuseau horaire
conso_train$fuseau <- paste0("+", sapply(conso_dt, function(x){x[2]}))
```

Dans les tables *meteo_train* et *meteo_prev*, nous devons également gérer les dates, mais pas seulement. En effet, la variable *vt_dir* est une variable circulaire : un vent de direction 350 et 10 degrés sont proches, et pourtant le vent moyen donnerait un vent en direction contraire (180 degrés).

```
## dans meteo train
dt_meteotrain <- as.POSIXct(strptime(
    meteo_train$datetime,
    format = "%d/%m/%y %Hh%M"
))
meteo_train <- meteo_train %>%
    dplyr::mutate(
        datetime = dt_meteotrain,
        vt_dir = circular(vt_dir, type = "direction", units = "degrees", zero
= 0)
    )

## dans meteo pred
dt_meteoprev <- as.POSIXct(strptime(
    meteo_prev$datetime,
    format = "%d/%m/%y %Hh%M"
))
meteo_prev <- meteo_prev  %>%
    dplyr::mutate(
        datetime = dt_meteoprev,
        vt_dir = circular(vt_dir, type = "direction", units = "degrees", zero
= 0)
    ) %>%
    ## /!\ meteo_prev a une ligne en trop
    dplyr::filter(datetime <= as.POSIXct("2016-09-20 23:00:00"))
```

Création de variables

A partir des données, nous avons créé d'autres variables explicatives, dont certaines dérivées de la dates.

- **Weekday** : jour de la semaine ("Monday", ..., "Sunday")
- **Day**: jour dans le mois
- **Month**: mois de l'année ("January", ..., "December")

- **Year**: année
- **Hour**: heure dans la journée

En outre, nous avons établis d'autres facteurs:

- **rose_vt**: faire une moyenne sur la direction du vent n'a concrètement aucun sens. Nous avons donc décidé de découper cette variable en plusieurs classes, selon la rose de direction :
  - *North*: vent venant du nord (315 degrés ou +, ou en dessous de 45 degrés).
  - *East*: vent venant de l'est (entre 45 et 134.9 degrés)
  - *South*: vent venant du Sud (entre 135 et 224.9 degrés)
  - *West*: vent venant de l'Ouest (entre 255 et 314.9 degrés)
- **f_weekend**: =1 si Weekday = "Saturday" ou "Sunday", =0 sinon.
- **f_season**: ="summer" entre les mois d'Avril à Octobre, et "winter" entre les mois de Novembre à Mars
- **f_evening**: Cette variable permet de mettre l'accent sur les pics de fin de journée, en général lorsque les individus consomment le plus d'électricité:
  - en été ("summer"), =1 au dessus de 21h, =0 sinon.
  - en hiver ("winter"), =1 au dessus de 18h, =0 sinon

```
meteo_train <- meteo_train %>%
    dplyr::mutate(
        Weekday = wday(datetime, label = TRUE, abbr = FALSE),
        Day = day(datetime),
        Month = month(datetime, label = TRUE, abbr = FALSE),
        Year = year(datetime),
        Hour = hour(datetime),
        rose_vt = cut(
            replace(vt_dir, vt_dir >= 315, 0),
            breaks = c(0, seq(45, 315, by = 90)),
            labels = c("North", "East", "South", "West"),
            right = FALSE
        ),
        f_weekend = Weekday %in% c("Saturday", "Sunday"),
        f_season = factor(
            ifelse(month(datetime) %in% c(4:10), "summer", "winter"),
            levels = c("summer", "winter")
        ),
        f_evening = as.numeric(
            ifelse(month(datetime) %in% c(4:10), Hour >= 21, Hour >= 18)
        )
    )

str(meteo_train)

## 'data.frame':    2920 obs. of  21 variables:
##  $ datetime : POSIXct, format: "2015-09-13 00:00:00" "2015-09-13 03:00:00"
...
##  $ temp     : num  12.5 12.3 12.3 14.2 13.3 15.5 15.3 14.7 14.3 12 ...
```

