Groupwork R Bootcamp

Felix Bigler, Régis Andréoli

2/23/2022

Table of Contents

# Introudction

In this group work for the R bootcamp module, we would like to analyze a photovoltaic system dataset. The dataset was created during the operation of AEW-owned PV plants in 2019. In total there are 3 different plants but we will only concentrate on one (Plant A). The plants are all located in Aargau, Switzerland. The power values in kW refer to the average over the 15min period. The data was published for the Energy Data Hackdays 2020 in Brugg.

We then merge this dataset with weather data from 2019 in order to evaluate the influence of the weather as well as be able to build a simple prediction model. Moreover we would like to create a simple battery charging algorithm in order to store surplus energy.

# Data Preperation

## Load Data

# set directory read 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 =",")

## Inspect Data Structure

str(df\_weather)  
str(df\_plantA)

We got 8760 observations for the weather dataset and 35040 observations for each PV plant Dataset. Which makes sense since we got weather data for every hour for the year 2019 and on the other hand, operational data of the PV plants for every 15 min in kW. The values are all of the type num except the datetime entries. These are of type char. In a next step we convert the datetime entries to a datetime object to have the possibility to work with the datetime entries. There are no missing values in both datasets.

## Convert datetime entries from char to datetime object

Since all datetime are from the type char we convert the local time to a datetime object.

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")

## Resample PV plant Timeseries from 15min to Hourly Intervalls

Since the PV plant entries are 15 minute observations and the weather data is hourly we resample the PV plant dataset by grouping every 15 min observation to its corresponding hour and take the sum of it. We got then directly the energy gained in kWh instead of the 15 min PV power in kW which is more convenient.

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

## Merge Datasets

In this step we merge both datasets with an inner\_join by the entry datetime. For that we have to rename the column local\_time to datetime in the weather dataset.

df\_weather <- df\_weather %>%   
 rename(  
 datetime = local\_time,  
 )

df\_A\_joined <- inner\_join(df\_plantA\_resample, df\_weather, by = "datetime")

## Create Categorical Variable

A prerequisite of the course assignment is that the data set contains at least one categorical variable. Since our data set does not have any categorical variable we create two of them by extracting the month as well as the day out of the date time entries with the package lubridate as follows.

df\_A\_joined <-df\_A\_joined %>%  
 mutate(  
 month = month(datetime),  
 month\_label = month(datetime, label=TRUE),  
 hour = hour(datetime),  
 day = day(datetime),  
 year = year(datetime)  
 )

# Graphical Exploration of Datasets

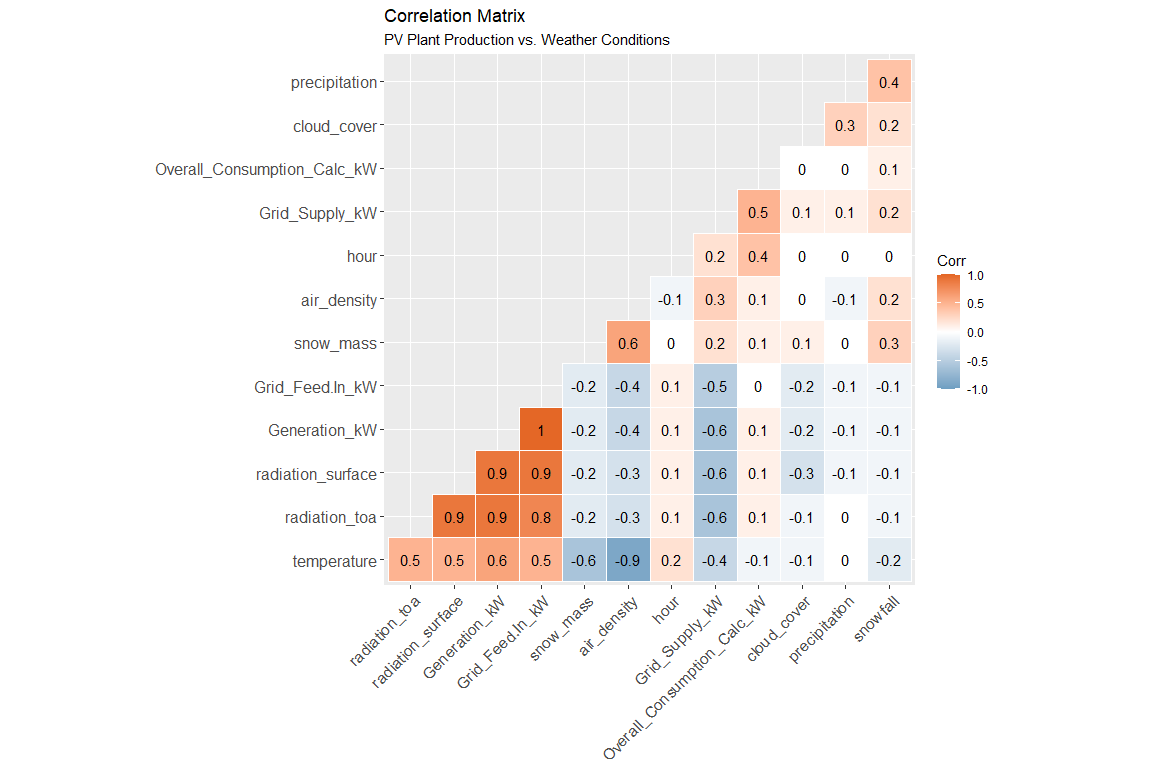
In this chapter we would like to explore the dataset graphically. First we are going create a correlation matrix in order to see which variables correlate with the PV plant production rate. Then we explore the continuous as well as the categorical variables graphically.

## Create Correlation Matrix

knitr::opts\_chunk$set(fig.width=12, fig.height=8)  
df\_corr <- select(df\_A\_joined,-c(datetime,time,month\_label,year,day,month))   
# Exclude datetime entries in order to create correlation matrix  
corr <- round(cor(df\_corr), 1)  
#head(corr[, 1:6])

head(df\_corr)

p1 <- ggcorrplot(corr, hc.order = TRUE, type = "lower",  
 outline.col = "white",  
 ggtheme = ggplot2::theme\_gray,  
 lab = TRUE,  
 colors = c("#6D9EC1", "white", "#E46726")) +   
 labs(title = "Correlation Matrix",  
 subtitle = "PV Plant Production vs. Weather Conditions")  
p1



We can conclude from the correlation matrix, that the variables radiation\_surface (Ground-level solar irradiance (W / m²)) and radiation\_toa (Top of atmosphere solar irradiance (W / m²)) have a strong positive correlation with the PV production rate (variable Generation\_kW). The coefficients are very high with a value of 0.9 which seems obvious, since a solar cell converts solar radiation into electrical energy when sunlight is present on the surface of the cell. The two variables certainly have a strong colinearity since the values differ only in respect to the fact that one value was measured at the ground and the other value above the atmosphere. For the actual solar production, however, only the value at the ground is relevant.

The variable temperature has also a rather high correlation with a value of 0.6. However, this correlation may have more in common with the fact that the temperatures are also higher in the case of strong solar irradiation. In fact, solar modules have a negative temperature coefficient in relation to the efficiency rating (<https://www.pveurope.eu/solar-modules/global-warming-growing-importance-temperature-coefficient-solar-modules>).

