regis\_r\_bootcamp

regis andreoli

1/31/2022

Table of Contents

knitr::opts\_chunk$set(echo=TRUE,message=FALSE,error=FALSE,warning=FALSE)

# load library

library(knitr)  
library(rstudioapi)  
library(ggplot2)  
library(gridExtra)  
library(dplyr)  
library(caret)  
library(GGally)  
library(mgcv)  
library(lubridate)

# Load Data

setwd(dirname(getActiveDocumentContext()$path))  
df\_weather <- read.csv("./data/weather.csv",header=TRUE,sep =",",comment.char ="#")  
df\_plantA <- read.csv("./data/A.csv",header=TRUE,sep =",",comment.char ="#")  
df\_plantB <- read.csv("./data/B.csv",header=TRUE,sep =",",comment.char ="#")  
df\_plantC <- read.csv("./data/C.csv",header=TRUE,sep =",",comment.char ="#")

# Data preparation

df\_weather$local\_time <- as.POSIXct(df\_weather$local\_time,tz="GMT",format="%Y-%m-%d %H:%M")  
df\_plantA$Timestamp <- as.POSIXct(df\_plantA$Timestamp,tz="GMT",format="%Y-%m-%d %H:%M:%S")  
df\_plantB$Timestamp <- as.POSIXct(df\_plantB$Timestamp,tz="GMT",format="%Y-%m-%d %H:%M:%S")  
df\_plantC$Timestamp <- as.POSIXct(df\_plantC$Timestamp,tz="GMT",format="%Y-%m-%d %H:%M:%S")

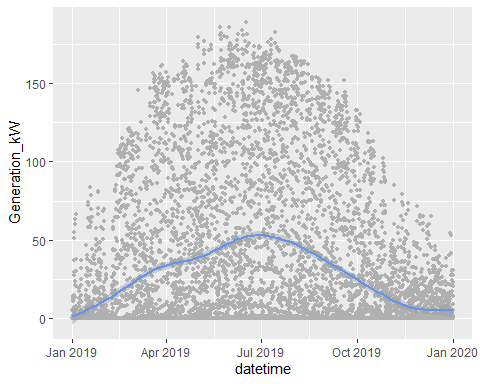
df\_plantA\_resample <- df\_plantA %>%  
 mutate(datetime = floor\_date(Timestamp, "1 hour")) %>%  
 group\_by(datetime) %>%  
 summarise(across(Generation\_kW:Overall\_Consumption\_Calc\_kW, sum))  
  
str(df\_plantA\_resample)

## tibble [8,760 x 5] (S3: tbl\_df/tbl/data.frame)  
## $ datetime : POSIXct[1:8760], format: "2019-01-01 00:00:00" "2019-01-01 01:00:00" ...  
## $ Generation\_kW : num [1:8760] 0 0 0 0 0 ...  
## $ Grid\_Feed.In\_kW : num [1:8760] 0 0 0 0 0 0 0 0 0 0 ...  
## $ Grid\_Supply\_kW : num [1:8760] 16.9 16.8 17.5 16.9 17.5 ...  
## $ Overall\_Consumption\_Calc\_kW: num [1:8760] 16.9 16.8 17.5 16.9 17.5 ...

# Visual analysis

First visual analysis of the data. The Graph is Supported by a GAM smoother.

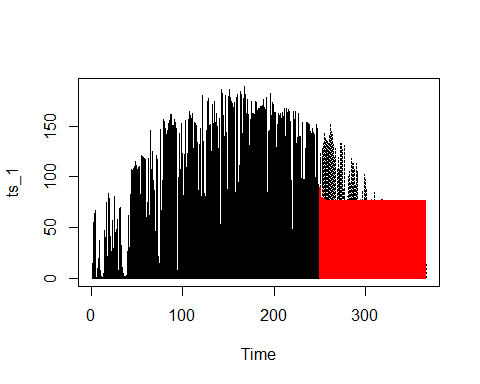
ggplot(data = df\_plantA\_resample,  
 mapping = aes(y = Generation\_kW, x = datetime)) +  
 geom\_point(size = 1, color = "grey69") +  
 geom\_smooth(method = "gam", color = "cornflowerblue")