```
##  $ pression : num  1009 1006 1005 1003 1001 ...
##  $ hr       : int  81 83 82 80 93 91 86 89 91 92 ...
##  $ p_rosee  : num  9.3 9.5 9.3 10.8 12.2 14 13 12.9 12.8 10.7 ...
##  $ visi     : num  40 40 40 40 4 10 20 20 15 5 ...
##  $ vt_moy   : num  9.26 11.11 14.82 18.52 18.52 ...
##  $ vt_raf   : num  18.5 16.7 22.2 31.5 38.9 ...
##  $ vt_dir   :Classes 'circular', 'integer'  atomic [1:2920] 140 120 130 14
0 140 160 220 210 240 270 ...
##   .. ..- attr(*, "circularp")=List of 6
##   .. .. ..$ type    : chr "directions"
##   .. .. ..$ units   : chr "degrees"
##   .. .. ..$ template: chr "none"
##   .. .. ..$ modulo  : chr "asis"
##   .. .. ..$ zero    : num 0
##   .. .. ..$ rotation: chr "counter"
##  $ rr_3h    : int  0 0 NA NA 4 6 NA 0 1 2 ...
##  $ neige    : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ nebul    : int  8 8 7 7 7 7 6 6 6 7 ...
##  $ Weekday  : Ord.factor w/ 7 levels "Sunday"<"Monday"<..: 1 1 1 1 1 1 1 1
2 2 ...
##  $ Day      : int  13 13 13 13 13 13 13 13 14 14 ...
##  $ Month    : Ord.factor w/ 12 levels "January"<"February"<..: 9 9 9 9 9 9
9 9 9 9 ...
##  $ Year     : num  2015 2015 2015 2015 2015 ...
##  $ Hour     : int  0 3 6 9 12 15 18 21 0 3 ...
##  $ rose_vt  : Factor w/ 4 levels "North","East",..: 3 2 2 3 3 3 3 3 4 4 ..
.
##  $ f_weekend: logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
##  $ f_season : Factor w/ 2 levels "summer","winter": 1 1 1 1 1 1 1 1 1 1 ..
.
##  $ f_evening: num  0 0 0 0 0 0 0 1 0 0 ...
```

```r
meteo_prev <- meteo_prev %>%
    dplyr::mutate(
        Weekday = wday(datetime, label = TRUE, abbr = FALSE),
        Day = day(datetime),
        Month = month(datetime, label = TRUE, abbr = FALSE),
        Year = year(datetime),
        Hour = hour(datetime),
        rose_vt = cut(
            replace(vt_dir, vt_dir >= 315, 0),
            breaks = c(0, seq(45, 315, by = 90)),
            labels = c("North", "East", "South", "West"),
            right = FALSE
        ),
        f_weekend = Weekday %in% c("Saturday", "Sunday"),
        f_season = factor(
            ifelse(month(datetime) %in% c(4:10), "summer", "winter"),
            levels = c("summer", "winter")
        ),
```

```
        f_evening = as.numeric(
            ifelse(month(datetime) %in% c(4:10), Hour >= 21, Hour >= 18)
        )
    )