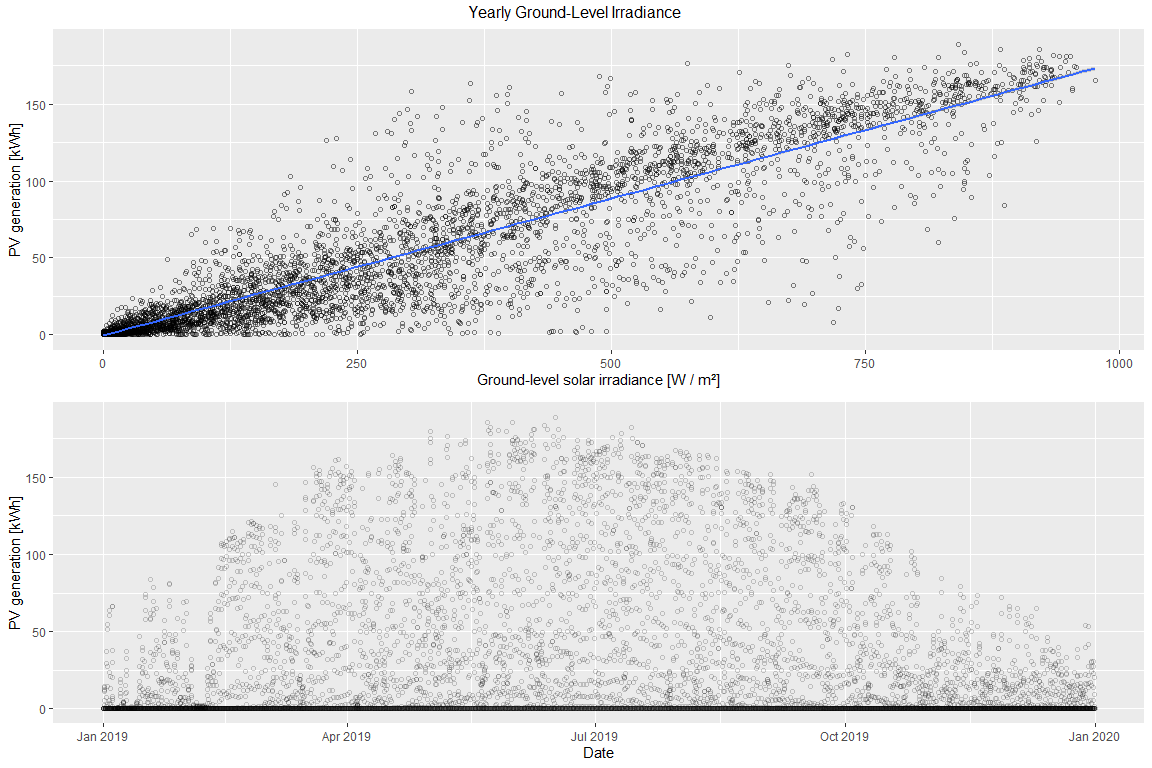
The other weather variables don’t have a significant influence, therefore we concentrate on the variable irradiance in the further analysis of the data set.

## Exploration of Contineous Variables

### Yearly Ground-Level Irradiance

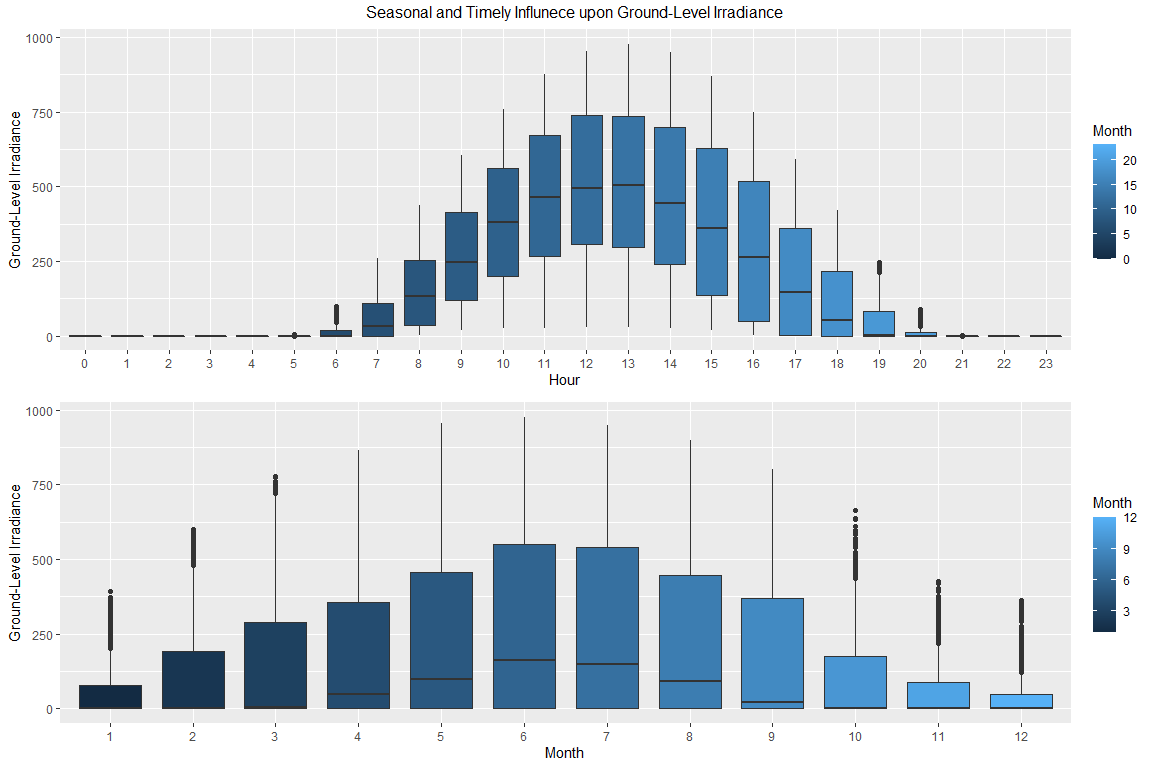
p2 <- ggplot(data = df\_A\_joined,  
 mapping = aes(y = Generation\_kW ,  
 x = radiation\_surface)) +   
 xlab("Ground-level solar irradiance [W / m²]") +   
 ylab("PV generation [kWh]") +   
 geom\_point(alpha = 0.5,shape = 1) +  
 geom\_smooth(method = "lm")   
  
p3 <- ggplot(data = df\_A\_joined,  
 mapping = aes(y = Generation\_kW ,  
 x = datetime)) + xlab("Date") + ylab("PV generation [kWh]") +   
 geom\_point(alpha = 0.2,shape = 1) +  
 labs(colour = "Method")  
  
grid.arrange(p2,p3, nrow=2,top = "Yearly Ground-Level Irradiance")

## `geom\_smooth()` using formula 'y ~ x'

 \* we can conclude from the plot above, that the PV production highly correlates with the gorund level solar irradiance as the correlation plot already explained.The blue line indicates a linear regression line.

## Exploration of Categorical Variables

### Seasonal and Timely Ground-Level Irradiance



* as we have expected, the most solar irradiance occurs between April and September and during the hours from 10am to 3pm as the boxplots above indicates.

