Analysis and Forecasting of Electricity Prices in Italy:

Classical Econometrics vs Automated Procedures

Dylan Tartarini, Mirco Balduini

Introduction:

This is a brief presentation regarding our study on electricity prices from the Italian market. Our goal is to compare approaches for modelling and forecasting complex, high-frequency data in a semi-automatic way. More details on why this issue is relevant and on the electricity prices univariate time series analysis can be found in this work's presentation freely available in our GitHub repos. We dispose of the hourly time series of the Unique National Price (PUN) starting from Jan, 1st, 2008 up to Dec, 31st, 2019. PUN is the reference price detected on Italian Power Exchange (IPEX): data are free to use and can be found on the web-page of the Italian Regulator for the Energy Markets (GME) following this link: https://www.mercatoelettrico.org/it/.
Let's dive in!

Set-Up

Loading needed packages..

```r
library(urca)

library(rmarkdown)

library(stats)
library(tseries)

library(tidyr)

library(readxl)

library(ggplot2)

library(dplyr)

library(tidyverse)

library(lubridate)

library(datetime)

library(colortools)

library(zoo)

library(gridExtra)

library(astsa)

library(describedata)

library(FactoMineR)
library(factoextra)

library(data.table)

library(ARDL)

library(dLagM)

library(prophet)

library(forecast)
```

Setting directory and loading data..

```
PUN_db <- read_excel("/PUN_db.xlsx", sheet = "2008_2019")

data.frame(PUN_db)
```

Data Manipulation

Manipulation of the full hourly dataset..

```
PUN_db <- PUN_db %>%
  separate(Hour, into = c("Date", "Time"), sep = " ", remove = FALSE) %>%
  mutate(Date = lubridate::as_date(Date, format = "%Y/%m/%d"),
         Time = hms::as_hms(str_c(Time, ":00")))

PUN_db$Date <- PUN_db$Hour <- NULL

PUN_db$DateReal <- as.Date(PUN_db$DateReal, "%Y/%m/%d")

PUN_db$DateTime <- as.POSIXct(paste(PUN_db$DateReal, PUN_db$Time), format = "%Y-%m-%d %H:%M:%S")

PUN_db <- unique(setDT(PUN_db)[order(DateTime, PUN)], by = "DateTime")
```

Widening data with the purpose of summarizing with daily averages..

```
PUN_db_wide <- pivot_wider(data = PUN_db, id_cols = DateReal, names_from = Time, values_from = PUN)

PUN_db_wide <- data.frame(PUN_db_wide)

colnames(PUN_db_wide)[2:25] <- c("h24","h01","h02", "h03", "h04", "h05", "h06", "h07",
                                 "h08", "h09", "h10", "h11", "h12", "h13", "h14",
                                 "h15", "h16", "h17", "h18","h19","h20", "h21",
                                 "h22","h23")
PUN_db_wide <- PUN_db_wide[,c("DateReal", "h01","h02", "h03", "h04", "h05", "h06", "h07",
                                "h08", "h09", "h10", "h11", "h12", "h13", "h14",
                                "h15", "h16", "h17", "h18","h19","h20", "h21",
                                "h22","h23", "h24")]

PUN_db_wide$DailyPun <- rowMeans(PUN_db_wide[2:25])

PUN_db_wide$DailyPun[is.na(PUN_db_wide$DailyPun)] <- mean(PUN_db_wide$DailyPun, na.rm = TRUE)

for(i in 1:ncol(PUN_db_wide[2:26])){
  PUN_db_wide[is.na(PUN_db_wide[,i]), i] <- mean(PUN_db_wide[,i], na.rm = TRUE)
}
```

Creating other dates variables..

```
PUN_db$Week <- as.Date(cut(PUN_db$DateReal, breaks = "week"))
```

```
PUN_db$WeekNumber <- isoweek(ymd(PUN_db$DateReal))

PUN_db$Month<- month(PUN_db$DateReal)

PUN_db$MonthName <- month.abb[PUN_db$Month]

PUN_db$Year <- year(PUN_db$DateReal)

PUN_db$DayCount <- day(PUN_db$DateReal)

PUN_db$YearMonth <- as.yearmon(PUN_db$DateReal)

PUN_db$DayName <- strftime(PUN_db$DateReal, "%A")

PUN_db_wide$Year <- year(PUN_db_wide$DateReal)
```

Looking at our 2 dataframes..

```
head(PUN_db)

##      DateReal     Time      PUN            DateTime       Week WeekNumber Month
## 1: 2008-01-01 00:00:00 60.65949 2008-01-01 00:00:00 2007-12-31          1     1
## 2: 2008-01-01 01:00:00 70.68000 2008-01-01 01:00:00 2007-12-31          1     1
## 3: 2008-01-01 02:00:00 60.68000 2008-01-01 02:00:00 2007-12-31          1     1
## 4: 2008-01-01 03:00:00 55.61446 2008-01-01 03:00:00 2007-12-31          1     1
## 5: 2008-01-01 04:00:00 51.53576 2008-01-01 04:00:00 2007-12-31          1     1
## 6: 2008-01-01 05:00:00 51.51517 2008-01-01 05:00:00 2007-12-31          1     1
##    MonthName Year DayCount YearMonth DayName
## 1:       Jan 2008        1  gen 2008 martedì
## 2:       Jan 2008        1  gen 2008 martedì
## 3:       Jan 2008        1  gen 2008 martedì
## 4:       Jan 2008        1  gen 2008 martedì
## 5:       Jan 2008        1  gen 2008 martedì
## 6:       Jan 2008        1  gen 2008 martedì

head(PUN_db_wide)

##     DateReal      h01      h02      h03      h04      h05      h06      h07
## 1 2008-01-01 70.68000 60.68000 55.61446 51.53576 51.51517 51.50580 54.60351
## 2 2008-01-02 51.60194 51.63099 51.62268 33.64163 33.62347 33.61500 52.65000
## 3 2008-01-03 51.63343 51.64186 51.62428 40.62479 40.61782 40.62525 60.64000
## 4 2008-01-04 67.27000 60.51000 50.61000 50.57000 50.60000 51.55814 58.67000
## 5 2008-01-05 60.59334 60.58541 51.63341 51.58708 51.57771 51.60591 51.62355
## 6 2008-01-06 61.61854 52.65000 51.16000 51.14000 51.13720 51.14000 51.64000
##        h08      h09       h10       h11       h12       h13      h14      h15
## 1 51.65000 50.65000  52.61706  61.39766  61.46347  60.66358 51.76000 51.75000
## 2 75.38000 75.42000  88.66000  88.67000  90.64000  68.64000 60.68000 59.69000
## 3 66.65000 73.64000  94.51178  97.32818  94.47810  76.66000 56.10000 55.10000
## 4 76.39053 74.71000  99.99223 109.07064 100.05762 100.32555 85.05000 85.05000
## 5 79.62000 94.71131 105.04592 105.02548  95.22786  80.42213 71.14578 70.08969
## 6 51.66000 51.69000  63.96753  80.56008  81.45371  68.48758 67.70436 57.69000
##        h16       h17       h18       h19       h20      h21       h22      h23
## 1 51.74000  51.77000  74.73408  79.42406  84.43219 112.4165 101.05128 60.71639
## 2 92.00000 102.75877 182.20822 182.28900 156.44321 155.5196  99.99913 80.00000
## 3 79.83297  90.79334 177.75430 177.82219 158.06140 157.2204 123.73107 89.58532
## 4 93.85322 101.20273 173.49702 173.37685 144.49503 136.6006 129.26353 90.72918
## 5 70.08071  94.94233 131.90909 145.89470 136.00536 104.1255  89.70437 60.67738
## 6 57.69000  84.32397 109.36927 144.09782 149.64839 104.6500  93.63190 69.67000
##        h24 DailyPun Year
## 1 60.65949 63.12627 2008
## 2 71.04000 84.93432 2008
## 3 65.61847 86.34562 2008
## 4 67.65000 92.96262 2008
## 5 60.63928 82.26972 2008
## 6 51.72000 73.27085 2008
```

Computing averages and standard deviations

```
mean <- mean(PUN_db$PUN)

hourly_average <- PUN_db %>%
  group_by(Time) %>%
  summarize(mean_PUN = mean(PUN))


daily_average <- PUN_db %>%
  group_by(DateReal) %>%
  summarize(mean_PUN = mean(PUN))

daily_average$DayOfYear <- strftime(daily_average$DateReal, "%j")
daily_average$Year <- year(daily_average$DateReal)
daily_average$Year <- as.factor(daily_average$Year)

weekly_average <- PUN_db %>%
  group_by(Week) %>%
  summarize(mean_PUN = mean(PUN))
weekly_average$WeekNumber <- isoweek(ymd(weekly_average$Week))
weekly_average$Year <- year(weekly_average$Week)
weekly_average$Year <- as.factor(weekly_average$Year)
weekly_average$WeekNumber <- as.factor(weekly_average$WeekNumber)

mon_avg_perYear <- aggregate(PUN_db$PUN, list(Date=format(PUN_db$YearMonth, "%Y-%m")),mean)
colnames(mon_avg_perYear) <- c("Date", "avg_PUN")
mon_avg_perYear <- mon_avg_perYear %>%
  separate(Date, into = c("Year", "Month"), sep = "-", remove = FALSE)
mon_avg_perYear$Year <- as.factor(mon_avg_perYear$Year)
mon_avg_perYear$Month <- as.factor(mon_avg_perYear$Month)

yearly_average <- PUN_db %>%
  group_by(Year) %>%
  summarize(mean_PUN = mean(PUN))

sd <- sd(PUN_db$PUN)

sd_hourly <- PUN_db %>%
  group_by(Time) %>%
  summarize(sd_PUN = sd(PUN))

sd_yearly <- PUN_db %>%
  group_by(Year) %>%
  summarize(sd_PUN = sd(PUN))

sd_montly_perYear <- PUN_db %>%
  group_by(YearMonth) %>%
  summarize(sd_PUN = sd(PUN))
```

From Data Frames to time series..

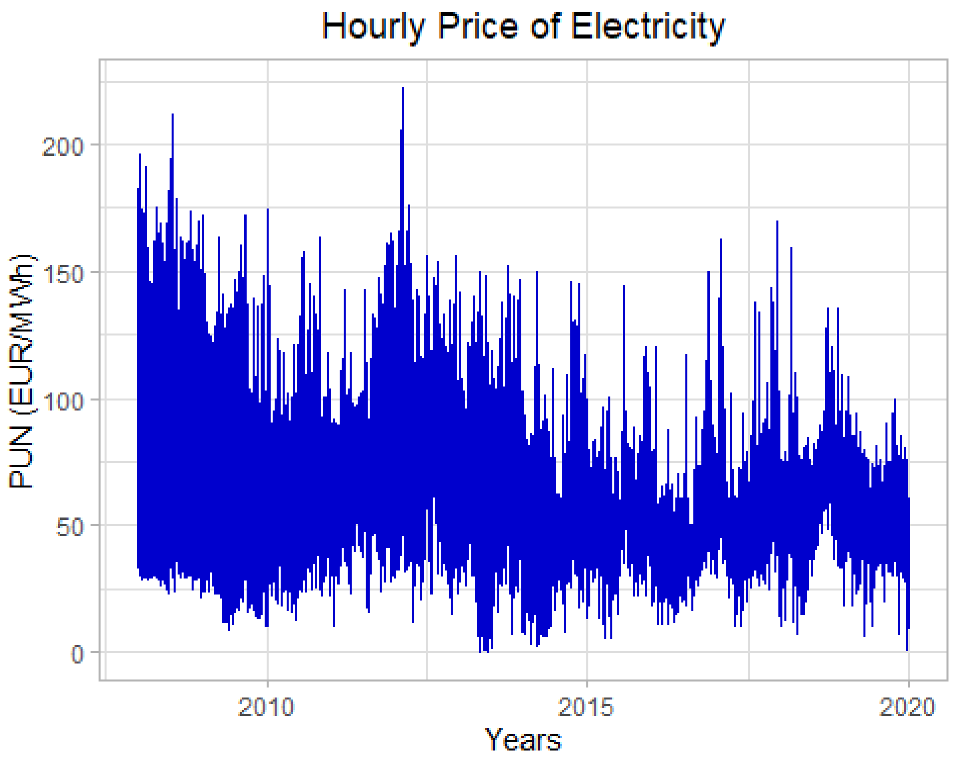
```
dec19 <- subset(PUN_db, (Year == 2019 & MonthName == "Dec"))

start_date <- c(2008,1,1)
PUN_ts <- ts(PUN_db$PUN, start = start_date, frequency = 24*365.25)

dailyPUN_ts <- ts(PUN_db_wide$DailyPun, start = start_date, frequency = 365.25)

PUN_2018 <- subset(PUN_db, Year == 2018)
```

```
hourlyPun_2018 <- ts(PUN_2018$PUN, start = c(2018, 1, 1) ,frequency = 24*365)

PUN_2018_19 <- subset(PUN_db, Year >= 2018)

hourlyPUN <- ts(PUN_2018_19$PUN, start = c(2018, 1, 1) ,frequency = 24*365)

dec19_ts <- ts(dec19$PUN, start = c(2019,12,01), freq = 24*31)
```