str(meteo_prev)
```

```
## 'data.frame':    64 obs. of  21 variables:
##  $ datetime : POSIXct, format: "2016-09-13 00:00:00" "2016-09-13 03:00:00"
...
##  $ temp     : num  18.3 17.1 17.6 20.1 23.3 18.8 17.2 17.2 17.5 16.2 ...
##  $ pression : num  1009 1007 1008 1007 1006 ...
##  $ hr       : int  97 98 99 87 76 93 98 98 97 97 ...
##  $ p_rosee  : num  17.8 16.8 17.4 17.9 18.8 17.6 16.9 16.9 17 15.7 ...
##  $ visi     : num  9 0.4 5 45 40 5 3.9 3 9 2.9 ...
##  $ vt_moy   : num  12.96 3.7 5.56 11.11 14.82 ...
##  $ vt_raf   : num  20.4 11.1 14.8 18.5 25.9 ...
##  $ vt_dir   :Classes 'circular', 'integer'  atomic [1:64] 180 50 160 40 10
10 360 60 110 80 ...
##   .. ..- attr(*, "circularp")=List of 6
##   .. .. ..$ type    : chr "directions"
##   .. .. ..$ units   : chr "degrees"
##   .. .. ..$ template: chr "none"
##   .. .. ..$ modulo  : chr "asis"
##   .. .. ..$ zero    : num 0
##   .. .. ..$ rotation: chr "counter"
##  $ rr_3h    : int  0 0 0 0 0 0 2 0 3 0 ...
##  $ neige    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nebul    : int  7 7 8 8 NA 8 8 8 8 NA ...
##  $ Weekday  : Ord.factor w/ 7 levels "Sunday"<"Monday"<..: 3 3 3 3 3 3 3 3
4 4 ...
##  $ Day      : int  13 13 13 13 13 13 13 13 14 14 ...
##  $ Month    : Ord.factor w/ 12 levels "January"<"February"<..: 9 9 9 9 9 9
9 9 9 9 ...
##  $ Year     : num  2016 2016 2016 2016 2016 ...
##  $ Hour     : int  0 3 6 9 12 15 18 21 0 3 ...
##  $ rose_vt  : Factor w/ 4 levels "North","East",..: 3 2 3 1 1 1 1 2 2 2 ..
.
##  $ f_weekend: logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ f_season : Factor w/ 2 levels "summer","winter": 1 1 1 1 1 1 1 1 1 1 ..
.
##  $ f_evening: num  0 0 0 0 0 0 0 1 0 0 ...
```

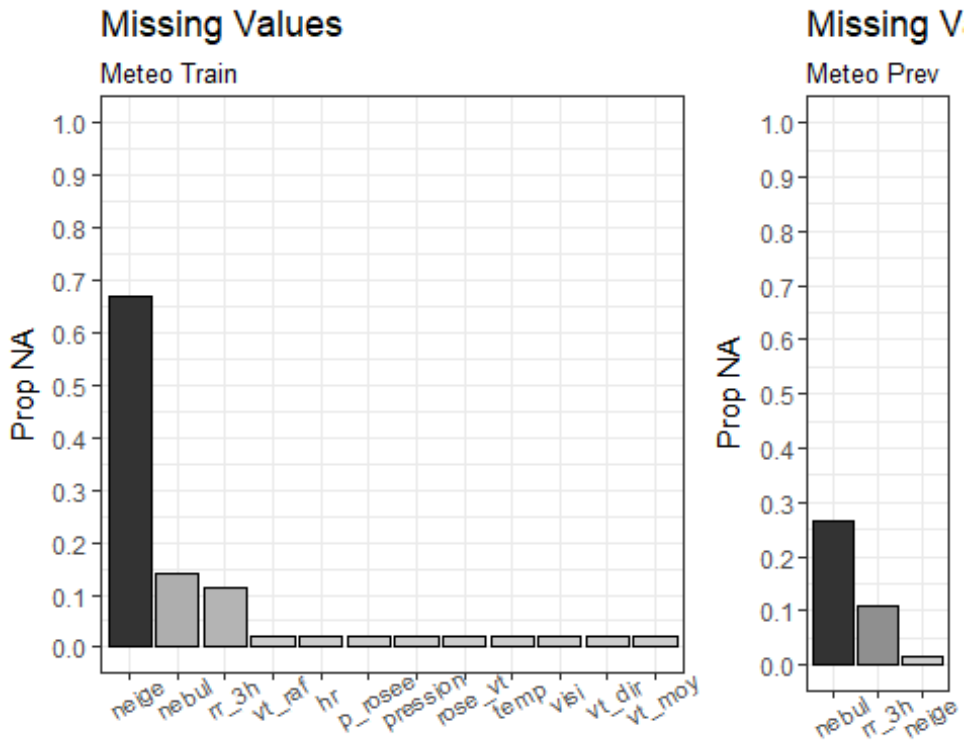
Nous avons à présent nos données.

Statistique Descriptive

Avant tout, il est recommandé de faire des statistiques descriptives. Nous allons ici décrire quelques statistiques qui ont eu de l'importance dans notre démarche.

Valeurs manquantes