## PV Plant Production

library(viridis)

## Warning: package 'viridis' was built under R version 4.0.5

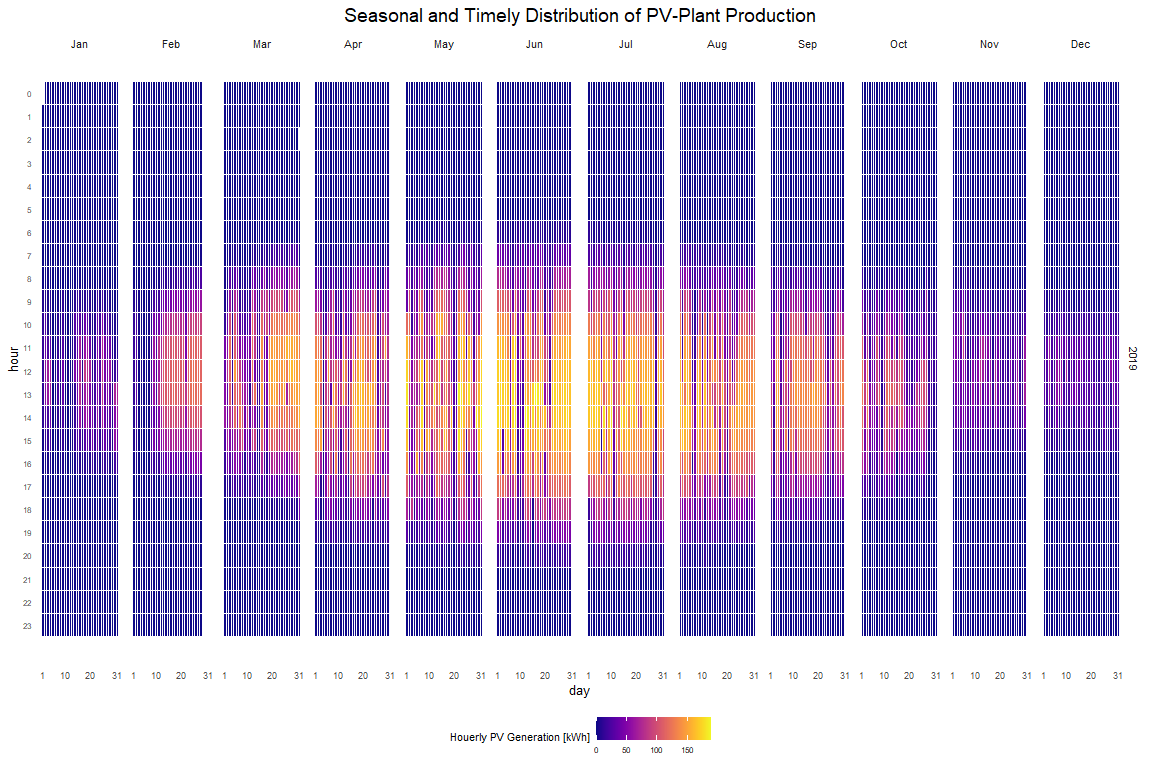
## Loading required package: viridisLite

## Warning: package 'viridisLite' was built under R version 4.0.5

library(ggExtra)

## Warning: package 'ggExtra' was built under R version 4.0.5

p6 <- ggplot(data = df\_A\_joined, aes(x = day,y = hour,fill=Generation\_kW))+  
 geom\_tile(color= "white",size=0.1) +   
 scale\_fill\_viridis(name="Houerly PV Generation [kWh]",option ="C") +   
 facet\_grid(year ~ month\_label) +  
 scale\_y\_continuous(trans = "reverse", breaks = unique(df\_A\_joined$hour)) +  
 scale\_x\_continuous(breaks =c(1,10,20,31)) +  
 theme\_minimal(base\_size = 10)+  
 labs(title= "Seasonal and Timely Distribution of PV-Plant Production")+   
 theme(legend.position = "bottom") +  
 theme(plot.title=element\_text(size = 14)) + theme(axis.text.y=element\_text(size=6)) +  
 theme(strip.background = element\_rect(colour="white")) +  
 theme(plot.title=element\_text(hjust=0)) + theme(axis.ticks=element\_blank()) +  
 theme(axis.text=element\_text(size=7)) + theme(legend.title=element\_text(size=8)) +  
 theme(legend.text=element\_text(size=6)) +theme(plot.title = element\_text(hjust = 0.5)) +  
 removeGrid()  
p6



* the plot above shows the seasonal and timely distribution of the PV production of the plant A.

# Battery Loading Algorithm (Chapter of choice 1)

The main drawback of photovoltaics is that most of the energy production falls into the summer months, which leads to a surplus of energy during this period. This surplus can be stored with a battery, for example. We would like to create a simple battery charging algorithm to simulate a battery in the solar system and evaluate how the self-consumption-rate can be increased by adding a storage.

## Self-Consumption-Rate Without Storage System

Lets first calculate the self-consumption-rate of the system without a battery. We define a function as follows to determine when the produced energy is directly consumed and not feed into the grid:

self\_consumption\_rate <- function(generation,consumption,i){  
 self\_con <- as.data.frame(matrix(1:8759,ncol=1))  
 if ((generation - consumption) >= 0) {  
 self\_con$V1[i] = consumption  
 }else {  
 self\_con$V1[i] = 0  
 }  
}

We create then a new data frame self\_consumption and apply the function with a for loop over the original data frame df\_A\_joined as follows:

self\_consumption <- as.data.frame(matrix(1:8759,ncol=1))  
for(i in 1:nrow(df\_A\_joined)){  
 self\_consumption$V1[i] <- self\_consumption\_rate(df\_A\_joined$Generation\_kW[i],  
 df\_A\_joined$Overall\_Consumption\_Calc\_kW[i],  
 i)  
}

Now we can calculate the self-consumption-rate as follows:

sum(self\_consumption$V1)/sum(df\_A\_joined$Generation\_kW)

## [1] 0.199887

* we can see that only 20 % of the produced energy is used directly, the other part is feed into the grid.

### Create Simple Battery Loading Algorithm

The next step is to create a simple battery loading function. The input variables for the function are battery capacity, max. battery capacity, min. battery capacity, the PV generation as well as the consumption and the initial battery charging state.

battery\_state <- function(battery\_capacity,max\_battery\_capacity,min\_battery\_capacity,pv\_generation, consumption,battery\_state) {  
   
 min\_battery\_load = battery\_capacity \* (min\_battery\_capacity)/100  
 max\_battery\_load = battery\_capacity \* (max\_battery\_capacity)/100  
 battery\_load = battery\_state  
   
 if ((pv\_generation - consumption) > 0 & battery\_load < max\_battery\_load) {  
 if (battery\_load + (pv\_generation - consumption) > max\_battery\_load) {  
 battery\_load = max\_battery\_load   
 } else   
 {battery\_load = battery\_load + (pv\_generation - consumption)}   
 }  
 else if ((pv\_generation-consumption) < 0 & (battery\_load > min\_battery\_load)) {  
 if (battery\_load + (pv\_generation - consumption) < min\_battery\_load) {  
 battery\_load = min\_battery\_load   
 } else {  
 battery\_load = battery\_load + (pv\_generation - consumption) }  
 }  
 return(battery\_load)   
}

We apply now the battery loading algorithm. We us different battery sizes from 45 kWh up to 450 kWh in 45 kWh steps.

max\_battery\_capacity <- 95 # in %  
min\_battery\_capacity <- 15 # in %  
  
battery\_states <- as.data.frame(matrix(0,nrow=8759,ncol=10))  
x <- 1  
  
for (j in seq(45, 450, length.out=10)) {  
 for(i in 2:nrow(df\_A\_joined)) {   
 battery\_states[i,x] <- battery\_state(j,max\_battery\_capacity,min\_battery\_capacity,  
 df\_A\_joined$Generation\_kW[i],  
 df\_A\_joined$Overall\_Consumption\_Calc\_kW[i],  
 battery\_states[i-1,x])  
 }  
 x = x + 1  
}