Exploratory Data Analysis

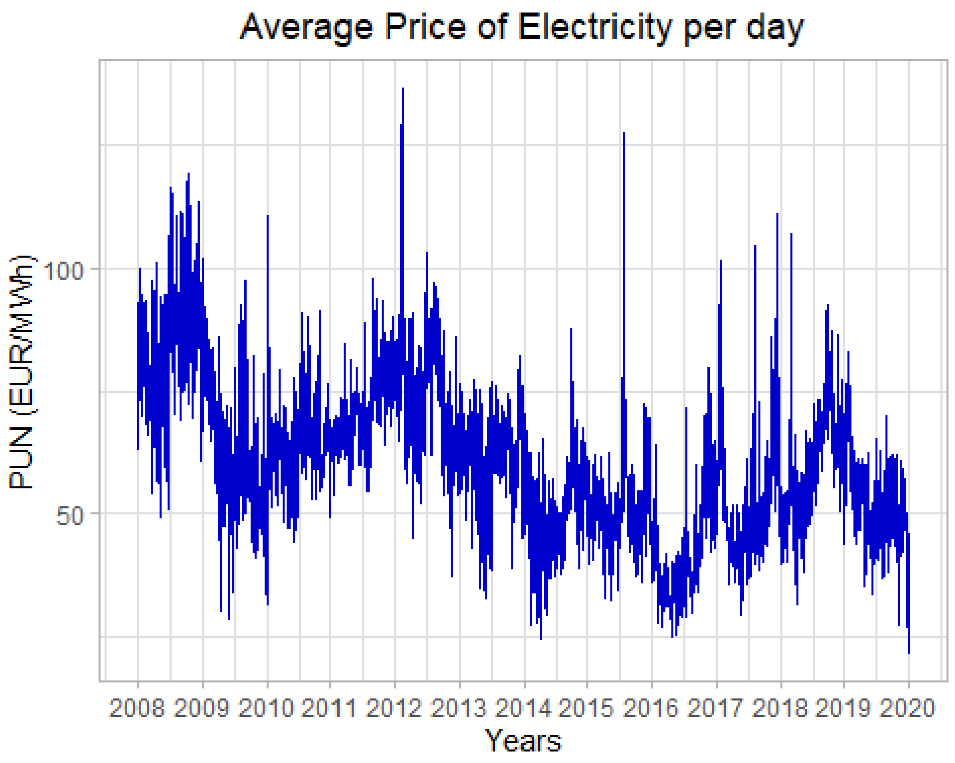
The full hourly dataset:

```
ggplot(PUN_db, aes(x=DateTime, y=PUN))+
  geom_line(color="mediumblue")+
  xlab("Years")+
  ylab("PUN (EUR/MWh)")+
  ggtitle("Hourly Price of Electricity")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))
```



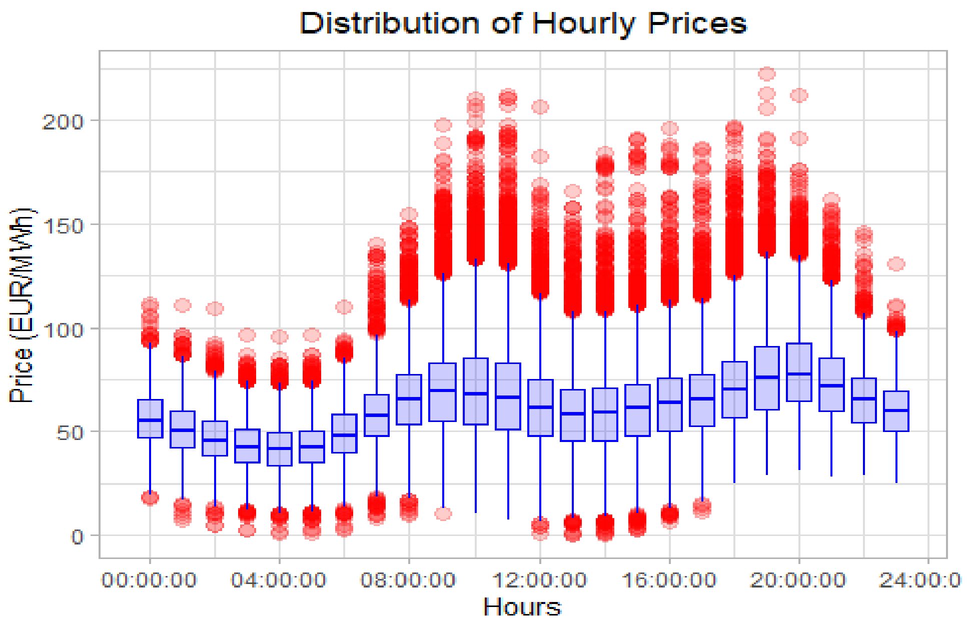
Same dataset, but for daily average observations:

```
ggplot(daily_average, aes(x=DateReal, y=mean_PUN))+
  geom_line(color = "mediumblue")+
  xlab("Years")+
  ylab("PUN (EUR/MWh)")+
  ggtitle("Average Price of Electricity per day")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))+
  scale_x_date(date_breaks = "1 year", date_labels = "%Y")
```



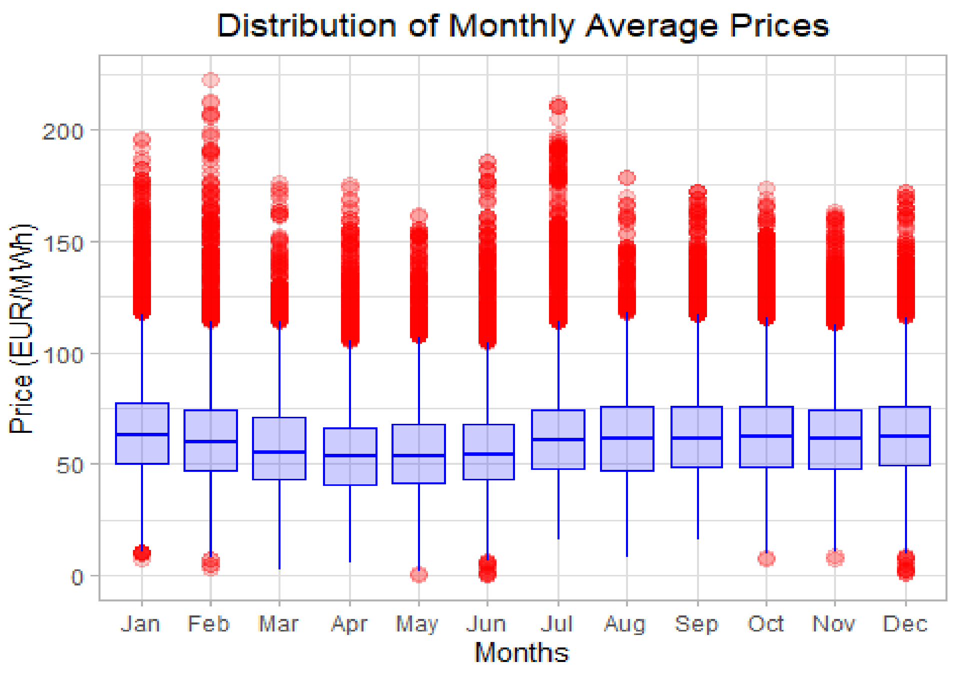
Distribution of PUN across hours of the day:

```
ggplot(PUN_db, aes(x=Time, y=PUN))+
  geom_boxplot(aes(group=Time), color = "blue", fill = "blue", alpha = 0.2,
               outlier.color = "red", outlier.fill = "red", outlier.size = 3)+
  xlab("Hours")+
  ylab("Price (EUR/MWh)")+
  ggtitle("Distribution of Hourly Prices")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))
```



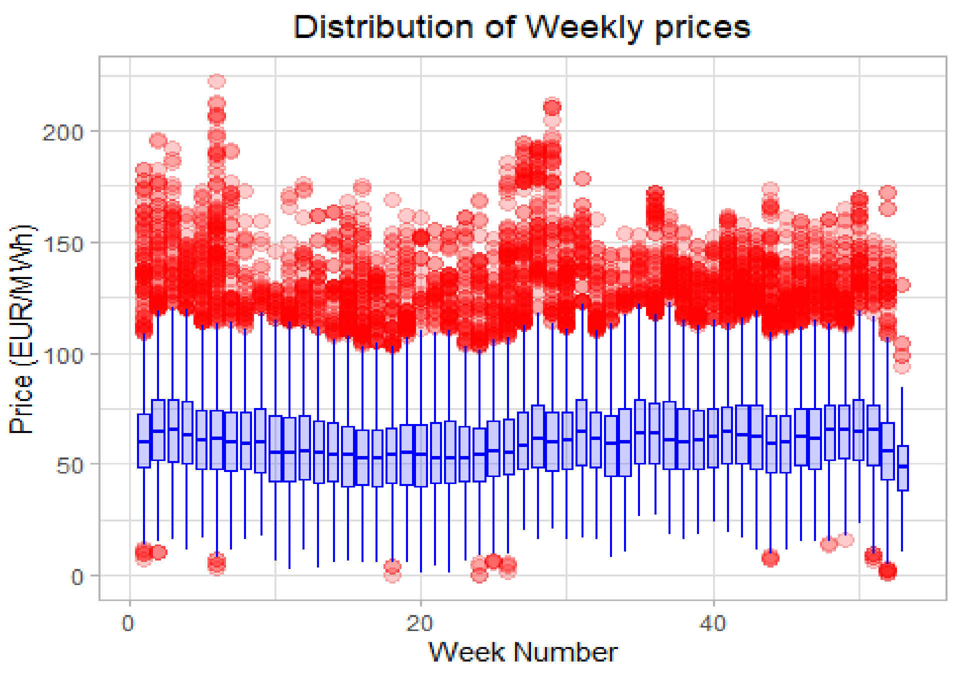
Distribution of PUN across months of the year

```r
as.factor(PUN_db$MonthName)

ggplot(PUN_db, aes(x=MonthName, y=PUN))+
  geom_boxplot(aes(group=Month), color = "blue", fill = "blue", alpha = 0.2,
               outlier.color = "red", outlier.fill = "red", outlier.size = 3)+
  scale_x_discrete(limits = month.abb)+
  xlab("Months")+
  ylab("Price (EUR/MWh)")+
  ggtitle("Distribution of Monthly Average Prices")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))
```