# Time Series Analysis - with auto.arima

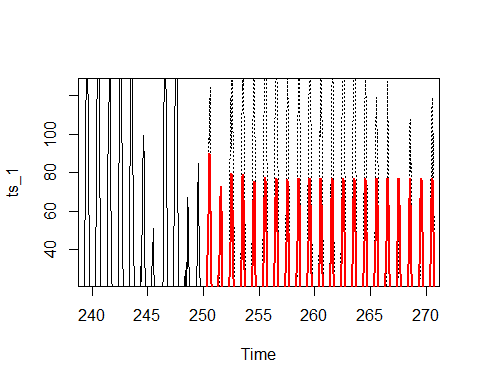
Convert data into time series object. As the data has an hourly resolution, the time interval is set to 24, which correspond in this case to a seasonality of one day.

library(forecast)  
  
ts\_1 <- ts((df\_plantA\_resample$Generation\_kW), deltat = 1/24)  
train <- window(ts\_1, start = 1, end = 250)  
fit <- auto.arima(train)

Plot the time series with the prediction. 

Plot the time series again, but with a focus on the prediction area. As it is to see, the model prediction has some noticable, small variation in the first five days. Then, it converge to value which is a bit

fc <- predict(fit, n.ahead = 115\*24)  
plot(ts\_1, lty=3, cex = 0.1, xlim=c(240, 270), ylim=c(25, 125))  
lines(train, lwd=1)  
lines(fc$pred, lwd=2, col="red", cex = 0.1)

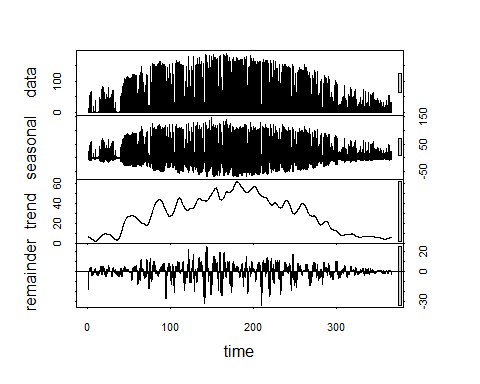


mean(df\_plantA\_resample$Generation\_kW[df\_plantA\_resample$Generation\_kW > 0])

## [1] 53.46822

Plot the decomposition of the time series with the function stl().

decomp<-stl(ts\_1, s.window = 1/24, t.window = 365)  
plot(decomp)



# Second Time Series Analysis - Manipulated Data

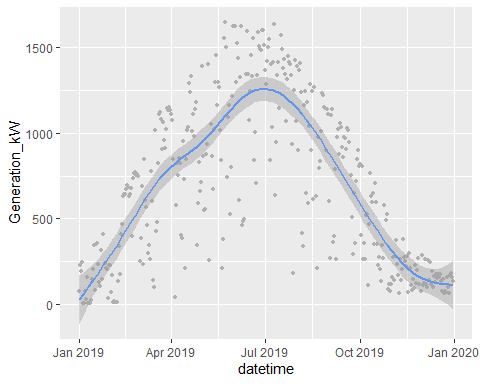
Some artificial years are added to the time series, to be able to capture the seasonal effect over the entire year.