### Calculate Self-Consumption-Rate for Different Battery Sizes

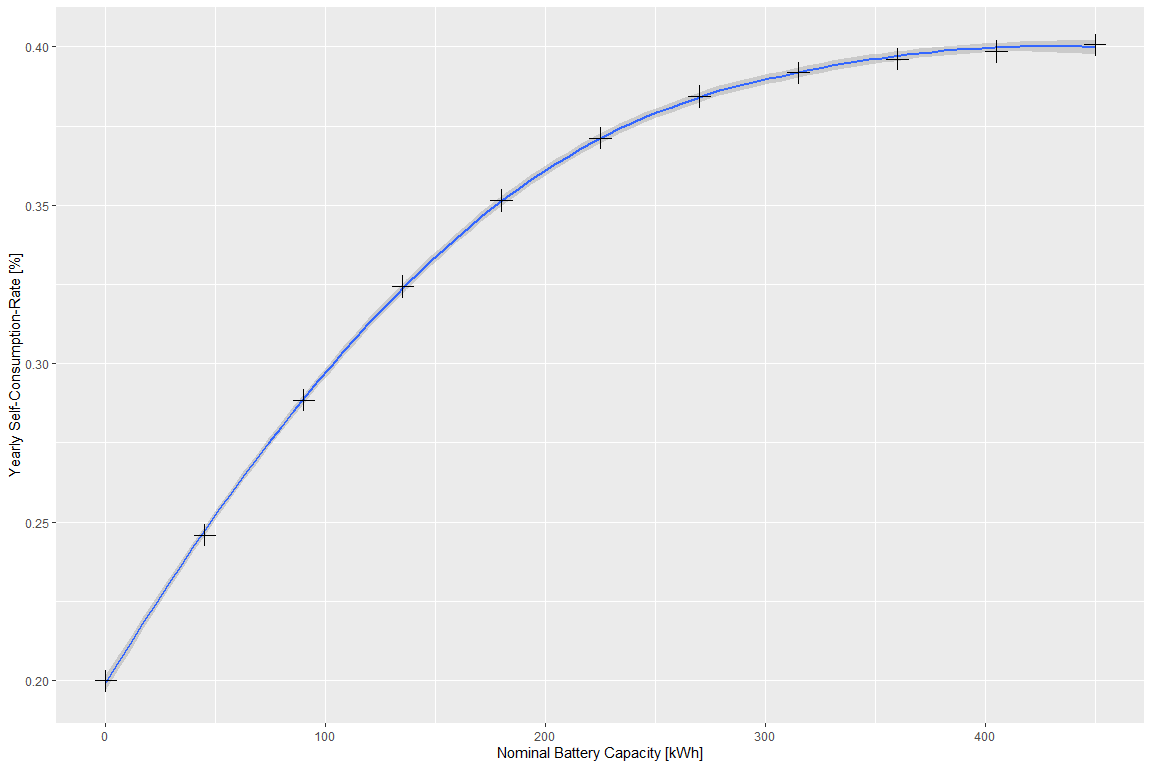
self\_consumption\_battery <- as.data.frame(matrix(0,nrow=8759,ncol=10))  
for(j in seq(1,10,length.out = 10)){  
 for(i in 1:8758) {  
 if ((battery\_states[i+1,j] - battery\_states[i,j]) < 0) {  
 self\_consumption\_battery[i+1,j] <- self\_consumption[i+1,1] + abs(battery\_states[i+1,j] - battery\_states[i,j])  
 } else{self\_consumption\_battery[i+1,j] <- self\_consumption[i+1,1]}  
 }  
}

* The next step is to create a dataframe containing the different self-consumption-ratios and the corresponding battery capacity.

df\_self\_con\_ratio <- as.data.frame(colSums(self\_consumption\_battery)/sum(df\_A\_joined$Generation\_kW))  
  
df\_self\_con\_ratio$battery\_capacity <- seq(45, 450, length.out=10)  
colnames(df\_self\_con\_ratio) <- c("self\_consumption\_ratio\_kWh","battery\_capacity\_kWh")  
  
new\_row <- c(sum(self\_consumption$V1)/sum(df\_A\_joined$Generation\_kW),0)  
df\_self\_con\_ratio <- rbind(new\_row,df\_self\_con\_ratio)

## <ScaleContinuousPosition>  
## Range:   
## Limits: 0 -- 500

## `geom\_smooth()` using method = 'loess' and formula 'y ~ x'



* as we can see from the plot above, the self-consumption-rate starts to flatten out at around a capacity of 220 kWh. This can be explained by the fact, that after a certain point, an increase in battery capacity no longer causes a noticeable increase in the self-consumption rate, since only individual peaks of the PV production are additionally stored.

In the next chapters we try to fit a timeseries model and a GAM model to predict the PV production.

# Predicting PV Prodcution (Chapter of choice 2)

## Time Series Analysis - With auto.arima

The explored model for prediction is a time series model, called SARIMA. To be able to compute those types of models, the data set has first to be converted into a time series object. As the data has an hourly resolution, the time interval of the TS object is set to 24, which correspond, in this case, to the seasonality of one day. The model is then trained with the first 250 days of the year.

library(forecast)

## Warning: package 'forecast' was built under R version 4.0.5

## Registered S3 method overwritten by 'quantmod':  
## method from  
## as.zoo.data.frame zoo

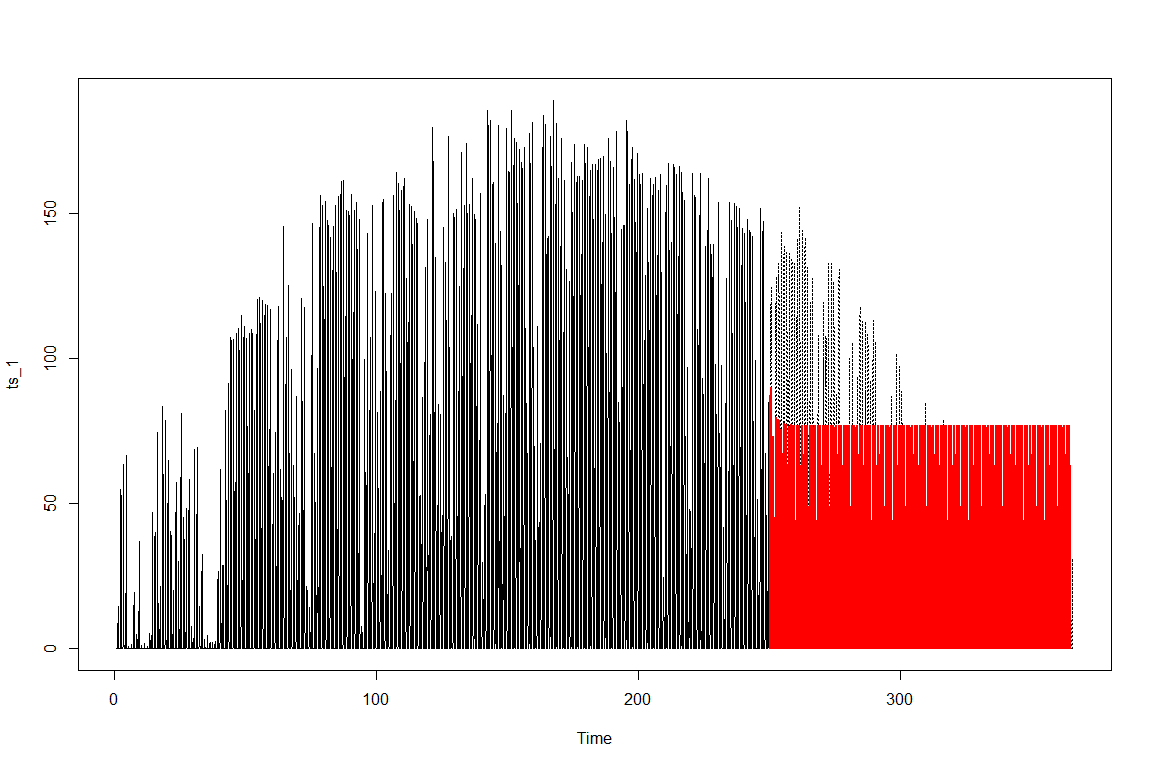
##   
## Attaching package: 'forecast'