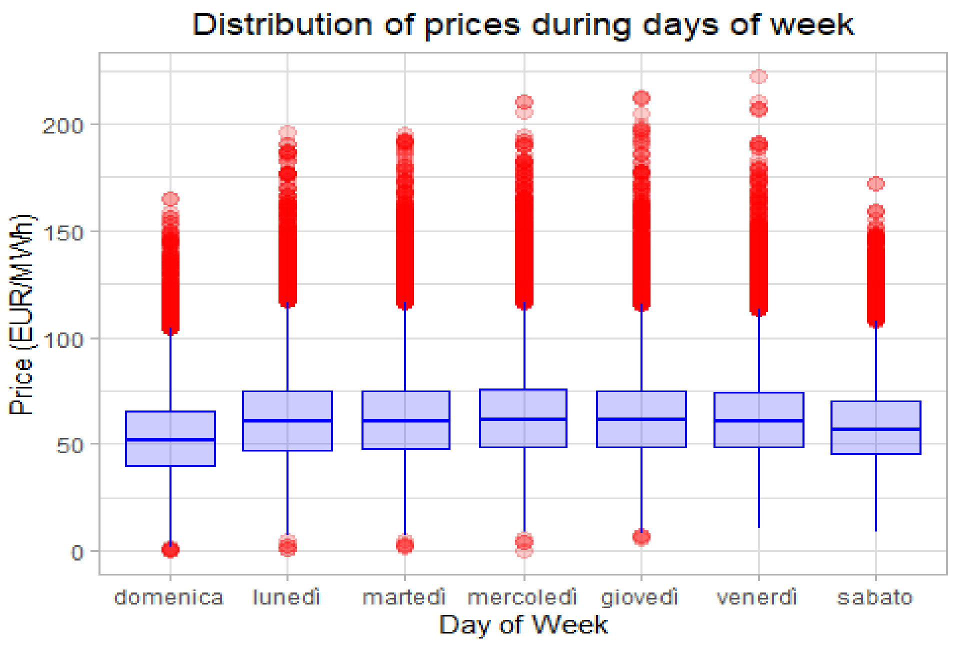
Distribution of PUN across weeks of the year

```r
ggplot(PUN_db, aes(x=WeekNumber, y=PUN))+
  geom_boxplot(aes(group=WeekNumber), color = "blue", fill = "blue", alpha = 0.2,
               outlier.color = "red", outlier.fill = "red", outlier.size = 3)+
  xlab("Week Number")+
  ylab("Price (EUR/MWh)")+
  ggtitle("Distribution of Weekly prices")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))
```



Distribution of PUN across days of the week

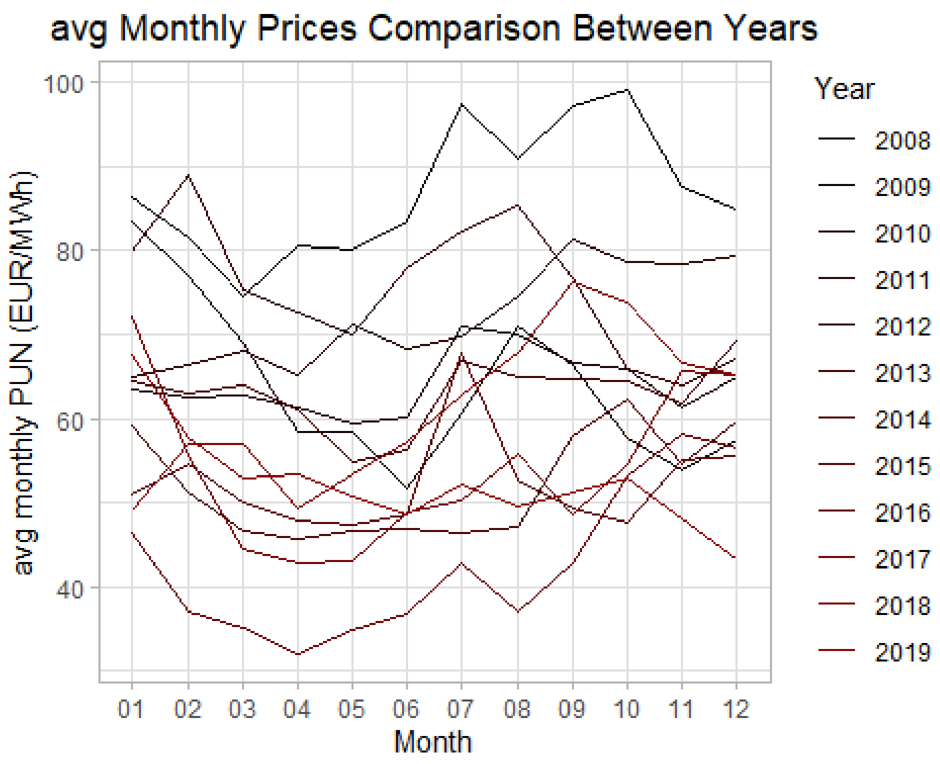
```r
PUN_db$DayName <- factor(PUN_db$DayName, c("domenica", "lunedì", "martedì", "mercoledì",
                                            "giovedì", "venerdì", "sabato"))

ggplot(PUN_db, aes(x=DayName, y=PUN))+
  geom_boxplot(aes(group=DayName), color = "blue", fill = "blue", alpha = 0.2,
               outlier.color = "red", outlier.fill = "red", outlier.size = 3)+
  xlab("Day of Week")+
  ylab("Price (EUR/MWh)")+
  ggtitle("Distribution of prices during days of week")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))
```



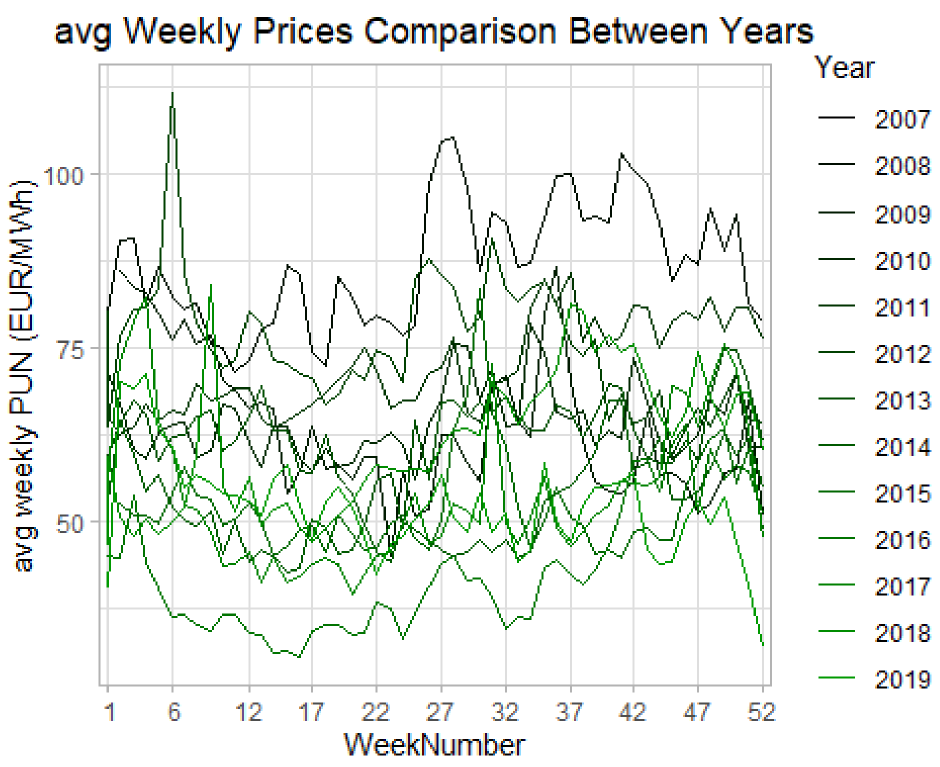
EDA: Visualizing Seasonality

The next three plots show that electricity prices follow the same pattern during different years

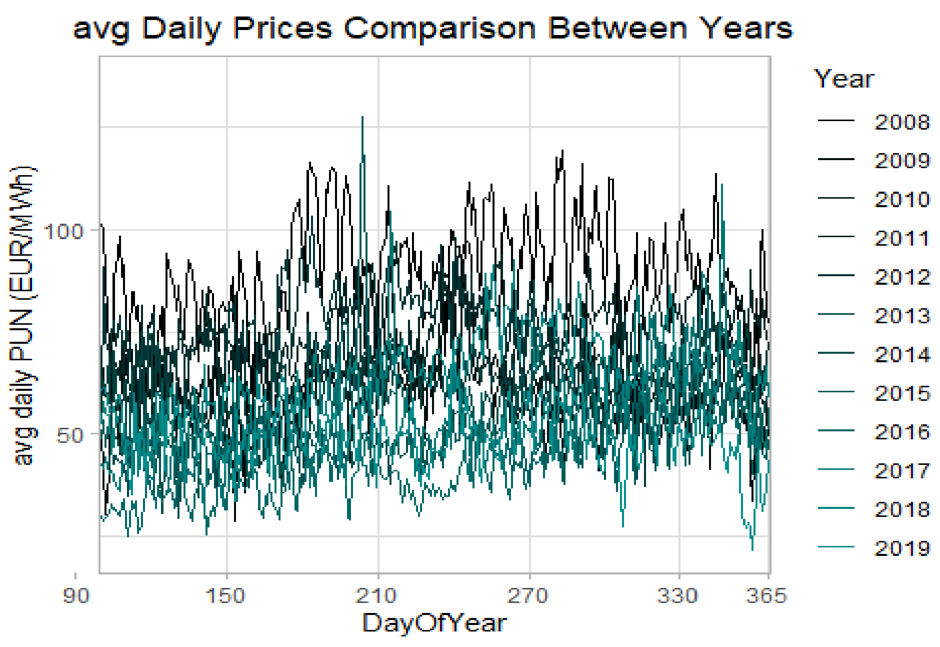
```
year_pal1 <- sequential(color = "red", percentage = 5, what = "value")

year_pal2 <- sequential(color = "green", percentage = 5, what = "value")

year_pal3 <- sequential(color = "cyan3", percentage = 5, what = "value")

ggplot(mon_avg_perYear, aes(x = Month, y = avg_PUN, group = Year))+
  geom_line(aes(colour = Year))+
  ggtitle("avg Monthly Prices Comparison Between Years")+
  ylab("avg monthly PUN (EUR/MWh)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))+
  scale_color_manual(values = year_pal1)
```



```
ggplot(weekly_average, aes(x = WeekNumber, y = mean_PUN, group = Year))+
  geom_line(aes(colour = Year))+
  ggtitle("avg Weekly Prices Comparison Between Years")+
  ylab("avg weekly PUN (EUR/MWh)")+
  scale_x_discrete(limits = c(1:52), breaks =c(1,6,12,17,22,27,32,37,42,47,52))+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))+
  scale_color_manual(values = year_pal2)
```