## Data preparation

df\_plantA\_resample\_2 <- df\_plantA %>%  
 mutate(datetime = floor\_date(Timestamp, "24 hour")) %>%  
 group\_by(datetime) %>%  
 summarise(across(Generation\_kW:Overall\_Consumption\_Calc\_kW, sum))  
  
ts\_2 <- ts((df\_plantA\_resample\_2$Generation\_kW), start = c(2019), deltat = 1/365)

Plot time series of one year with a gam smoother.

ggplot(data = df\_plantA\_resample\_2,  
 mapping = aes(y = Generation\_kW, x = datetime)) +  
 geom\_point(size = 1, color = "grey69") +  
 geom\_smooth(method = "gam", color = "cornflowerblue")



Add random noise to first time series, in order to generate new artificial years.

ts\_artif\_1 <- {jitter(ts\_2, factor=500, amount = NULL)}  
ts\_artif\_2 <- {jitter(ts\_2, factor=500, amount = NULL)}  
head(ts\_2)

## Time Series:  
## Start = c(2019, 1)   
## End = c(2019, 6)   
## Frequency = 365   
## [1] 71.260 226.412 193.904 242.044 1.248 4.432

head(ts\_artif\_1)

## Time Series:  
## Start = c(2019, 1)   
## End = c(2019, 6)   
## Frequency = 365   
## [1] 133.42868 250.83120 293.33971 190.10344 -33.31234 -51.81625

head(ts\_artif\_2)

## Time Series:  
## Start = c(2019, 1)   
## End = c(2019, 6)   
## Frequency = 365   
## [1] 32.523435 224.625020 114.419124 296.626727 5.390415 -64.041309

Set negative values to zero, as due to the nature of the data, negative values cannot occur.

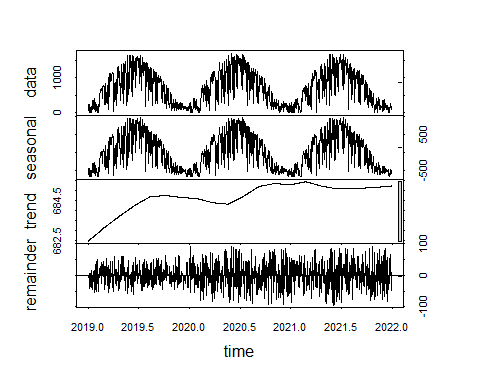
ts\_artif\_1[][ts\_artif\_1[] < 0] <- 0  
ts\_artif\_2[][ts\_artif\_2[] < 0] <- 0

Create time series of several years, starting with real data year and followed by two artificial years. A TS decomposition is then plotted via the function stl().

ts\_artificial <- ts(c(ts\_2, ts\_artif\_1, ts\_artif\_2), start = c(2019), deltat = 1/365)  
decomp<-stl(ts\_artificial, s.window = 365)  
str(ts\_artificial)

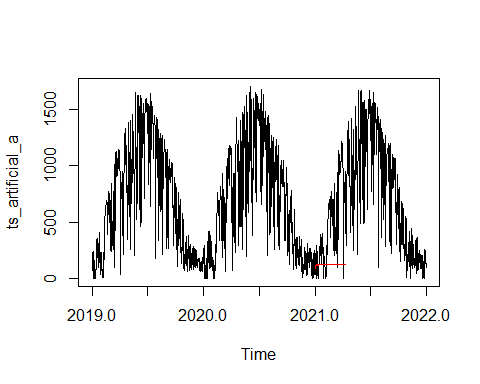
## Time-Series [1:1095] from 2019 to 2022: 71.26 226.41 193.9 242.04 1.25 ...

plot(decomp)



Train model with the first two years of the time series. To train the model, the auto.arima function is made use of.

ts\_artificial\_a <- ts(c(ts\_2, ts\_artif\_1, ts\_artif\_2), start = c(2019), frequency = 365)  
  
train\_2 <- window(ts\_artificial\_a, start = c(2019,1), end = c(2020,365))  
fit\_2 <- auto.arima(train\_2)  
arima\_prediciton <- predict(fit\_2, n.ahead = 100)  
  
plot(ts\_artificial\_a, cex = 0.1)  
lines(train\_2)  
lines(arima\_prediciton$pred, col = "red")



# Artificial time series - new attemp: use of gam smoother as base for an artificial year and add normally distributed noise.