## The following object is masked from 'package:nlme':  
##   
## getResponse

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

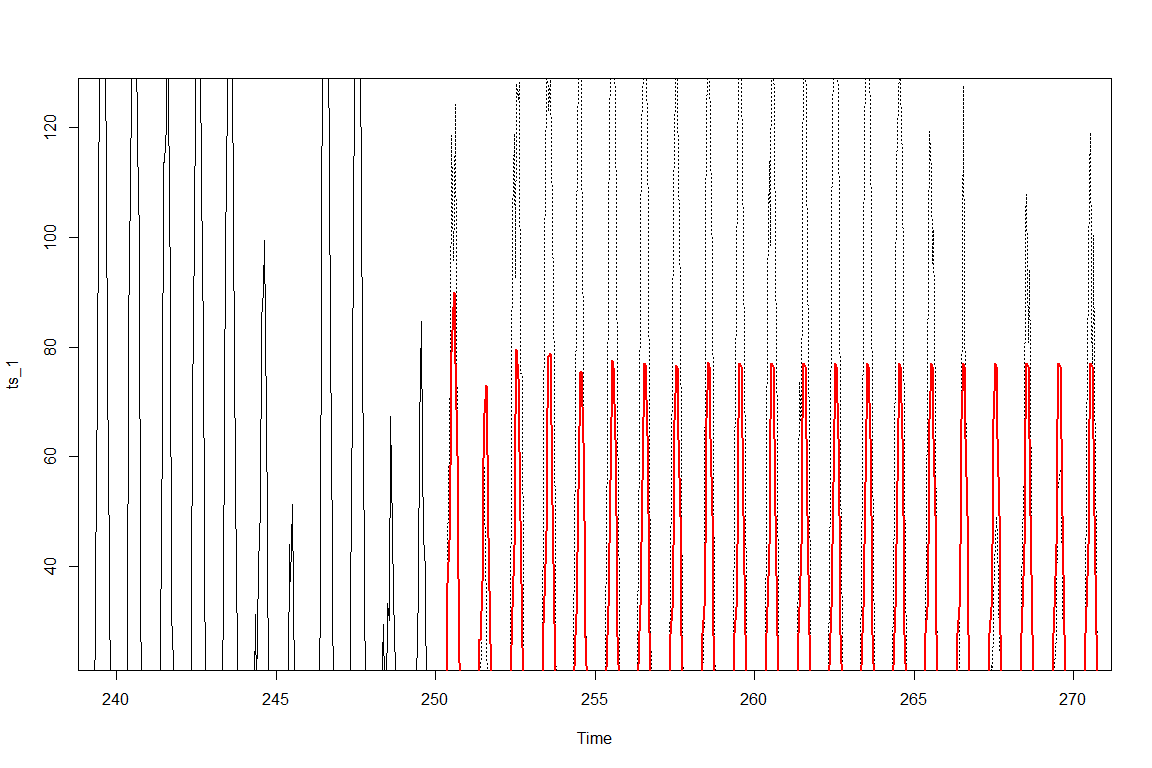
### Model Prediction

The trained model is then used for a prediction. The predicted days are from day 250 to 365. The result can be seen in the following plot. Red are the predicted data, black are the real data.



As one can see above, prediction range of the plot is not very helpful. So, another plot is computed, but this time with a zoom set on the prediction area. As it is to see, the prediction has some variation for the first few days. Then, it converges to value which is a little bit higher than the mean of the data.

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)



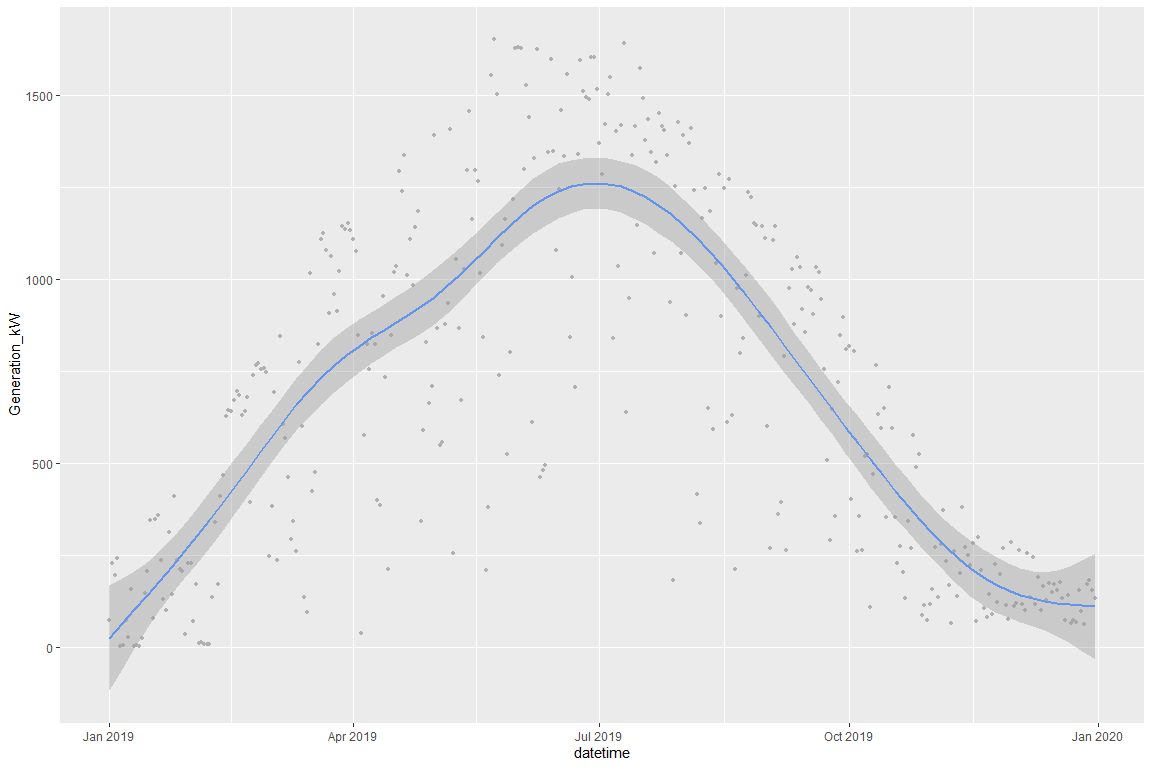
## Time Series Analysis - with Manipulated Data

Because the previous results were not satisfying at all, a few changes to the data set are undertaken. First, the resolution is set to daily. Secondly, some artificial years are generated and added to the time series. This, in order to be able to capture the seasonal effect over the entire year. This entire part is done with an experimental intention and from curiosity. With the resolution set to daily, the data set shrink to 365 entries.