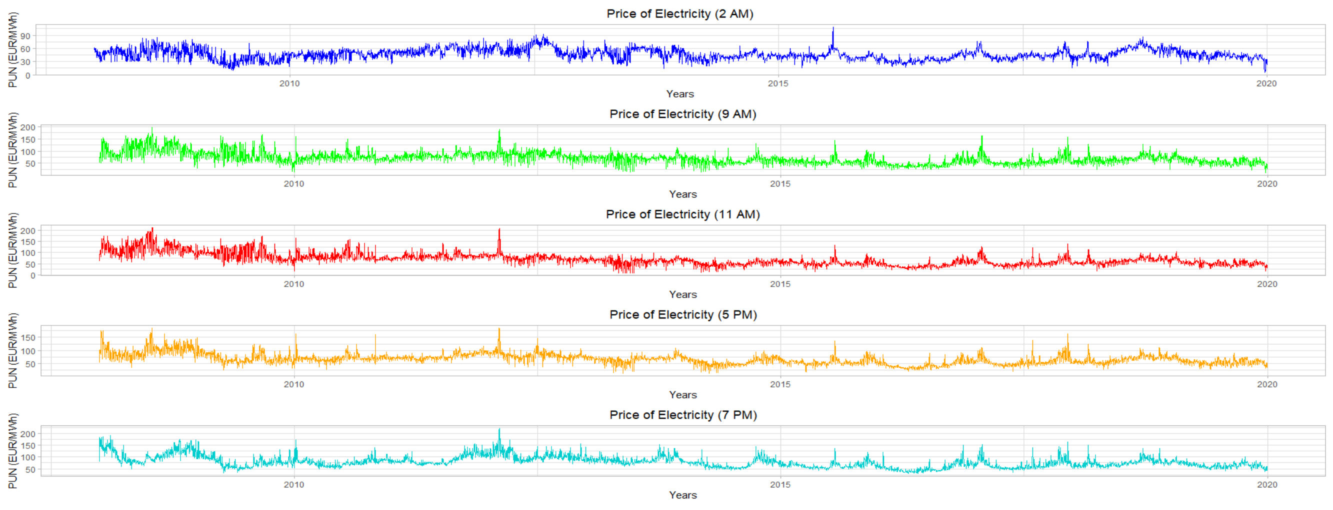


```r
ggplot(daily_average, aes(x = DayOfYear, y = mean_PUN, group = Year))+
  geom_line(aes(colour = Year))+
  ggtitle("avg Daily Prices Comparison Between Years")+
  ylab("avg daily PUN (EUR/MWh)")+
  scale_x_discrete(limits = c(1:365), breaks =c(1,30,90, 150, 210,270, 330, 365))+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))+
  scale_color_manual(values = year_pal3)
```



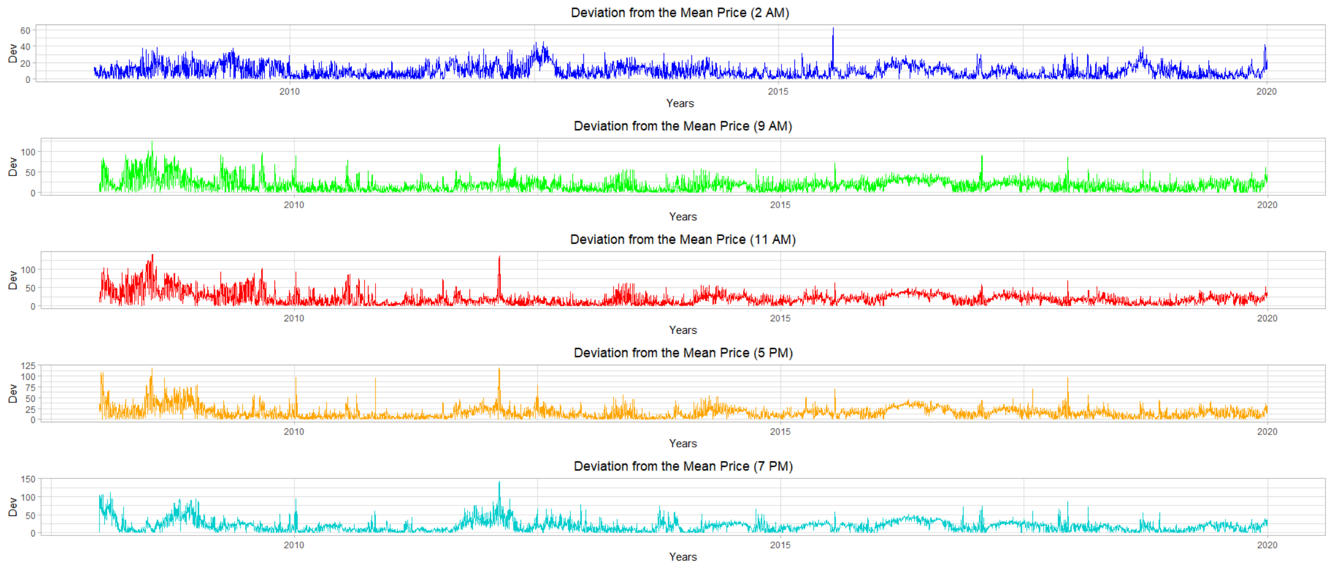
Electricity Prices show a **common structure** both in the **conditional mean..**

```r
plt_h2 <- ggplot(PUN_db_wide , aes(x=DateReal, y=h02))+
  geom_line(color="blue")+
  xlab("Years")+
  ylab("PUN (EUR/MWh)")+
  ggtitle("Price of Electricity (2 AM)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))

plt_h9 <- ggplot(PUN_db_wide , aes(x=DateReal, y=h09))+
  geom_line(color="green")+
  xlab("Years")+
  ylab("PUN (EUR/MWh)")+
  ggtitle("Price of Electricity (9 AM)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))

plt_h11 <- ggplot(PUN_db_wide , aes(x=DateReal, y=h11))+
  geom_line(color="red")+
  xlab("Years")+
  ylab("PUN (EUR/MWh)")+
  ggtitle("Price of Electricity (11 AM)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))

plt_h17 <- ggplot(PUN_db_wide , aes(x=DateReal, y=h17))+
  geom_line(color="orange")+
  xlab("Years")+
  ylab("PUN (EUR/MWh)")+
  ggtitle("Price of Electricity (5 PM)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))

plt_h19 <- ggplot(PUN_db_wide , aes(x=DateReal, y=h19))+
  geom_line(color="cyan3")+
  xlab("Years")+
  ylab("PUN (EUR/MWh)")+
  ggtitle("Price of Electricity (7 PM)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))

grid.arrange(plt_h2, plt_h9, plt_h11, plt_h17, plt_h19, nrow = 5)
```



..and in the **conditional conditional variance**

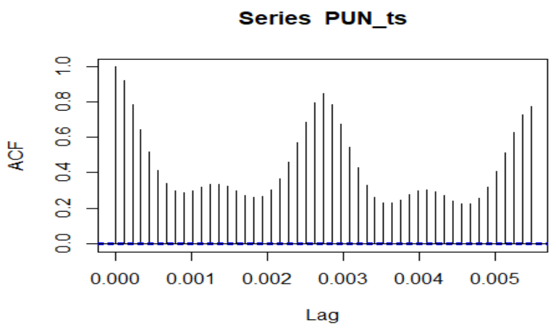
```
PUN_db_wide_h <- data.frame(PUN_db_wide$DateReal, PUN_db_wide$h02, PUN_db_wide$h09, PUN_db_wide$h11,
                            PUN_db_wide$h17, PUN_db_wide$h19 )

colnames(PUN_db_wide_h)[1:6] <- c("DateReal","h02", "h09", "h11", "h17","h19")

mean_h2 <- mean(PUN_db_wide_h$h02)

mean_h9 <- mean(PUN_db_wide_h$h09)

mean_h11 <- mean(PUN_db_wide_h$h11)

mean_h17 <- mean(PUN_db_wide_h$h17)

mean_h19 <- mean(PUN_db_wide_h$h19)

PUN_db_wide_h$Dev2 <- abs(PUN_db_wide_h$h02 - mean_h2)

PUN_db_wide_h$Dev9 <- abs(PUN_db_wide_h$h09 - mean_h9)

PUN_db_wide_h$Dev11 <- abs(PUN_db_wide_h$h11 - mean_h11)

PUN_db_wide_h$Dev17 <- abs(PUN_db_wide_h$h17 - mean_h17)

PUN_db_wide_h$Dev19 <- abs(PUN_db_wide_h$h19 - mean_h19)

plt_Dev2 <- ggplot(PUN_db_wide_h , aes(x = DateReal, y = Dev2))+
  geom_line(color="blue")+
  xlab("Years")+
  ylab("Dev")+
  ggtitle("Deviation from the Mean Price (2 AM)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5), legend.position = "top")

plt_Dev9 <- ggplot(PUN_db_wide_h , aes(x=DateReal, y=Dev9))+
  geom_line(color="green")+
  xlab("Years")+
  ylab("Dev")+
  ggtitle("Deviation from the Mean Price (9 AM)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))

plt_Dev11 <- ggplot(PUN_db_wide_h , aes(x=DateReal, y=Dev11))+
  geom_line(color="red")+
  xlab("Years")+
  ylab("Dev")+
  ggtitle("Deviation from the Mean Price (11 AM)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))

plt_Dev17 <- ggplot(PUN_db_wide_h , aes(x=DateReal, y=Dev17))+
  geom_line(color="orange")+
  xlab("Years")+
  ylab("Dev")+
  ggtitle("Deviation from the Mean Price (5 PM)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))

plt_Dev19 <- ggplot(PUN_db_wide_h , aes(x=DateReal, y=Dev19))+
  geom_line(color="cyan3")+
  xlab("Years")+
  ylab("Dev")+
  ggtitle("Deviation from the Mean Price (7 PM)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))

grid.arrange(plt_Dev2, plt_Dev9, plt_Dev11, plt_Dev17, plt_Dev19, nrow = 5)
```

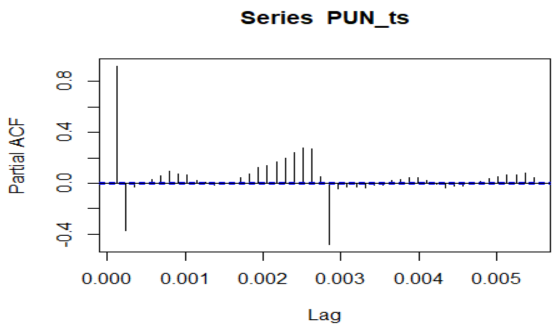


EDA: Stationarity Check of the Hourly Series

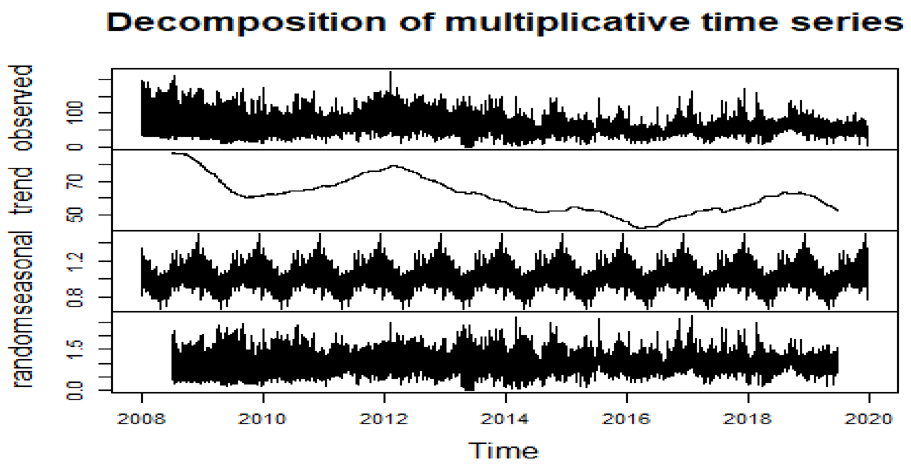
```
pp.test(PUN_ts, alternative = c("s"), type = c("Z(alpha)", "Z(t_alpha)"), lshort = TRUE)

#p-value is 0.01, so there is no unit root
-> PUN is stationary
## Phillips-Perron Unit Root Test
## data:  PUN_ts
## Dickey-Fuller Z(alpha) = -5761.1, Truncation lag parameter = 22,
## p-value = 0.01
## alternative hypothesis: stationary

summary(ur.kpss(PUN_ts))

#PUN is not stationary according to KPSS test 1
## #######################
## # KPSS Unit Root Test #
## #######################
## Test is of type: mu with 22 lags.
## Value of test-statistic is: 145.3911
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739

summary(ur.kpss(PUN_ts, lags = "short"))

#pun is not stationary according to KPSS test 2
## #######################
## # KPSS Unit Root Test #
## #######################
## Test is of type: mu with 22 lags.
## Value of test-statistic is: 145.3911
```

```
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739

kpss.test(PUN_ts, null = "Level", lshort = TRUE)

#pun is not stationary according to KPSS test 3
##  KPSS Test for Level Stationarity
##
## data:  PUN_ts
## KPSS Level = 145.39, Truncation lag parameter = 22, p-value = 0.01

adf.test(PUN_ts) #this is the standard ADF test from the tseries() package
#The p-value is smaller than 0.05 --> #We can reject the null hypothesis. The null hypothesis is
that the data are non-stationary.

##  Augmented Dickey-Fuller Test
##
## data:  PUN_ts
## Dickey-Fuller = -15.141, Lag order = 47, p-value = 0.01
## alternative hypothesis: stationary

ur.df(PUN_ts, type = "none", lags = 24  #ADF test from the urca() package for having control on the
lags and type of ts)  #value of t-stat is: -4.1668 --> p-value is less that 0.01 -->
we can reject the null hypothesis
## ###############################################################
## # Augmented Dickey-Fuller Test Unit Root / Cointegration Test #
## ###############################################################
##
## The value of the test statistic is: -4.1668

acf(PUN_ts, lag.max=24*2)

#with seasonal patterns (which electricity does have), order of lag     should be 2*m, where m is
the periodicity

pacf(PUN_ts, lag.max = 24*2)
```