Create new data frame with indexed data. This is done to avoid date handling

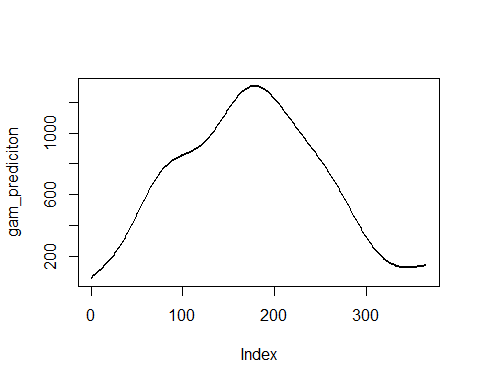
df\_artif <- data.frame(time = (1:365), generation\_kw = df\_plantA\_resample\_2$Generation\_kW)

Compute a smoother line function with the new data frame

gam\_model <- gam(generation\_kw ~ s(time), data = df\_artif)

Get the discrete steps from the smoother line by computing prediction in the desired step interval. The result is plotted to visually control the results. As shown below, the data frame plot is, at least qualitatively, very similar to the ggplot gam smoother line. This result will now be taken as smoothed year.

df\_time <- data.frame(time = c(1:365))  
gam\_prediciton <- predict(gam\_model, newdata = df\_time)  
plot(gam\_prediciton, cex = 0.1, )



After the creation of an artificial “smoothed” year, noise is added. In order to keep the heteroskedastic behavior of the real data, the noise is added as a multiplication of a random coefficient. After trying around, the random coefficient are created with values between 0.2 to 1. A quick look at the plot show a satisfying result.

Add noise to smoothed base year

library(BBmisc)  
library(fGarch)  
set.seed(4)  
random\_coef <- rsnorm(365, mean = 1, sd = 1, xi = 0.1)  
ran\_coef\_norm <- normalize(random\_coef, method = "range", range = c(0, 1))  
#ran\_coef\_norm <- ran\_coef\_norm \* 1.3  
ran\_coef\_norm\_2 <- ran\_coef\_norm \* runif(ran\_coef\_norm, min = 0.3, max = 1.6)  
gam\_yr\_with\_noise <- gam\_prediciton \* ran\_coef\_norm\_2  
  
par(mfrow=c(2,2))  
  
print("df\_plantA\_resample\_2$Generation\_kW")

## [1] "df\_plantA\_resample\_2$Generation\_kW"

mean(df\_plantA\_resample\_2$Generation\_kW)

## [1] 684.2468

median(df\_plantA\_resample\_2$Generation\_kW)

## [1] 643.164

hist(df\_plantA\_resample\_2$Generation\_kW, breaks = 11)  
  
print("")

## [1] ""

print("gam\_prediciton")

## [1] "gam\_prediciton"

mean(gam\_prediciton)

## [1] 684.2468

median(gam\_prediciton)

## [1] 746.8579

hist(gam\_prediciton, breaks = 11)  
  
print("")

## [1] ""

print("gam\_yr\_with\_noise")

## [1] "gam\_yr\_with\_noise"

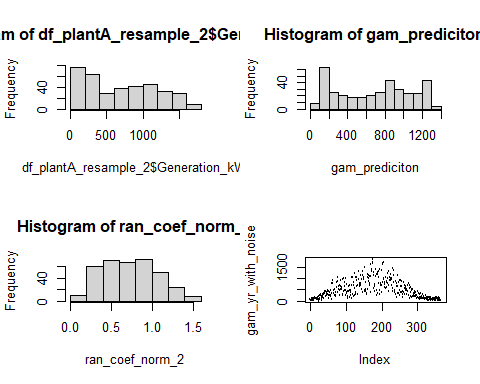
hist(ran\_coef\_norm\_2)  
mean(gam\_yr\_with\_noise)