Plot of new time series with a resolution of one day and 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")

## `geom\_smooth()` using formula 'y ~ s(x, bs = "cs")'

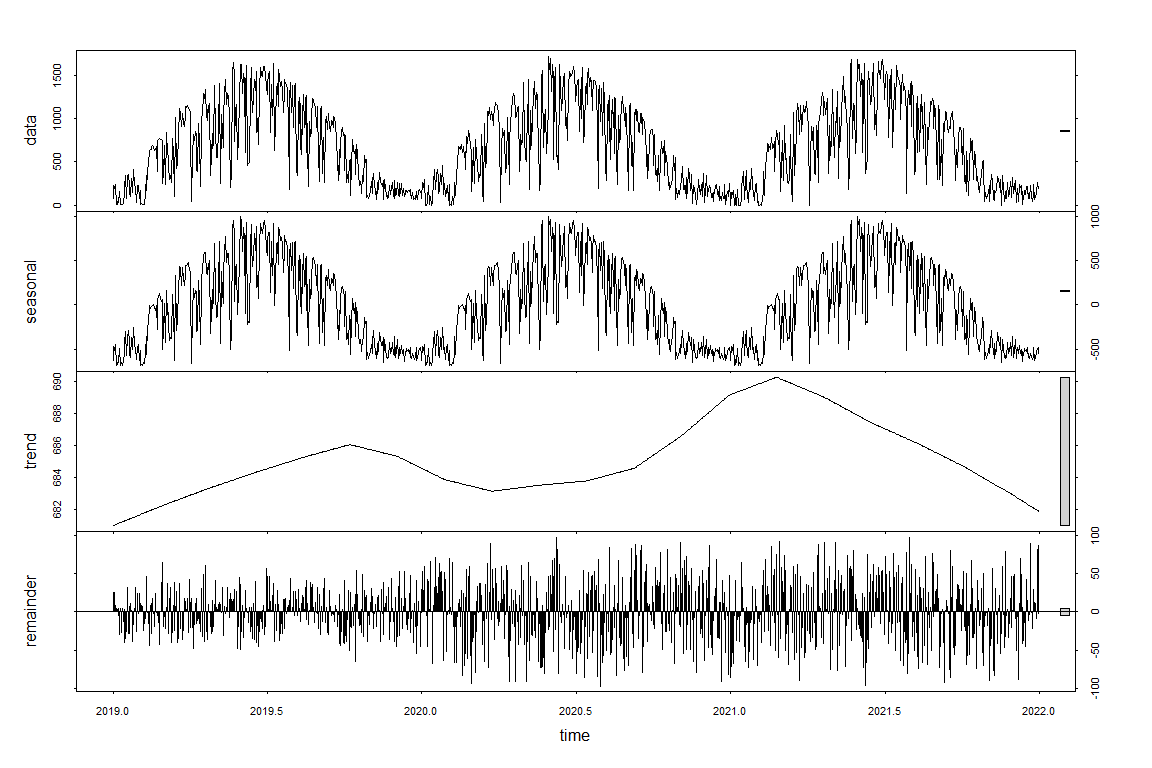


First, some artificial years were created by simply adding noise with the jitter() function. Because of the occurrence of minus values in this procedure, minus values have to be corrected to zero in a second process. With the new data set, a TS decomposition is plotted via the function stl(). Then, a new model is trained and a forecast is predicted. Again, the prediction performance is rather poor.

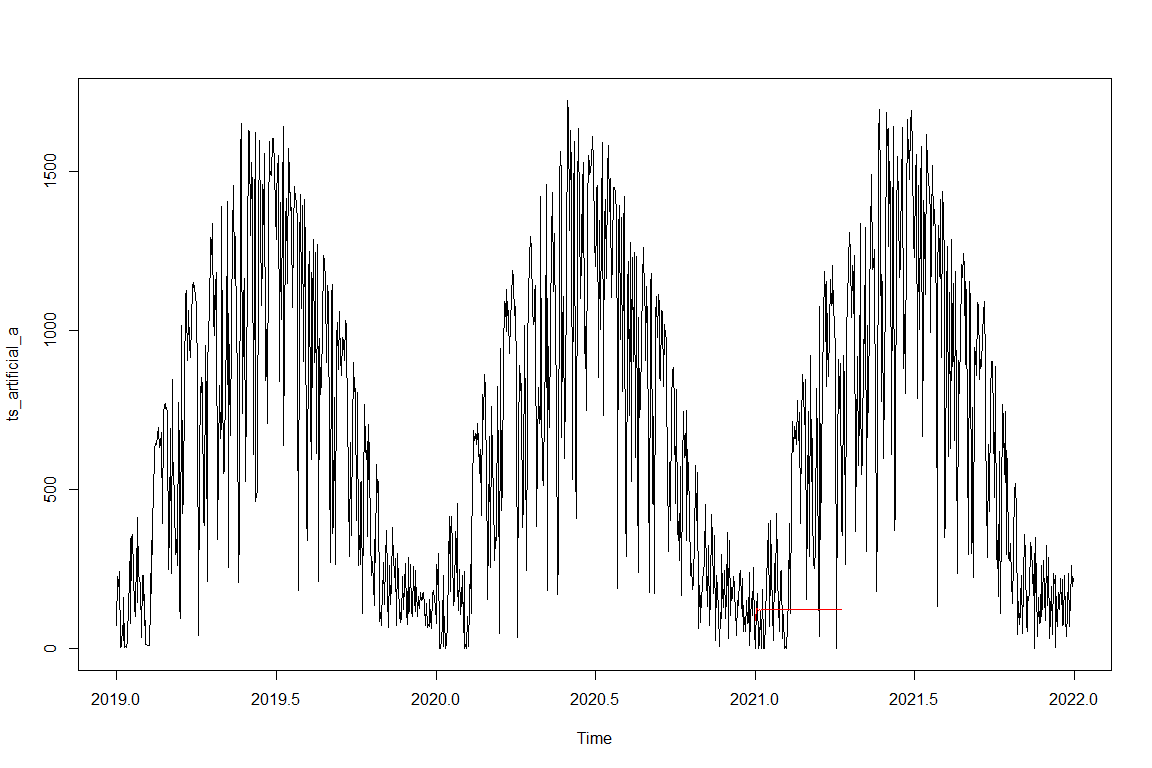
ts\_artificial <- ts(c(ts\_2, ts\_artif\_1, ts\_artif\_2), start = c(2019), deltat = 1/365)  
decomp<-stl(ts\_artificial, s.window = 365)  
str(ts\_artificial)

## 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.



## Time Series Analysis - with Manipulated Data - new approach

In the previous part, the creation of the artificial years brought up some concerns regarding the quality of those artificial data. In this part, the focus is set to discover a method to generate artificial data similar to the real data in terms of variance, mean and distribution of the data. The method is as follow: : 1. Use of GAM smoother as base for an artificial year 2. Add normally distributed, left screwed noise.

For that, a new data frame with indexed data is created. This is done to avoid the handling of date type.

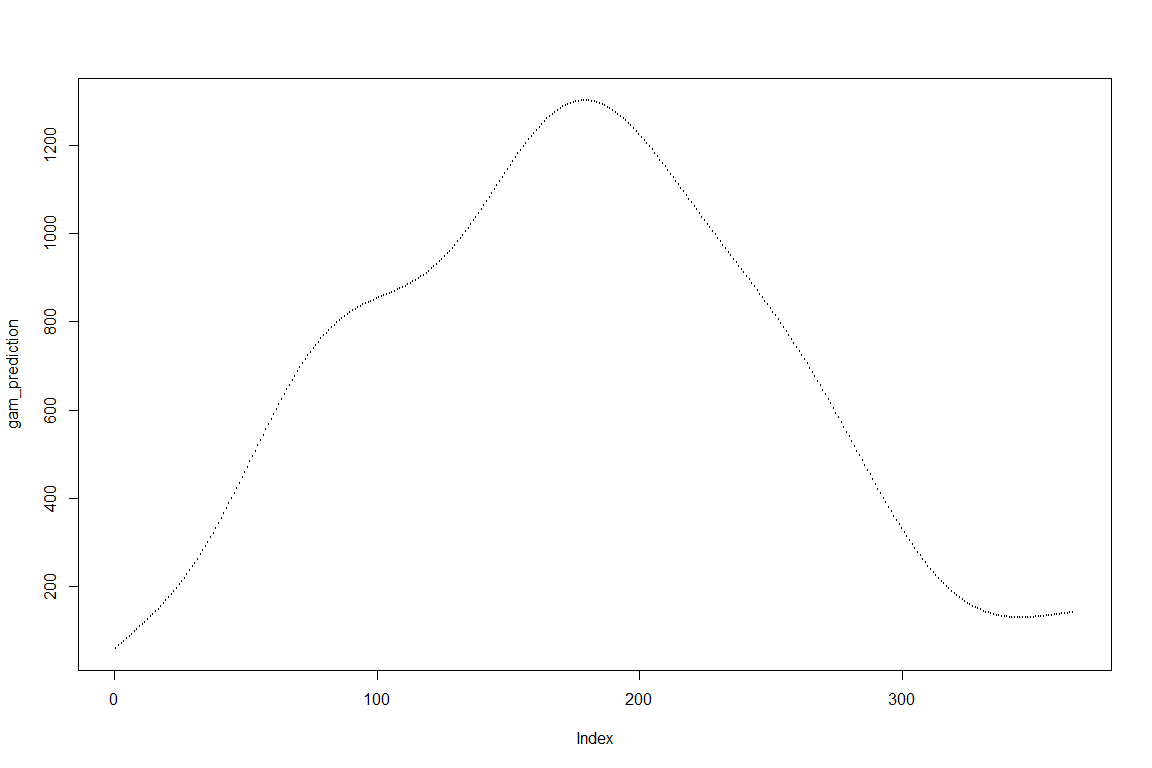
df\_artif <- data.frame(time = (1:365), generation\_kw = df\_plantA\_resample\_2$Generation\_kW)