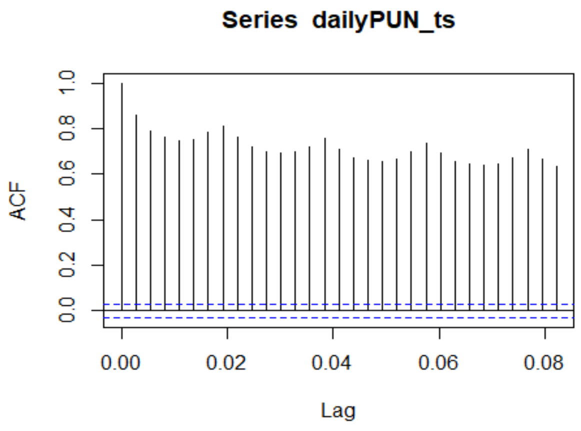
```
dec_pun <- decompose(PUN_ts, type="mult")  #decomposition of elements of the ts of electricity
prices..
plot(dec_pun)
```

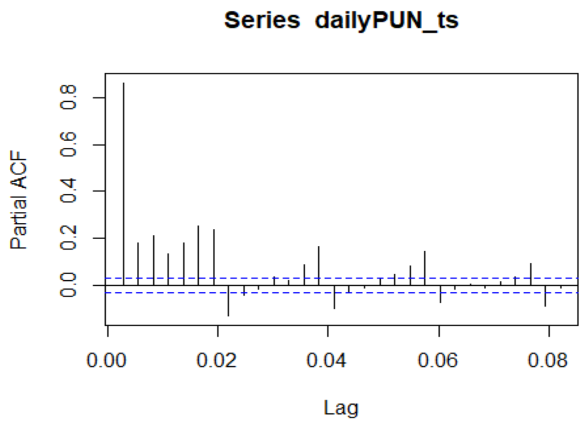


EDA: Stationarity Check on Daily AVG TS

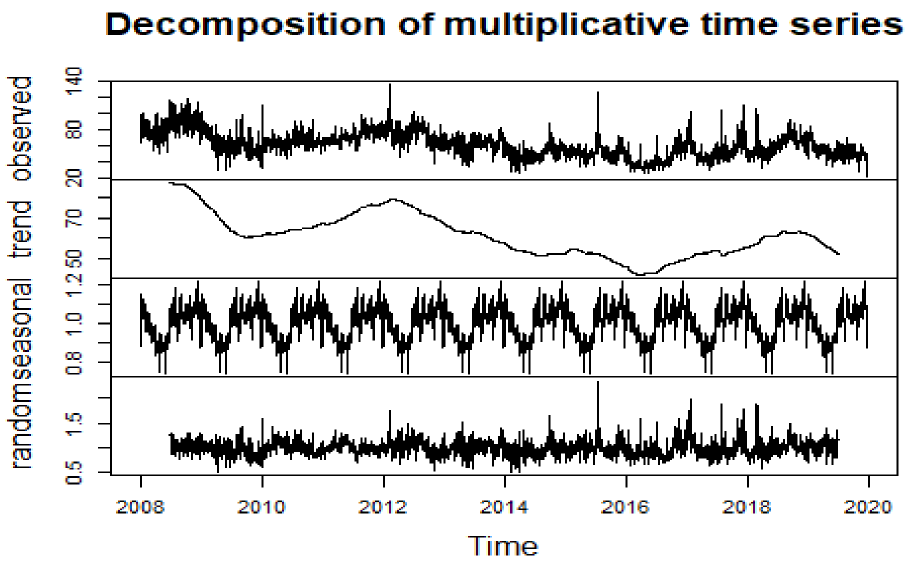
```
pp.test(dailyPUN_ts, alternative = c("s"),
        type = c("Z(alpha)", "Z(t_alpha)"), lshort = TRUE)

#p-value is 0.01, so there is no unit root -> dailyPUN_ts is stationary

##  Phillips-Perron Unit Root Test
## data:  dailyPUN_ts
## Dickey-Fuller Z(alpha) = -867.09, Truncation lag parameter = 10,
## p-value = 0.01
## alternative hypothesis: stationary

summary(ur.kpss(dailyPUN_ts)) #dailyPUN_ts is not stationary according to KPSS test
## #######################
## # KPSS Unit Root Test #
## #######################
## Test is of type: mu with 10 lags.
## Value of test-statistic is: 15.5315
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739

adf.test(dailyPUN_ts)

#this is the standard ADF test from the tseries() package
#The p-value is smaller than 0.05 --> We can reject the null hypothesis. The null hypothesis is that
the data are non-stationary.
##  Augmented Dickey-Fuller Test
##
## data:  dailyPUN_ts
## Dickey-Fuller = -5.812, Lag order = 16, p-value = 0.01
## alternative hypothesis: stationary

ur.df(dailyPUN_ts, type = "none", lags = 12)
#ADF test from the urca() package for having control on the lags and type of ts value of t-stat is: -
1.4064  --> p-value is less that 0.01 --> we can reject the null hypothesis
## ###############################################################
## # Augmented Dickey-Fuller Test Unit Root / Cointegration Test #
## ###############################################################
##
## The value of the test statistic is: -1.4064
```

```r
acf(dailyPUN_ts, lag.max=30)

pacf(dailyPUN_ts, lag.max = 30)
```





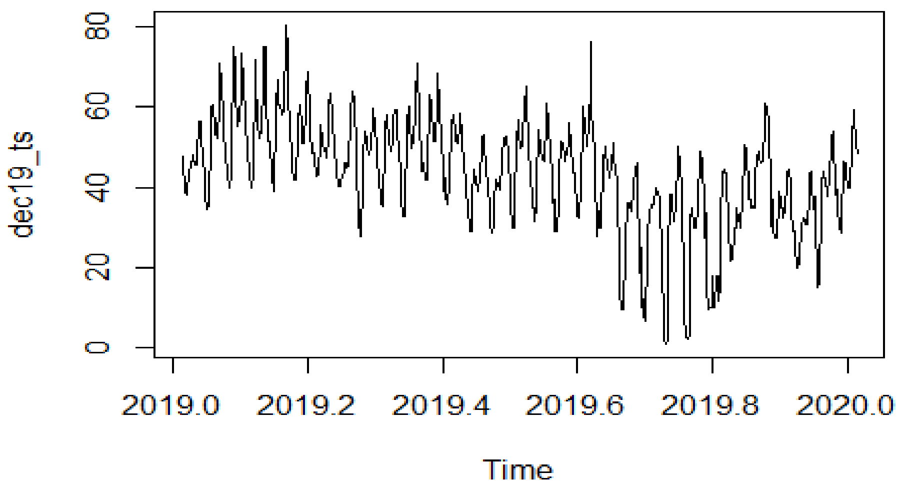
```r
dec_pun_daily <- decompose(dailyPUN_ts, type="mult")  #decomposition of elements of the ts of electricity                                              prices..
plot(dec_pun_daily)
```



NOWCASTING: sARIMA on the HOURLY SUBSET

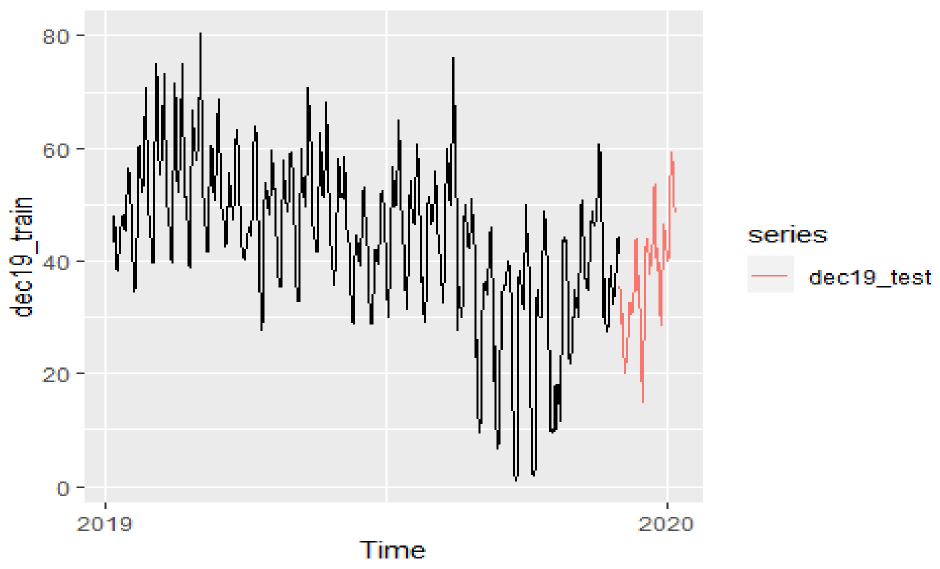
We first run *auto.arima()* and then adapt a seasonal pattern employing a minimum of domain knoledge. Looking at the hourly subset employed for modelling:

```r
ts.plot(dec19_ts)
```



Splitting the series in training and test set

```r
dec19_train <- head(dec19_ts, round(length(dec19_ts) * 0.9))

h <- length(dec19_ts) - length(dec19_train)

dec19_test <- tail(dec19_ts, h)

autoplot(dec19_train) + autolayer(dec19_test)
```

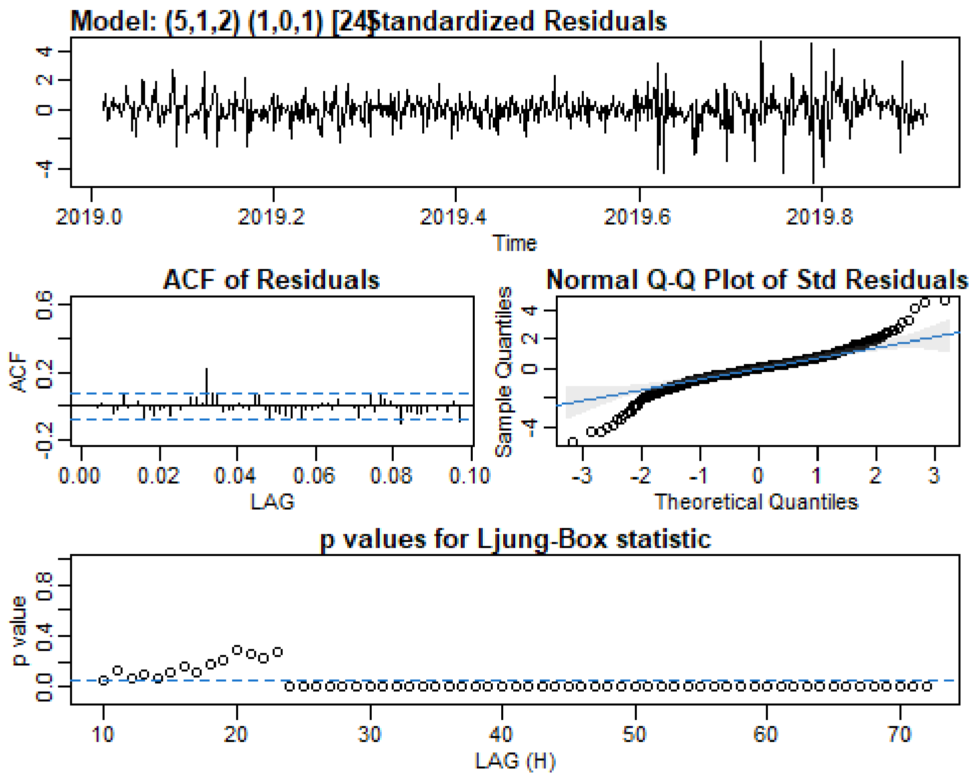


Running *auto.arima()* suggest to use **ARIMA(5,1,2)** as a starting point..

```
auto.arima(dec19_train)  #suggested: ARIMA(5,1,2)

## Series: dec19_train
## ARIMA(5,1,2)
##
## Coefficients:
##          ar1     ar2      ar3      ar4     ar5      ma1      ma2
##       0.4642  0.7226  -0.2587  -0.2916  0.0514  -0.2271  -0.7052
## s.e.  0.1767  0.1986   0.0560   0.0411  0.0538   0.1742   0.1645
##
## sigma^2 estimated as 15.62:  log likelihood=-1866
## AIC=3748   AICc=3748.22   BIC=3784.05
```