```
ggarrange(
    plot_na(meteo_train, subtitle = "Meteo Train") +
        theme(axis.text.x = element_text(angle = 30)),
    plot_na(meteo_prev, subtitle = "Meteo Prev") +
        theme(axis.text.x = element_text(angle = 30)),
    nrow = 1,
    ncol = 2,
    widths = c(8, 3)
)
```



Nous remarquons qu'il y a beaucoup de valeurs manquantes pour la variable *neige* . Nous supposons que cette variable n'a pas été remplie lorsqu'il n'a pas neigé. Cette variable semble donc perdre de l'importance.

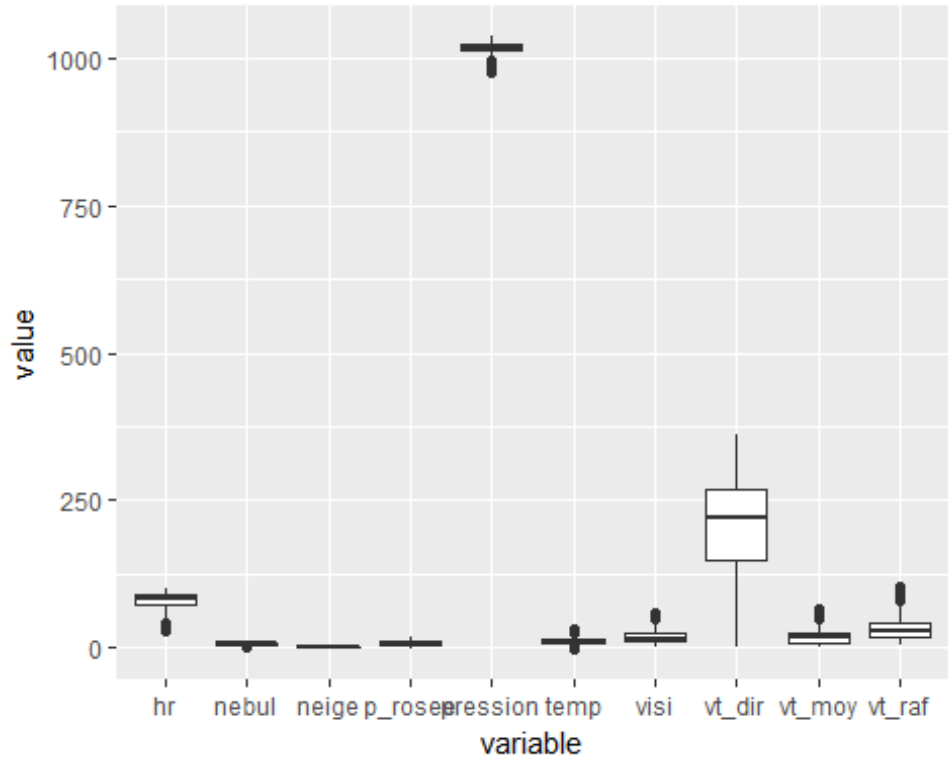
Dans le jeu test, il manque également certaines données pour les variables *nebul* et *rr_3h*. Il faudra remplacer les valeurs manquantes, probablement par une moyenne. Les valeurs manquantes

Dispersion des variables explicatives

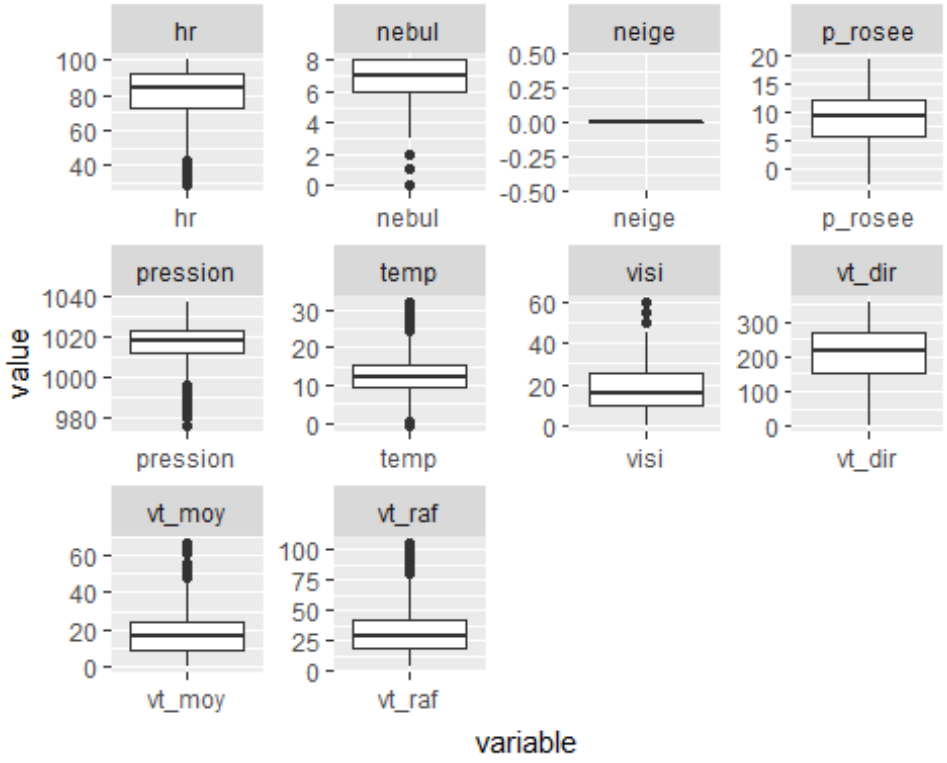
```
df1 <- tidyr::gather(
    meteo_train,
    "variable",
    "value",
    c(temp, pression, hr, p_rosee, visi, vt_moy, vt_raf, vt_dir, neige, nebul
)
```

```
)
boxplot <- ggplot(data = df1) +
    geom_boxplot(
        mapping = aes(variable, value),
        na.rm = TRUE
    )