Now, a GAM is computed from the new data frame.

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

In order to generate a data set out of the GAM, 365 prediction are computed within the entire year. The result is plotted to visually check the result. As shown below, the data frame plot is, at least qualitatively, very similar to the ggplot GAM smoother line. This result is taken as a base for the artificial years.

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



After the creation of a “base year”, noise is added. In order to keep the heteroskedastic behavior of the real data, the noise is added as a multiplication by a random coefficient. This coefficient is first created as a left screwed, normally distributed random number. Then, it is normalized from 0 to 1. Last, those noise coefficient between 0 and 1 are again multiplied with an random coefficient in a predefined range (0.3 to 1.6), in order to level the mean of the data set. A quick look at the plot show a satisfying result.

Add noise to smoothed base year:

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\_2 <- ran\_coef\_norm \* runif(ran\_coef\_norm, min = 0.3, max = 1.6)  
gam\_yr\_with\_noise <- gam\_prediction \* ran\_coef\_norm\_2

## [1] "Real Data"

## [1] "Mean: 684.24677260274"

## [1] "Variance: 232351.304469434"

## [1] "GAM prediction"

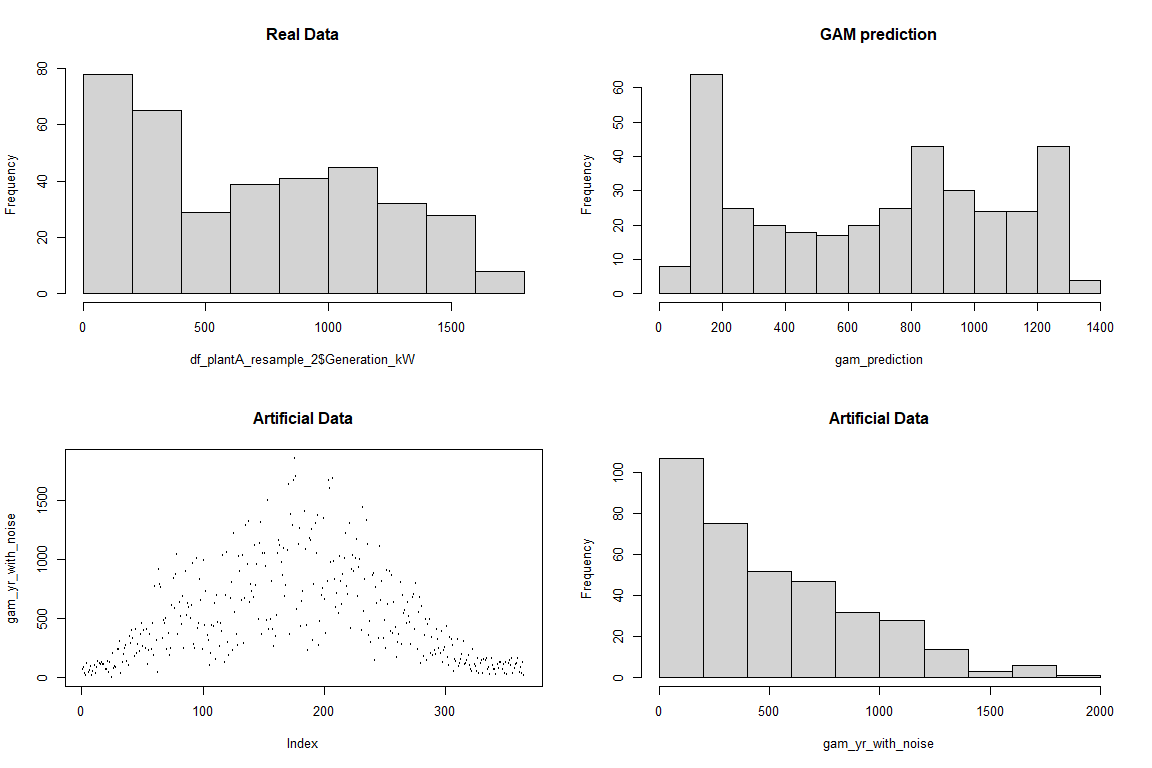
## [1] "Mean: 684.246772602739"

## [1] "Variance: 159111.627971204"

## [1] "Artificial Data"

## [1] "Mean: 504.585754013336"

## [1] "Variance: 161066.585646483"



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 the 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 and if they are cumulatively fulfilled, the function saves the year data in a list. At the end of the loop, the list of years is returned by the function.

## Function To Get Artificial Year

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)  
 }  
 }  
 }

Following is the function that loops through parameter list:

get\_artificial\_years <- function(mean\_real\_yr, variance\_real\_yr, gam\_prediction) {  
   
 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  
 }  
 ran\_coef <- rsnorm(365, mean = 1, sd = 1, xi = as.numeric(u[i,][1]))  
 ran\_coef\_norm <- normalize(ran\_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\_prediction \* 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(1:10000)\*5000)){  
 paste("Year computed: ", i)  
 }  
   
 i <- i + 1  
 }  
 df\_sets  
}

Call the function to compute about 40’000 simulated years and and save all ones that meet the threshold requirements. With set.seed() to 41, four years fulfill the thresholds.

## [1] 15498  
## [1] 0.9998943  
## [1] 0.9574415  
## [1] 32421  
## [1] 1.018895  
## [1] 0.9638508  
## [1] 35515  
## [1] 1.022846  
## [1] 0.9602693  
## [1] 35637  
## [1] 1.019101  
## [1] 0.9542123

Plot and compare the results with the real data. As it is to see, the results appear to be satisfying, excepting the few outliers, which are values over ~1800.