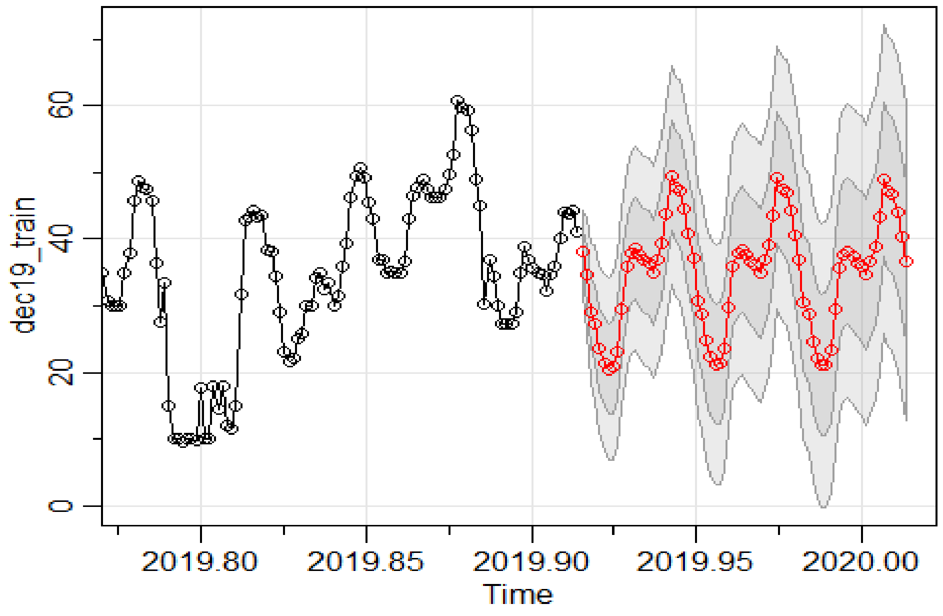
Following auto.arima's suggestion and adding the concept of seasonality: looking at the model..

```
sARIMA_dec19.train <- sarima(dec19_train, p = 5, d = 1, q = 2, P = 1, D = 0, Q = 1, S = 24, details
= TRUE, xreg = NULL, Model = TRUE, fixed = NULL, no.constant = TRUE)
```



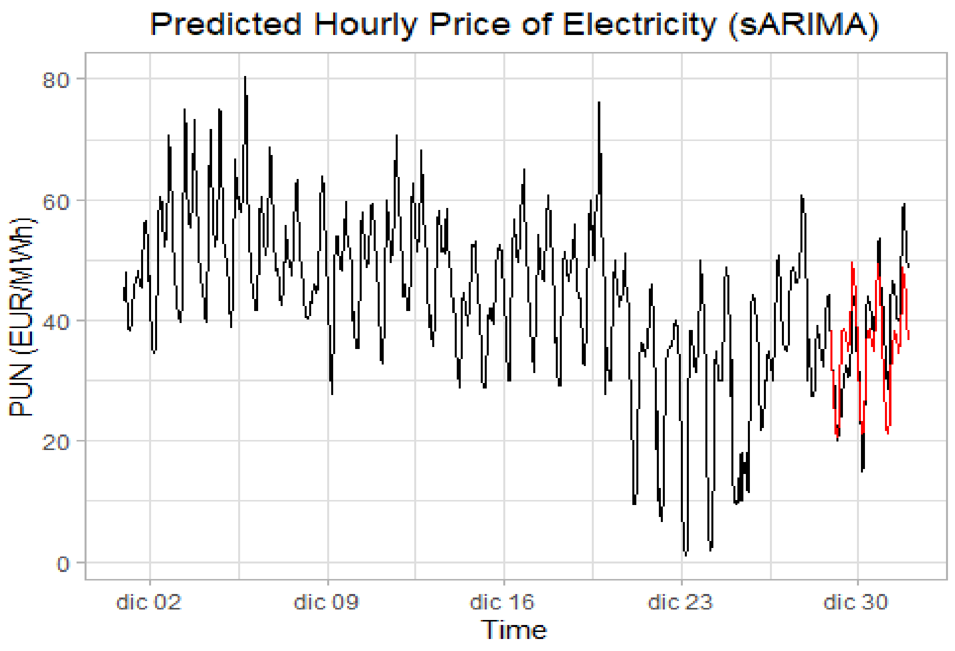
..and forecasting

```
fcst_dec19.sar <- sarima.for(dec19_train, n.ahead = 74 , p = 5, d = 1, q = 2, P = 1, D = 0, Q = 1, S
= 24, xreg = NULL, plot = TRUE, plot.all = FALSE, fixed = NULL, no.constant = TRUE)
```



Plotting sARIMA's forecast against actual values: the results seem satisfying.

```
dec19_df_conf <- slice(dec19, 671:744)

dec19_df_conf$Pred <- cbind(fcst_dec19.sar$pred)

ggplot() +
  geom_line(data = dec19, aes(x = DateTime, y = PUN)) +
  geom_line(data = dec19_df_conf, aes(x = DateTime , y = Pred), color='red')+
  theme_light()+
  xlab("Time")+
  ylab("PUN (EUR/MWh)")+
  ggtitle("Predicted Hourly Price of Electricity (sARIMA)")+
  theme(plot.title = element_text(hjust = 0.5))
```



Computing **Performance Metrics** for sARIMA model..

```
sARIMA_dec19 <- sarima(dec19_ts, p = 5, d = 1, q = 2, P = 1, D = 0, Q = 1, S = 24, details = FALSE,
xreg = NULL, Model = FALSE, fixed = NULL, no.constant = TRUE)

MAE_sARIMA_hourly <- mean(abs(residuals(sARIMA_dec19$fit)))
MAPE_sARIMA_hourly <- mean(100*abs(residuals(sARIMA_dec19$fit))/dec19_ts)
RMSE_sARIMA_hourly <- sqrt(mean(residuals(sARIMA_dec19$fit)^2))
```

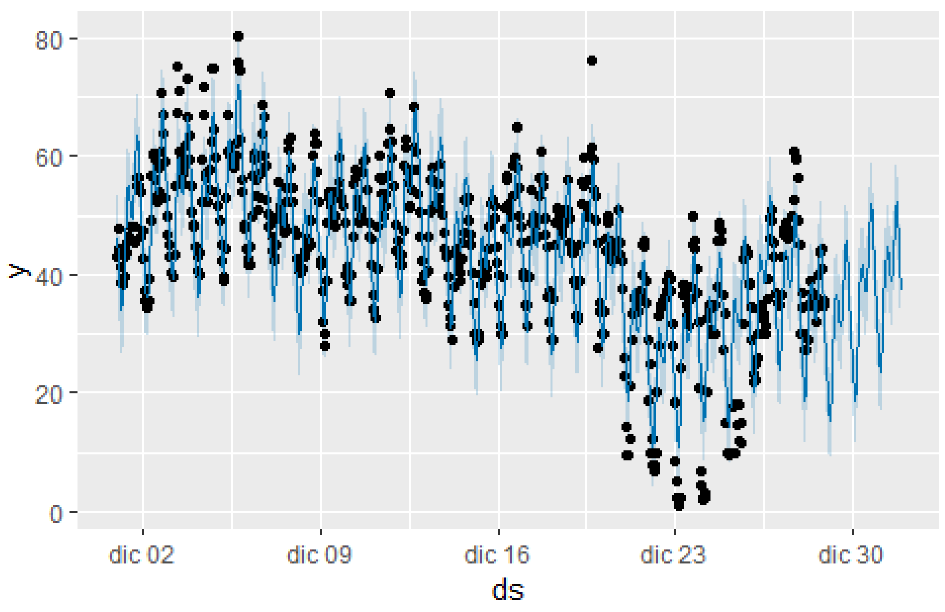
NOWCASTING: Prophet on the HOURLY SUBSET

We can now run the *prophet* package on the same series and verify which method performs better.

```
dec_19_df <- dec19[1:672,]

HourlyPUN_df <- data.frame(dec_19_df$DateTime, dec_19_df$PUN)

colnames(HourlyPUN_df) <- c("ds", "y")   #Prophet need exactly those two names for variables

mod_prophet_hourly <- prophet(df = HourlyPUN_df, growth = "linear", yearly.seasonality = FALSE,
weekly.seasonality = TRUE, daily.seasonality = TRUE, seasonality.mode = "additive", fit = TRUE )
#Prophet model
```

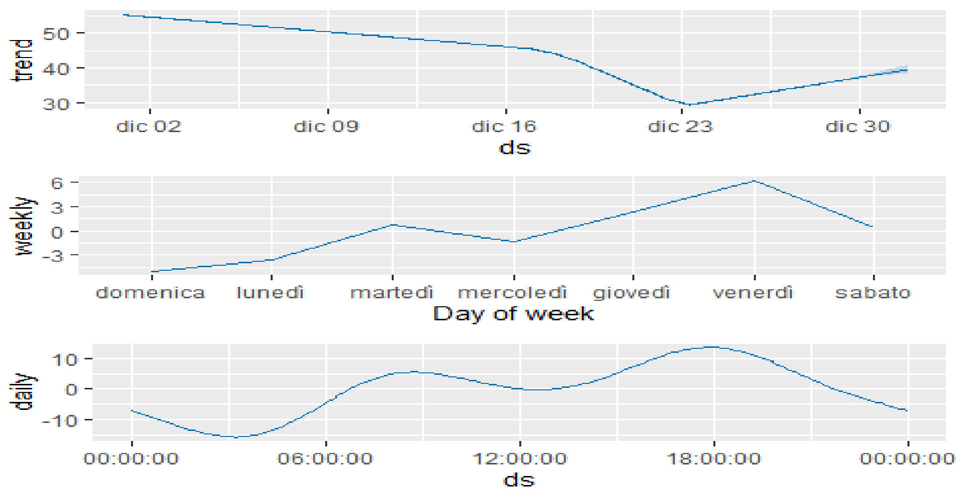
Prophet method needs a data frame containing future dates for storing Prophet's predictions. After creating it, we are able to obtain predictions from the model

```
future_dec <- make_future_dataframe(mod_prophet_hourly, periods = 24*3, freq = 3600)

fcst_hourly_prophet <- predict(mod_prophet_hourly, future_dec)

plot(mod_prophet_hourly, fcst_hourly_prophet)
```