print(boxplot)
```



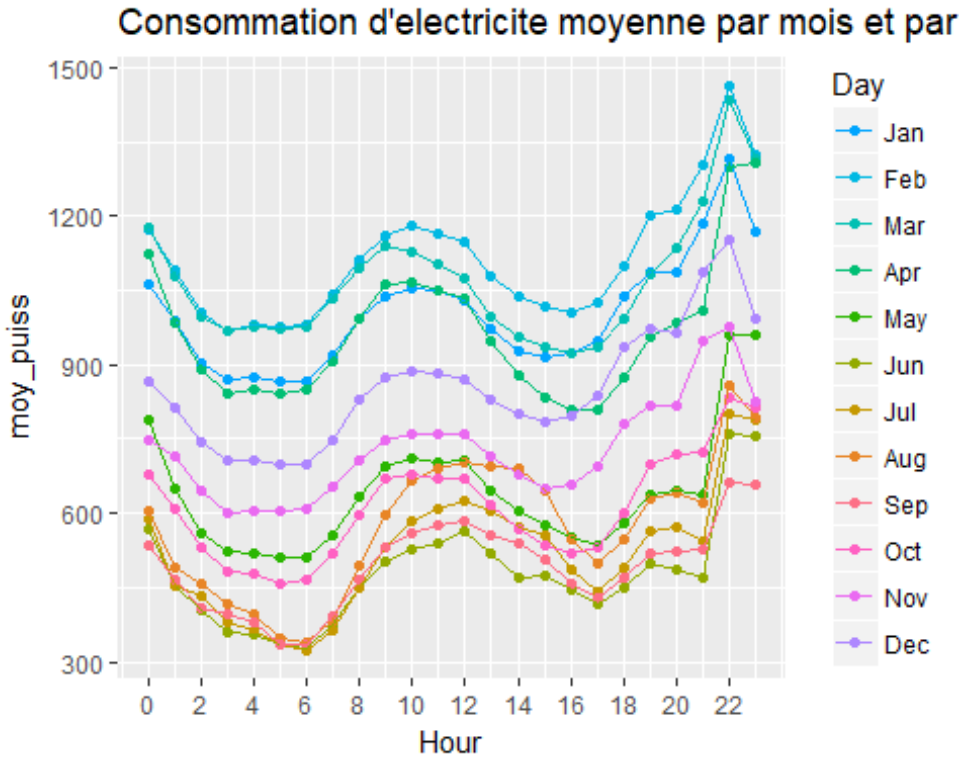

```
print(boxplot + facet_wrap(~variable, scales = "free"))
```

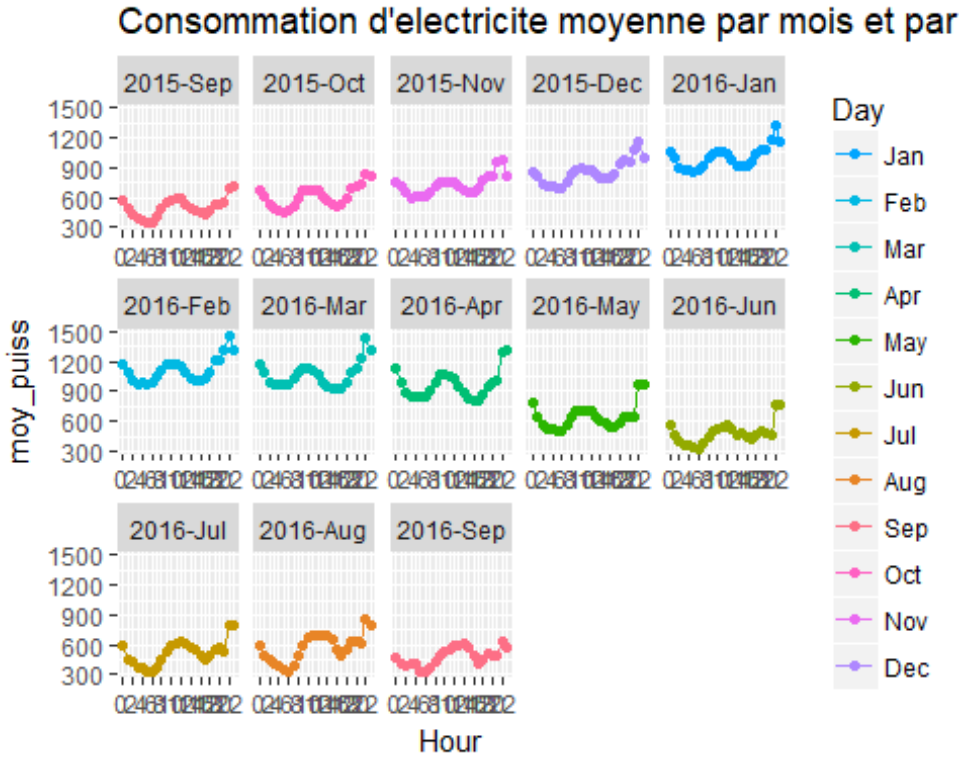


Les boxplots ci-dessus montrent que nos variables explicatives se dispersent dans des échelles différentes. Afin de limiter les biais liés à cette dispersion, nous conviendrons de centrer et réduire lors de la mise en place des modèles.

La variable **vt_dir** est plus dispersée que les autres, mais il s'agit d'une variable numérique circulaire qu'on aura répartie en classe, cela n'a donc pas d'importance. Notons également que la variable **neige** n'a aucune variation : il serait alors inutile de l'inclure dans les modèles s'il est impossible de la prédire.

Consommation observée.





Ces graphiques représentent la consommation horaire moyenne par mois. Nous remarquons qu'en hiver, la consommation d'électricité globalement plus élevée. cela nous a

conduit a créé la variable *f_season* pour illustrer ce fait. Il existe également une tendance journalière, où la consommation atteint un pic à 22heures.

Mise en place du modèle

Cette partie explique la démarche de prédiction que nous avons appliquée lors de ce projet. Nous avons choisi de la présenter sous forme de questions réponses.

Comment prédire une maille horaire en ayant des données météo en maille tri-horaire ?

Nous avons divisé la pression en 3 vecteurs :

- P0 contient la consommation aux heures H, ayant la météo donnée.
- P1 contient la consommation aux heures H + 1.
- P2 contient la consommation aux heures H + 2

Nous avons calculé 2 autres vecteurs : * DP1 = P1 - P0 (écart entre les vecteurs P0 et P1) * DP2 = P2 - P0 (écart entre les vecteurs P0 et P2)

Nous effectuerons 3 prédictions, sur les vecteurs P0, DP1 et DP2.