## [1] "Real Data"

## [1] "Mean: 684.24677260274"

## [1] "Variance: 232351.304469434"

## [1] "Artificial Data"

## [1] "Mean: 671.5575893729"

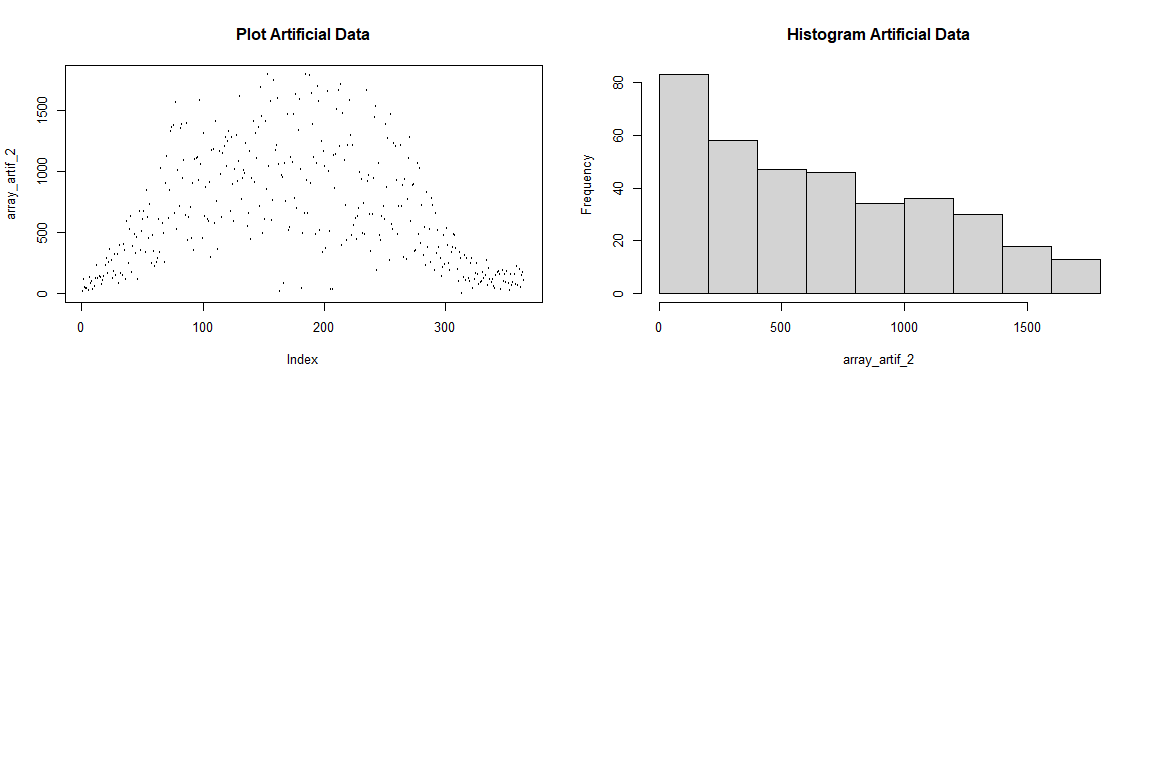
## [1] "Variance: 241065.625833288"

As last manipulation, the few outliers over a value of 1800 are pulled down. This is done for all the returned artificial years.

noised\_years[1,][noised\_years[1,] > 1800] <- (as.numeric(noised\_years[1,]) \* 0.75)

## Warning in noised\_years[1, ][noised\_years[1, ] > 1800] <-  
## (as.numeric(noised\_years[1, : number of items to replace is not a multiple of  
## replacement length

df\_artif\_2 <- t(noised\_years[1,])  
array\_artif\_2 <- array(as.numeric(unlist(df\_artif\_2)))  
par(mfrow=c(2,2))  
plot(array\_artif\_2, cex = .2, main="Plot Artificial Data")  
hist(array\_artif\_2, main = "Histogram Artificial Data")



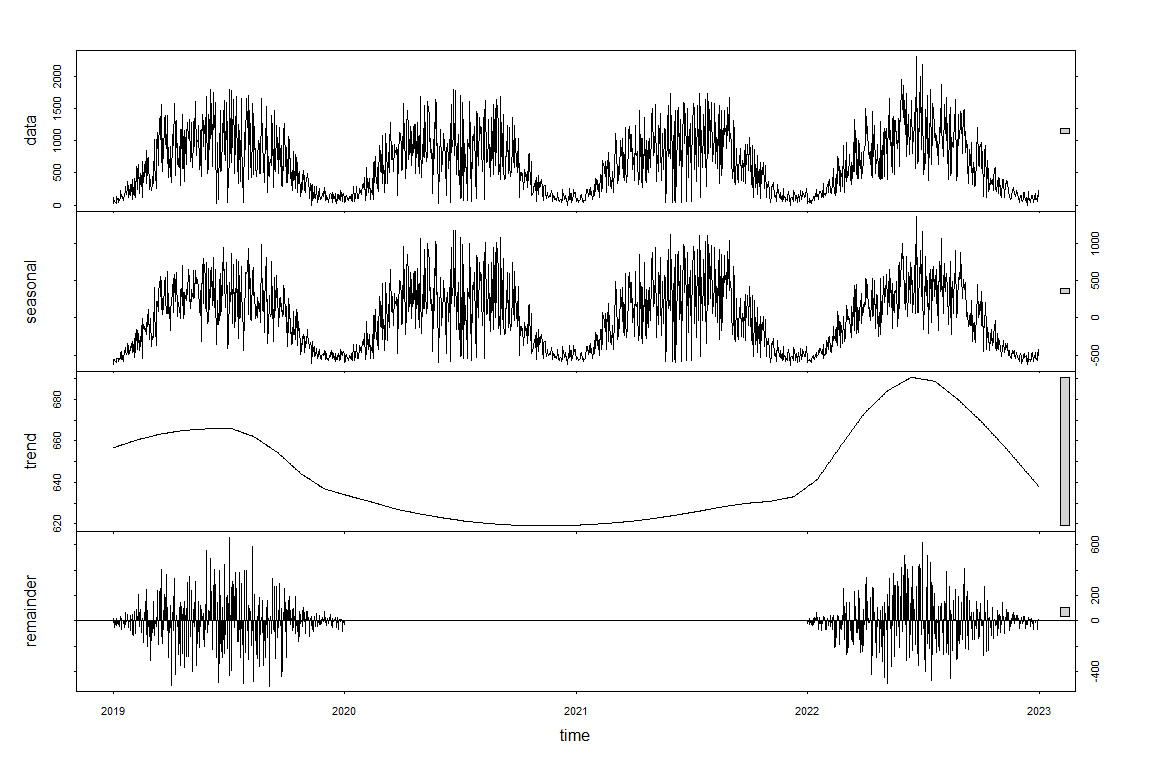
## Warning in noised\_years[2, ][noised\_years[2, ] > 1800] <-  
## (as.numeric(noised\_years[2, : number of items to replace is not a multiple of  
## replacement length

## Warning in noised\_years[3, ][noised\_years[3, ] > 1800] <-  
## (as.numeric(noised\_years[3, : number of items to replace is not a multiple of  
## replacement length

### Plot final Results Of Artificial Data

Create TS with artificial years. Then investigate with the stl() function. Each year differs now much more than the others. This result is considered as successful. This experimental excursion delivers an interesting result which could be now further used to explore predictive modelling. As for now, this will be the end of this part.

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



# Conclusion

First of all, we would like to mention that by taking this module and working on the group project, the advantages of the software R became apparent to us, also compared to Python. Especially the packages ggplot, lubridate as well as dplyr are extremely useful and intuitive. We are convinced that we will use R for certain tasks in the future. R Markdown is also interesting. However, we had an error message at the end when creating the pdf, which we could not solve until now. One possibility we see is to switch to Jupyter Lab, which also supports R and the language markdown to create documents.

Regarding a forecast model for photovoltaic production, it should be mentioned that a model is strongly dependent on the weather forecast. Especially to predict exact hourly values is sometimes difficult because a passage of a cloud can change the situation like a switch. Therefore, it might be difficult to predict exact hourly values or even 15min values in the future, depending on the weather situation. The inclusion of a physical model could further improve the prediction.

Furthermore, we have found that a purely mathematical model like the ARIMA is clearly inferior to a GAM model with weather data as predictors. Since we considered both models in different modules it was very interesting for us to compare them.

For further investigation of the ideal battery size, including the cost per battery size and electricity prices, would be an interesting addition to improve the overall system profitability.