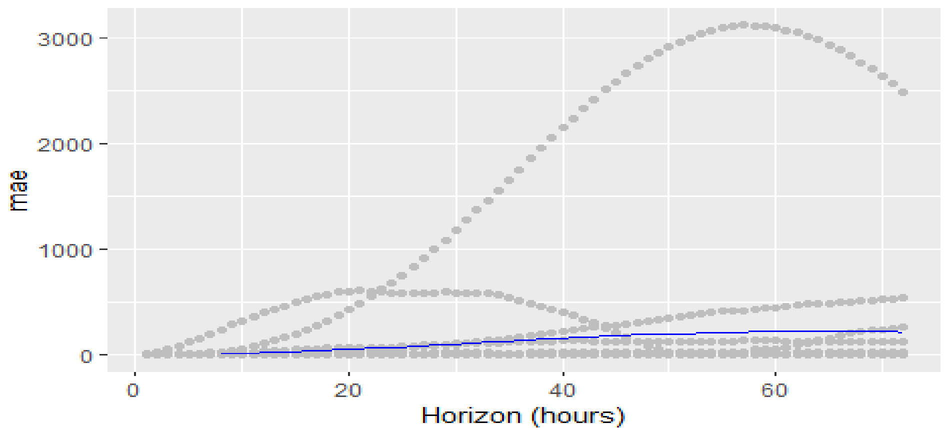
We can also plot the components of the model

```
prophet_plot_components(mod_prophet_hourly, fcst_hourly_prophet)
```

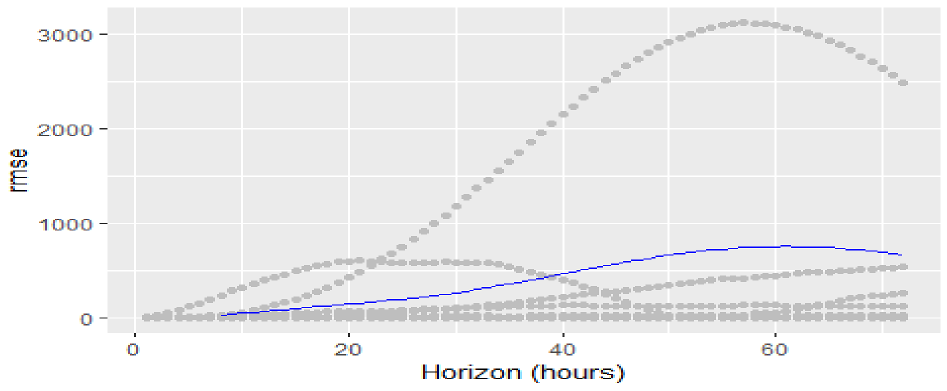


Prophet method provide a useful function for **cross validation**:

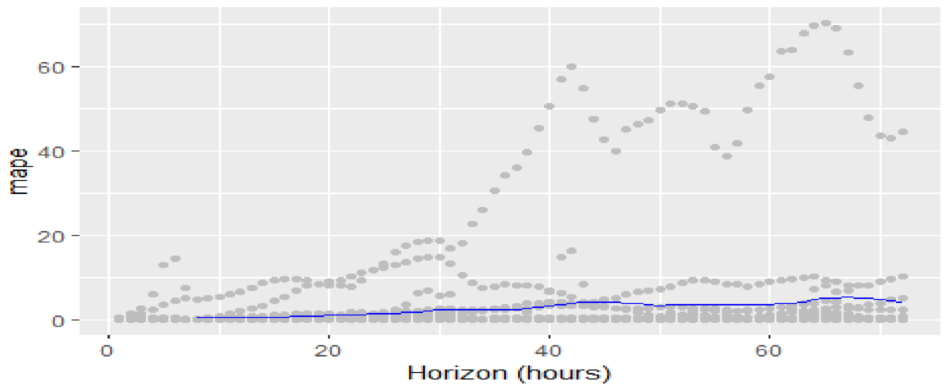
```
prophet_cv_hourly <- cross_validation(mod_prophet_hourly, initial = length(HourlyPUN_df)*0.8,
horizon = 24*3, units = 'hours')

head(prophet_cv_hourly)

##          y                  ds       yhat  yhat_lower yhat_upper
## 1 47.43000 2019-12-01 23:00:00   34.39265   33.57541   35.22474
## 2 42.69833 2019-12-02 00:00:00   13.22312   12.31418   14.04067
## 3 37.22000 2019-12-02 01:00:00  -16.47703  -17.34905  -15.61559
## 4 35.40000 2019-12-02 02:00:00  -51.60151  -52.45623  -50.74368
## 5 34.45818 2019-12-02 03:00:00  -88.15489  -89.03864  -87.28503
## 6 35.40000 2019-12-02 04:00:00 -123.76263 -124.59194 -122.90375
##                cutoff
## 1 2019-12-01 22:00:00
## 2 2019-12-01 22:00:00
## 3 2019-12-01 22:00:00
## 4 2019-12-01 22:00:00
## 5 2019-12-01 22:00:00
## 6 2019-12-01 22:00:00

plot_cross_validation_metric(prophet_cv_hourly, "mae")
```



```r
plot_cross_validation_metric(prophet_cv_hourly, "rmse")
```



```r
plot_cross_validation_metric(prophet_cv_hourly, "mape")
```

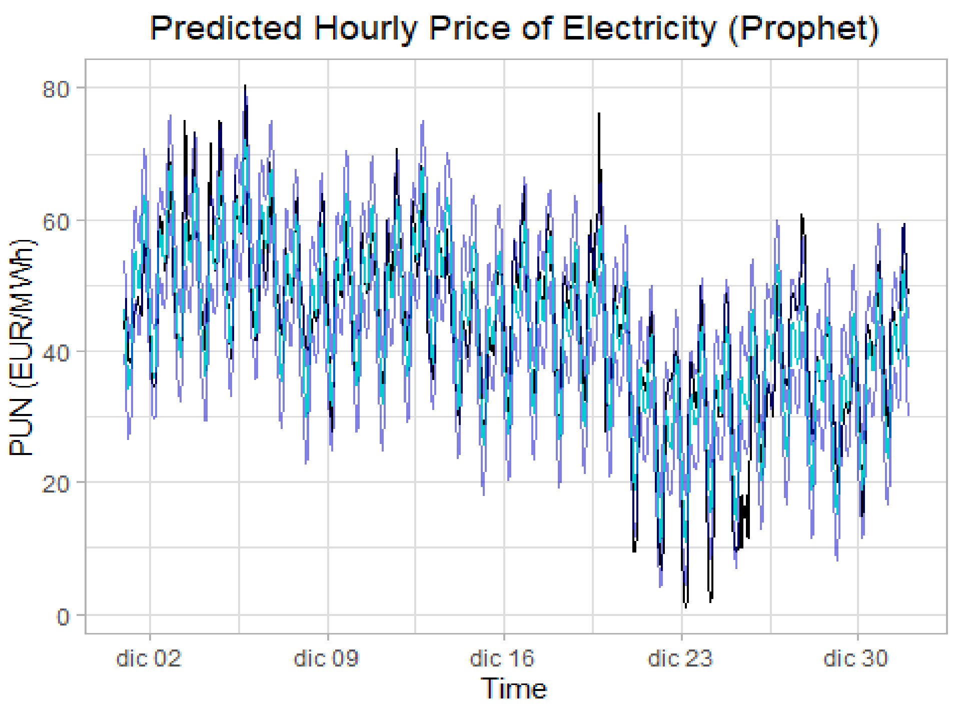


Prophet also permits to output the performance metrics from its cross validation method:

```
prophet_metrics_h <- performance_metrics(prophet_cv_hourly)

head(prophet_metrics_h)

##    horizon      mse     rmse      mae      mape  coverage
## 1  8 hours 1302.117 36.08485 12.84257 0.6712471 0.5814851
## 2  9 hours 1998.609 44.70581 15.63752 0.7073527 0.5486982
## 3 10 hours 2909.874 53.94325 18.79701 0.7351119 0.4995178
## 4 11 hours 4029.557 63.47879 22.09751 0.7387834 0.4667310
## 5 12 hours 5351.241 73.15218 25.50974 0.6909965 0.4570878
## 6 13 hours 6868.181 82.87449 29.16519 0.6345440 0.4378014

MAPE_prophet_h <- mean(prophet_metrics_h$mape)
MAE_prophet_h <- mean(prophet_metrics_h$mae)
RMSE_prophet_h <-  mean(prophet_metrics_h$rmse)
```

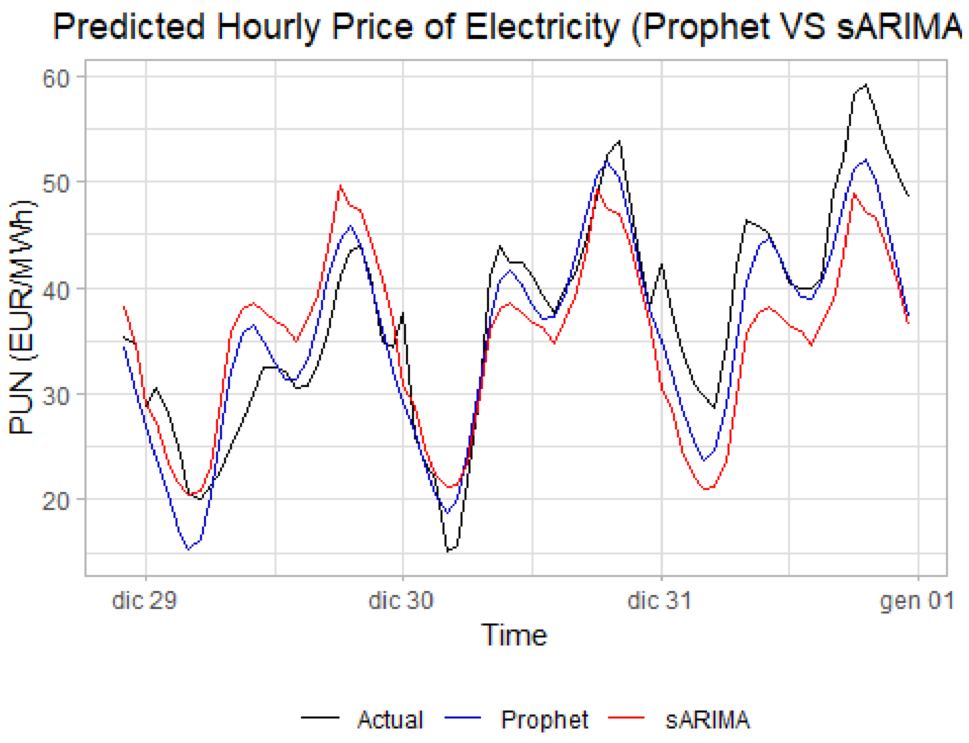
Actual Values and Fitted values, with confidence intervals:

```
ggplot()+
  geom_line(data = dec19, aes(x = DateTime, y = PUN))+
  geom_line(data = fcst_hourly_prophet, aes(x = ds, y = yhat), color='cyan3')+
  geom_line(data = fcst_hourly_prophet, aes(x = ds, y = yhat_lower), color='mediumblue', alpha=0.5)+
  geom_line(data = fcst_hourly_prophet, aes(x = ds, y = yhat_upper), color='mediumblue', alpha=0.5)+
  theme_light()+
  xlab("Time")+
  ylab("PUN (EUR/MWh)")+
  ggtitle("Predicted Hourly Price of Electricity (Prophet)")+
  theme(plot.title = element_text(hjust = 0.5))
```



NOWCASTING: PROPHET vs sARIMA

```
fcst_hourly_prophet_sub <- slice(fcst_hourly_prophet, 671:744)

dec19_df_conf$Pred_Prophet <- fcst_hourly_prophet_sub$yhat

ggplot(data = dec19_df_conf, aes(x = DateTime)) +
  geom_line(aes(y = PUN, colour = "Actual")) +
  geom_line(aes(y = Pred_Prophet, colour = "Prophet")) +
  geom_line(aes(y = Pred, colour = "sARIMA")) +
  scale_colour_manual("",
                      breaks = c("Actual", "Prophet", "sARIMA"),
                      values = c("black", "mediumblue", "red")) +
  xlab("Time") +
  ylab("PUN (EUR/MWh)")+
  ggtitle("Predicted Hourly Price of Electricity (Prophet VS sARIMA)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5), legend.position = "bottom")
```