```
# consoDf : contains consommation :
#   - puissance P0 for each 3 hours.
#   - puissance variation for each 3 hours + 1 (DP1 = P1 - P0)
#   - puissance variation for each 3 hours + 2 (DP1 = P1 - P0)

consoDf <- conso_train %>%
    dplyr::select(datetime, puissance) %>%
    ## on détermine les heures H, H+1 et H+2
    dplyr::mutate(
        R3 = hour(datetime) %% 3,
        vpuiss = factor(R3, levels = 0:2, labels = paste0("P", 0:2)),
        datetime = datetime - 3600 * R3,
        R3 = NULL
    ) %>%
    ## on distribue les puissances en 3 colonnes P0, P1 et P2
    tidyr::spread(vpuiss, puissance) %>%
    dplyr::filter(!is.na(P0)) %>%
    dplyr::mutate(
        DP1 = P1 - P0,
        DP2 = P2 - P0
    )

str(consoDf)
```

```
## 'data.frame':    2920 obs. of  6 variables:
##  $ datetime: POSIXct, format: "2015-09-13 03:00:00" "2015-09-13 06:00:00"
...
```

```
##  $ P0      : num  446 352 556 748 566 ...
##  $ P1      : num  366 375 684 634 508 ...
##  $ P2      : num  341 459 745 615 458 ...
##  $ DP1     : num  -80.3 22.5 128.7 -113.5 -58.5 ...
##  $ DP2     : num  -105 107 189 -133 -108 ...
```

Comment gérer le changement d'heure ?

Les données sont effectivement impactées par le changement d'heure, les derniers dimanches des mois d'octobre et mars.

En octobre, on recule d'une heure, mais dans les données, il n'y a pas deux lignes à 2h du matin. Avec les données à disposition, ce n'est donc pas un problème.

En mars, on avance d'une heure. Il y a donc une valeur de moins, puisqu'il n'y a pas de ligne à 2h du matin. Par conséquent, le modèle H+2 aura donc une ligne de moins.

Quelles variables ont été retenues dans le modèle final ?

Après de nombreux essais, notre modèle final s'est révélé comme le suivant :

- **Month** (disjonctif, 12 colonnes)
- **Weekday** (disjonctif, 7 colonnes)
- **Hour**
- **temp** (centrée réduite)
- **pression** (centrée réduite)
- **hr** (centrée réduite)
- **p_rosee** (centrée réduite)
- **visi** (centrée réduite)
- **vt_raf** (centrée réduite)
- **rr_3h** (centrée réduite)
- **nebul** (centrée réduite)
- **rose_vt** (disjonctif, 4 colonnes)
- **f_season** (disjonctif, 2 colonnes)
- **f_weekend** (déjà considérée comme disjonctive)

Cela donne 14 variables, débouchant sur 35 colonnes.

La variable **neige** n'avait aucun interêt car aucune variation dans les données d'apprentissage, et **vt_moy** est corrélée à **vt_raf**.

```
scenario <- list(
    factors = c("Month", "Weekday", "rose_vt", "f_season"),
    numerics = c("temp", "pression", "hr", "p_rosee",
                 "visi", "vt_raf", "rr_3h", "nebul" ),
    others = c('Hour', "f_weekend")
)
```

Comment les variables ont-elles été pré-traitées ?

Les valeurs manquantes ont été remplacées par la moyenne par mois et par heure (toutes les 3heures, donc), ce afin de diminuer le biais de l'étude. En effet, la température n'est pas la même le 1er janvier à midi qu'à minuit, et encore moins que le 1er juillet, par exemple.

Nous avons également centré réduit en fonction du mois et de l'heure. Nous avions également essayé de centrer et réduire sur l'ensemble des données, mais cela semblait moins efficace au vu du critère MAPE.

Les facteurs ont quant à eux été transformés en variables disjonctives (1 / 0) pour chaque modalité.

Comment a été determiné le "meilleur modèle" ?

Nous avons réalisé nos tests en local. Pour cela :