## [1] 504.5858

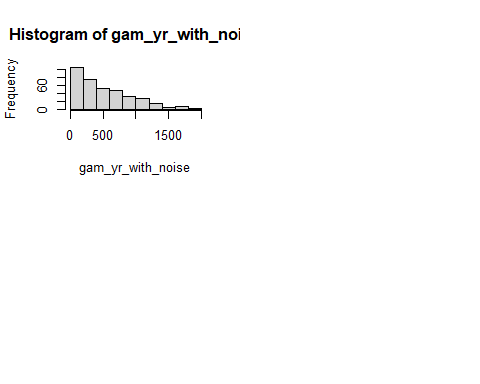
median(gam\_yr\_with\_noise)

## [1] 402.1255

plot(gam\_yr\_with\_noise, cex = 0.2)



hist(gam\_yr\_with\_noise, breaks = 11)



While the artificial year visually looks satisfying, the mean generally is too low. It is not easy to change the noise parameters in such a way, that the variance and the mean of the artificial year gets near a similar value of the real data. Therefore, a function is create which aims to find the best parameter setting:

First, a list of parameter sets is created. This list is then used in a function that loops through the list and creates for each set a noised year. The noised year is then compared with the real year by its mean and variance. A threshold for mean and variance is set. As soon as the thresholds are cumulatively fulfilled, the function stops and passed the noised year as an output of the function.

#create list of parameter sets  
library(BBmisc)  
xi <- c(25:1)/30  
min <- c(1:50)/100  
max <- c(60:90)/40  
u <- list()  
 for (i in xi) {  
 for (n in min) {  
 for (q in max) {  
 o <- c(i, n, q)  
 u <- rbind(u,o)  
 }  
 }  
 }

get\_artificial\_years <- function(mean\_real\_yr, variance\_real\_yr, gam\_prediciton) {  
   
 div\_mean <- 0  
 div\_var <- 0  
 i = 1  
 treshold\_1 <- TRUE  
 treshold\_2 <- TRUE  
 df\_sets <- list()  
 m <- (length(u)/3 - 1)  
   
 for (n in u) {  
 if (i == m) {  
 break  
 }  
 random\_coef <- rsnorm(365, mean = 1, sd = 1, xi = as.numeric(u[i,][1]))  
 ran\_coef\_norm <- normalize(random\_coef, method = "range", range = c(0, 1))  
 ran\_coef\_norm\_2 <- ran\_coef\_norm \* runif(ran\_coef\_norm, min = as.numeric(u[i,][2]), max = as.numeric(u[i,][3]))  
  
 yr\_with\_noise <- gam\_prediciton \* ran\_coef\_norm\_2  
 div\_mean <- (mean\_real\_yr / mean(yr\_with\_noise))  
 div\_var <- (variance\_real\_yr / var(yr\_with\_noise))  
   
 treshold\_1 <- (div\_mean < 1.025 && div\_mean > 0.95)  
 treshold\_2 <- (div\_var < 1.05 && div\_var > 0.95)  
   
 if (treshold\_1 && treshold\_2) {  
 df\_sets <- rbind(df\_sets, yr\_with\_noise)  
 print(i)  
 print(div\_mean)  
 print(div\_var)  
 }  
 if (i %in% (c(1:1000)\*2500)){  
 print(i)  
 }  
   
 i <- i + 1  
 }  
 df\_sets  
}

Call the function to compute about 40’000 simulated years and and save all one that meet the threshold requirements.

## [1] 2500  
## [1] 5000  
## [1] 7500  
## [1] 10000  
## [1] 12500  
## [1] 15000  
## [1] 15498  
## [1] 0.9998943  
## [1] 0.9574415  
## [1] 17500  
## [1] 20000  
## [1] 22500  
## [1] 25000  
## [1] 27500  
## [1] 30000  
## [1] 32421  
## [1] 1.018895  
## [1] 0.9638508  
## [1] 32500  
## [1] 35000  
## [1] 35515  
## [1] 1.022846  
## [1] 0.9602693  
## [1] 35637  
## [1] 1.019101  
## [1] 0.9542123  
## [1] 37500

df\_artif <- t(noised\_year[2,])  
arr <- array(as.numeric(unlist(df\_artif)))  
print("Real")

## [1] "Real"

mean(df\_plantA\_resample\_2$Generation\_kW)