```
1. Nous avons décomposé notre jeu d'apprentissage en deux sous-jeux (79% en s
ous-apprentissage et 21% en sous-test).
2. Nous avons traités les variables comme vu précédemment,
3. Nous entrainons trois modèles par une méthode de Machine Learning aux para
mètres identiques sur P0, DP1 et DP2 à partir du sous-jeu d'apprentissage.
4. Nous obtenons les prédictions du sous-jeu de test pour 3 modèles, que nous
rassemblons et comparons au sous-jeu de test (critère MAPE).
5. Nous réalisons ces étapes 70 fois, et comparons le MAPE moyen de chaque mo
dèle, et prenons le modèle ayant le critère minimum.
```

Nous avons réalisé ces étapes :

- Pour choisir les variables, sur une même méthode
- Pour chosir la méthode de machine learning, ainsi que les paramètres. Il s'est rapidement avéré que XGBoost donne des meilleurs résultats, avec pour paramètres :
  - nrounds = 300,
  - objective = "reg:linear",
  - booster = "gbtree",
  - eta = 0.01,
  - max_depth = 15,
  - min_child_weight = 5,
  - subsample = 1,
  - normalize_type = 'forest'

Le MAPE moyen minimal sur l'ensemble des méthodes est obtenu avec XGboost, de **8.16093**, contre plus de **9** pour RandomForest.

```
## On requitisionne
data_train <- consoDf %>%
    dplyr::right_join(
        meteo_train[c("datetime", unlist(scenario))],
        by = c("datetime")
```

```
    ) %>%
    dplyr::arrange(datetime) %>%
    dplyr::filter(!is.na(P0))

data_prev <- meteo_prev[c("datetime", unlist(scenario))]

## initialiser les criteres
res_mape <- numeric()
res_rmse <- numeric()

# Modifier la valeur de N par un grand nombre (nous avons pris 70).
N <- 1

for(i in 1:N){
    # spliter a 79%
    list[iTrain, iTest] <- train_test_split(data_train, 0.79)
    ## on conserve une sauvegarde de la table de données
    # remplacer les NA + centrer reduire
    list[iTrain, iTest, varnums] <- deal_train_test_numerics(
        iTrain, iTest,
        variables = scenario$numerics,
        by = c('Month', 'Hour'),
        na_replace = TRUE,
        center = TRUE,
        scale = TRUE
    )
    # transformer facteurs en variables disjonctives
    list[iTrain, iTest, varfacts] <- deal_train_test_factors(
        train = iTrain,
        test = iTest,
        variables = scenario$factors
    )
    ##
    columns <- c(varnums, varfacts, scenario$others)
    Ytrain <- iTrain[c('P0', 'DP1', 'DP2')]
    Xtrain <- as.matrix(iTrain[columns])
    Xtest <- as.matrix(iTest[columns])

    # Realiser xgboost prediction
    xgb_list <- xgboost_predict(
        Ytrain, Xtrain, Xtest,
        nrounds = 300,
        objective = "reg:linear",
        eta = 0.01,
        max_depth = 15,
        min_child_weight = 5,
        subsample = 1,
        booster = "gbtree",
        normalize_type = 'forest',
        verbose = 0
```

```
    )
    puiss_prev <- xgb_list$predictions

    ipredP0 <- puiss_prev$P0
    ipredP1 <- ipredP0 + puiss_prev$DP1
    ipredP2 <- ipredP0 + puiss_prev$DP2
    iPred <- as.vector(t(cbind(ipredP0, ipredP1, ipredP2)))
    iReal <- as.vector(t(iTest[,c('P0', 'P1', 'P2')]))

    ## Evaluation
    mape <- MAPE(iPred, iReal) * 100
    rmse <- RMSE(iPred, iReal)

    res_mape <- c(res_mape, mape)
    res_rmse <- c(res_rmse, rmse)
}
summary(res_mape)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    8.52    8.52    8.52    8.52    8.52    8.52

summary(res_rmse)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   93.24   93.24   93.24   93.24   93.24   93.24
```

Comment a été determinée la “meilleure prédiction” ?

Une fois le modèle optimal obtenu, nous avons tout simplement réalisé 3 modèles d’apprentissage sur P0, DP1 et DP2 à partir des variables conservées.

Nous avons ainsi prédis nos vecteurs P0, DP1 et DP2 finaux, que nous avons réordonné par ordre chronologique.