## [1] 684.2468

var(df\_plantA\_resample\_2$Generation\_kW)

## [1] 232351.3

print("Artif")

## [1] "Artif"

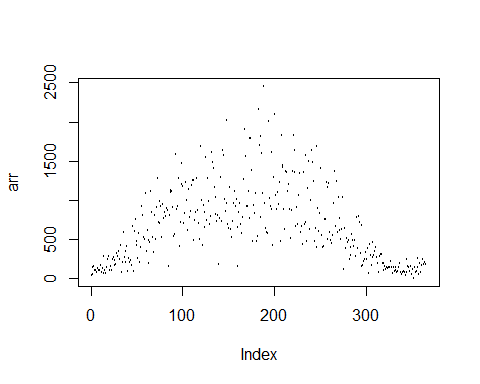
mean(arr)

## [1] 671.5576

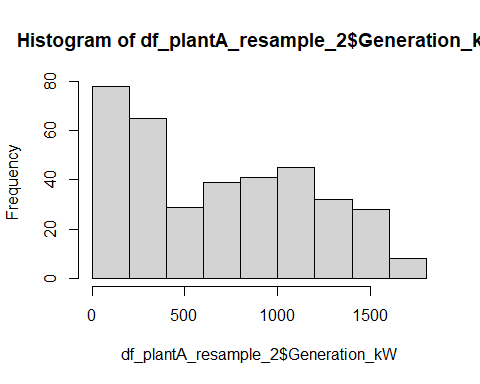
var(arr)

## [1] 241065.6

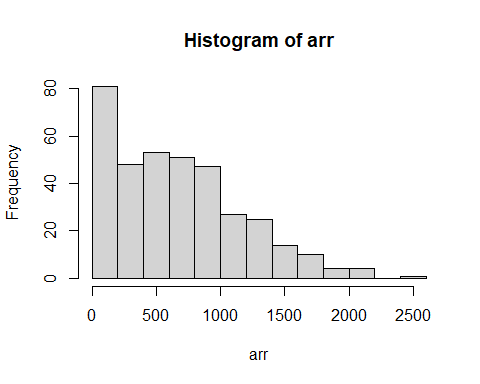
plot(arr, cex = .2)



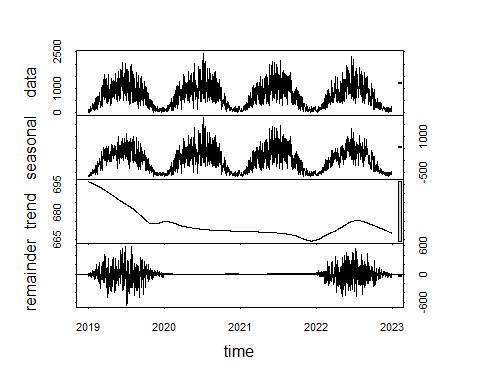
hist(df\_plantA\_resample\_2$Generation\_kW)



hist(arr)



ts\_artificial\_2 <- ts(c(noised\_year[1,], noised\_year[2,], noised\_year[3,], noised\_year[4,]), start = c(2019), deltat = 1/365)  
decomp\_2 <- stl(ts\_artificial\_2, s.window = 1/24, t.window = 365)  
plot(decomp\_2)



#{r} ts\_artificial\_3 <- ts(c(noised\_year[1,], noised\_year[2,], noised\_year[3,], noised\_year[4,], noised\_year[5,]), start = c(2019), deltat = 1/365) train\_2 <- window(ts\_artificial\_3, start = 2019, end = 2024) fit <- auto.arima(train\_2)

Plot the time series with the prediction. #{r} fc <- predict(fit, n.ahead = 115\*24) plot(ts\_1, lty=3, size = 0.1) lines(train, lwd=1) lines(fc$pred, lwd=2, col="red", size = 0.01) #lines(fc$pred+fc$se\*1.96, col="red") #lines(fc$pred-fc$se\*1.96, col="red")