```
## Scenario Final
scenario <- list(
    factors = c("Month", "Weekday", "rose_vt", "f_season"),
    numerics = c("temp", "pression", "hr", "p_rosee",
                 "visi", "vt_raf", "rr_3h", "nebul" ),
    others = c('Hour', "f_weekend")
)

## On coonstitue notre table de données
data_train <- consoDf %>%
    dplyr::right_join(
        meteo_train[c("datetime", unlist(scenario))],
        by = c("datetime")
    ) %>%
    dplyr::arrange(datetime) %>%
    dplyr::filter(!is.na(P0))
```

```r
data_prev <- meteo_prev[c("datetime", unlist(scenario))]

## center and scale
list[trainDf, prevDf, varnums] <- deal_train_test_numerics(
    data_train,
    data_prev,
    variables = scenario$numerics,
    by = c('Month', 'Hour'),
    na_replace = TRUE,
    center = TRUE,
    scale = TRUE
)

## variable disjonctives
list[trainDf, prevDf, varfacts] <- deal_train_test_factors(
    train = trainDf,
    test = prevDf,
    variables = scenario$factors
)

columns <- c(varnums, varfacts, scenario$others)

# Xgboost
Ytrain <- trainDf[c('P0', 'DP1', 'DP2')]

Xtrain <- trainDf[columns]
Xprev <- prevDf[columns]

puiss_prev <- xgboost_predict(
    Ytrain, Xtrain, Xprev,
    nrounds = 300,
    objective = "reg:linear",
    eta = 0.01,
    max_depth = 15,
    min_child_weight = 5,
    subsample = 1,
    booster = "gbtree",
    normalize_type = 'forest',
    verbose = 0
)$predictions

predP0 <- puiss_prev$P0
predP1 <- predP0 + puiss_prev$DP1
predP2 <- predP0 + puiss_prev$DP2
final_prediction <- as.vector(t(cbind(predP0, predP1, predP2)))

print(final_prediction)
```

```
##   [1] 489.1234 427.9543 373.9571 380.0971 374.2006 290.6196 308.9800
##   [8] 372.5511 455.4619 481.2918 513.4369 526.1346 556.5893 545.7565
##  [15] 577.2168 553.4160 476.1950 424.5484 495.8026 546.5446 547.8661
##  [22] 515.1140 633.1644 563.8554 467.5431 403.6330 372.0957 381.4951
##  [29] 368.6286 315.5099 313.7007 371.1263 450.4241 487.9641 498.6702
##  [36] 496.4560 554.3687 545.1304 564.6949 497.8402 440.4768 416.8531
##  [43] 453.7831 500.1417 483.1337 520.7943 657.6952 636.5801 523.3286
##  [50] 463.9845 386.9604 379.9799 367.3252 306.4080 316.9112 369.1239
##  [57] 455.4804 557.7831 573.8608 612.8580 588.2965 590.3091 546.2731
##  [64] 548.8488 506.6368 451.5522 500.2021 552.2325 546.4001 520.3088
##  [71] 645.3256 588.3449 461.1835 392.4093 357.3481 380.2921 369.7368
##  [78] 301.1916 315.4198 378.4001 435.4166 512.5323 533.0252 560.9723
##  [85] 534.0498 498.7172 464.7363 491.7502 455.0356 475.3643 450.5661
##  [92] 493.0123 496.7802 517.1208 647.2751 650.6038 541.3394 444.8388
##  [99] 397.2155 381.5140 362.1988 326.6553 323.4003 373.9228 439.7376
## [106] 513.9564 550.6764 543.3050 548.2073 513.0449 503.1854 495.6083
## [113] 454.3998 454.4971 456.9756 498.9441 502.0436 498.0202 630.3898
## [120] 618.5062 542.0450 457.2415 392.6947 382.3639 367.0236 304.7165
## [127] 314.3475 340.0088 403.8457 509.3997 578.3331 591.4126 550.5626
## [134] 498.6101 474.0209 478.2731 440.4358 427.9513 431.2154 484.1924
## [141] 454.7923 493.1051 630.3178 622.2843 511.7504 452.6250 373.3108
## [148] 371.2773 359.5381 300.0260 300.8727 356.0481 429.1925 501.0924
## [155] 531.3387 522.7993 538.2274 537.9882 510.7463 510.8856 448.4471
## [162] 387.7142 441.1818 477.1065 494.3093 507.8433 626.7573 618.7329
## [169] 450.7289 395.2796 352.4610 373.6449 370.8573 298.7507 305.0106
## [176] 367.1634 440.1064 501.9814 517.1515 538.4091 523.8375 527.3861
## [183] 501.2747 484.8274 423.2821 408.1946 459.0724 500.3463 509.2607
## [190] 515.2286 651.0857 622.4013
```

Bilan

Ce challenge nous a permis de mettre nos compétences en application dans un contexte compétitif, ce qui est très enrichissant sur les plans personnels comme professionnels lorsqu'on a pas l'habitude des challenges : il nous a permis de choisir des stratégies de machine learning, par exemple, quand la variable à prédire n'est pas de même échelle que les données explicatives.

Nous tenons à remercier le groupe des Jeunes Statisticien(ne)s d'avoir organisé ce challenges, Mme Valérie Robert d'avoir répondu à nos questions, EDF d'avoir mis leur données à disposition, rendant ce challenge possible, ainsi que tout le personnel ayant participé à l'organisation.