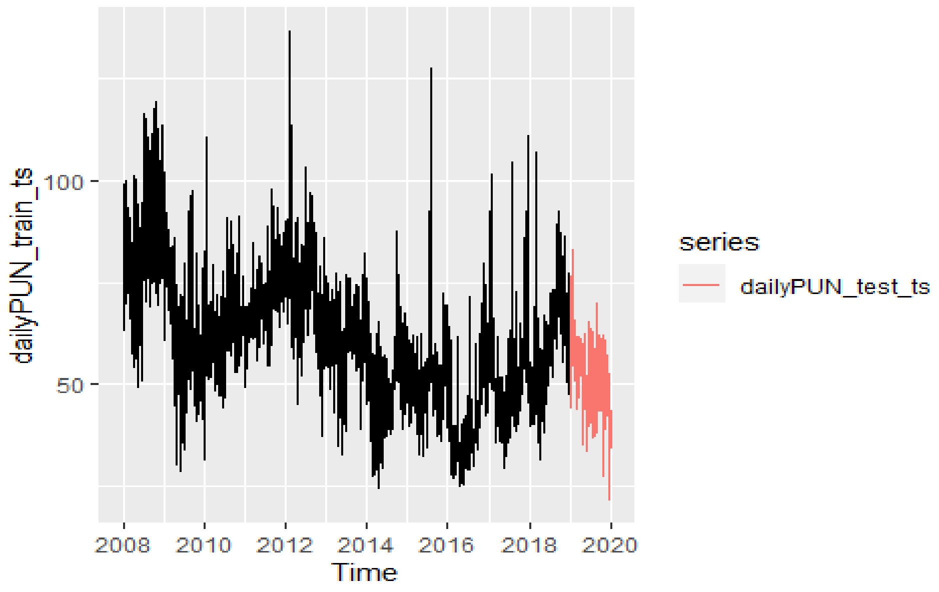
As can be seen by the figure, we can obtain similar results between Prophet and sARIMA models; what gives the edge to Prophet however, is the extreme user-friendliesness and time saving capabilities: sARIMA has to be tuned manually and necessitates of way more domain knowledge than Prophet does.

LONG RUN FORECASTING: sARIMA on daily data

Since sARIMA and Prophet both perform well in the task of forecasting day-ahead hourly prices, we try to test them with the task of forecasting daily prices one year ahead. We start with sARIMA.

Splitting in training and test set..

```
dailyPUN_train <- subset(PUN_db_wide, Year <= 2018)

dailyPUN_test <- subset(PUN_db_wide, Year > 2018)

dailyPUN_train_ts <- ts(dailyPUN_train$DailyPun, start = c(2008,1,1), freq = 365.25)

dailyPUN_test_ts <- ts(dailyPUN_test$DailyPun,  start = c(2019,1,1), freq = 365)

autoplot(dailyPUN_train_ts) + autolayer(dailyPUN_test_ts)
```

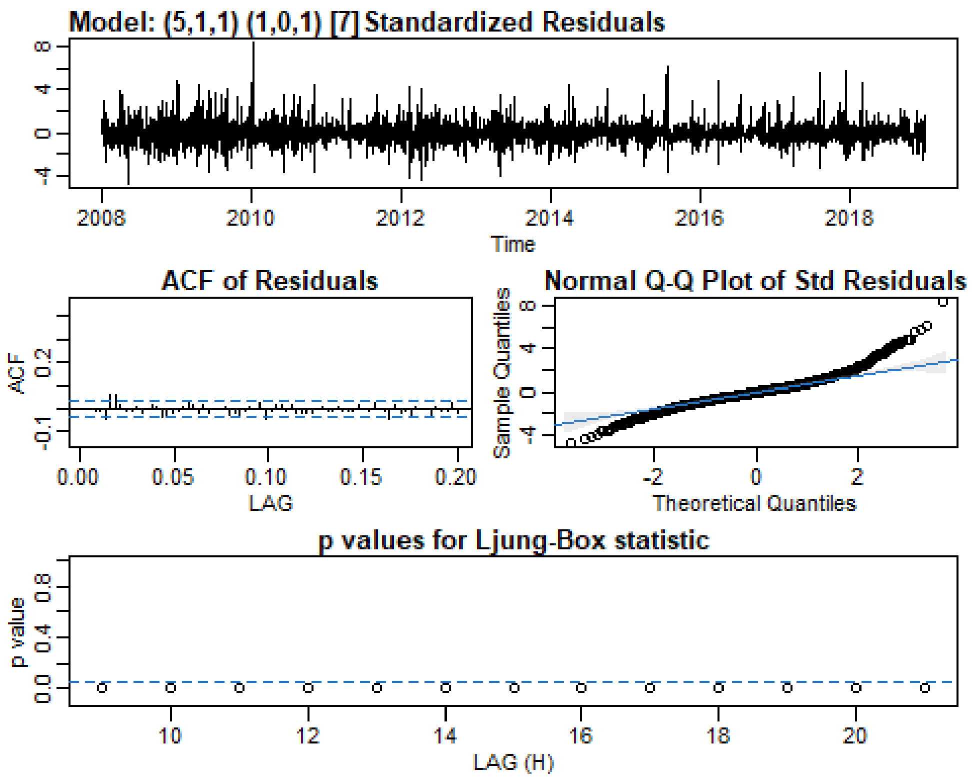


..and using *auto.arima()* on the training set.

```
auto.arima(dailyPUN_train_ts, seasonal = TRUE, stepwise = TRUE)

## Series: dailyPUN_train_ts
## ARIMA(5,1,1) with drift
##
## Coefficients:
##           ar1      ar2      ar3      ar4      ar5      ma1    drift
##       -0.0702  -0.2828  -0.2022  -0.2258  -0.2168  -0.4035  -0.0043
## s.e.   0.0344   0.0194   0.0195   0.0172   0.0184   0.0332   0.0341
##
## sigma^2 estimated as 52.43:  log likelihood=-13649.31
## AIC=27314.61   AICc=27314.65   BIC=27365
```

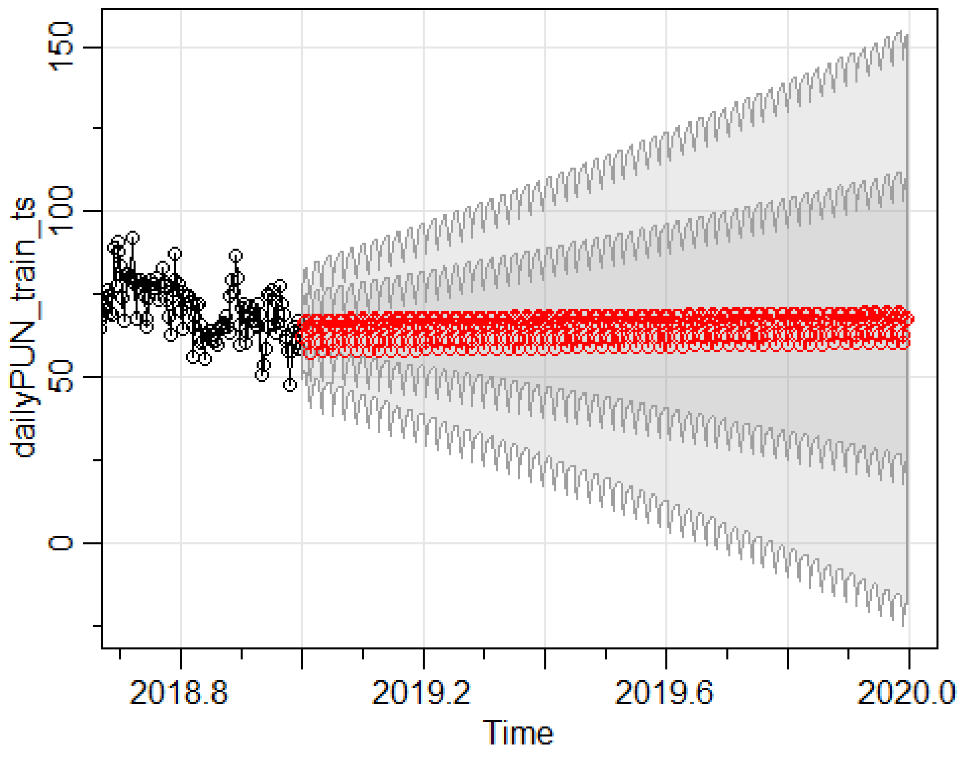
We run the suggested model on the set, adding some seasonality to it:

```
sARIMA_daily.train <- sarima(dailyPUN_train_ts, p = 5, d = 1, q = 1, P = 1, D = 0, Q = 1, S = 7,
details = TRUE, xreg = NULL, Model = TRUE, fixed = NULL, no.constant = TRUE)
```

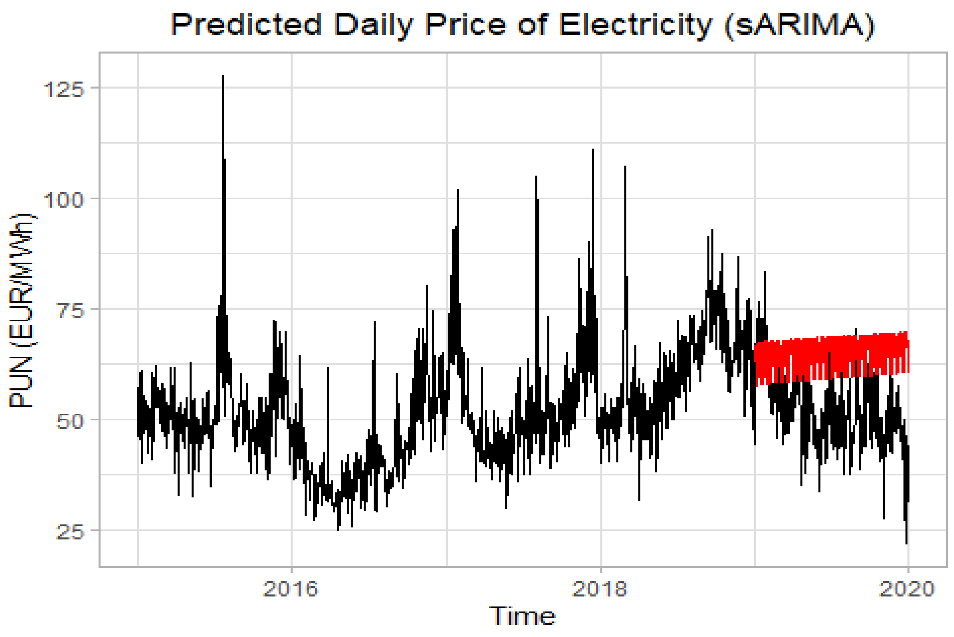


Long run forecasting with daily data: we already got the impression of something being off about long-run daily predictions of PUN..

```
fcst_daily.test <- sarima.for(dailyPUN_train_ts, n.ahead = 365 , p = 5, d = 1, q = 1, P = 1, D = 0,
Q = 1, S = 7, xreg = NULL, plot = TRUE, plot.all = FALSE, fixed = NULL, no.constant = TRUE)
```



```
fcst_daily.test_df <- data.frame(fcst_daily.test$pred)

fcst_daily.test_df$Date <- seq(ymd("2019-1-1"), ymd("2019-12-31"), by = "days")

colnames(fcst_daily.test_df) <- c("Pred_PUN", "Date")

ggplot() +
  geom_line(data = subset(PUN_db_wide, Year >= 2015), aes(x = DateReal, y = DailyPun)) +
  geom_line(data = fcst_daily.test_df, aes(x = Date , y = Pred_PUN), color='red')+
  theme_light()+
  xlab("Time")+
  ylab("PUN (EUR/MWh)")+
  ggtitle("Predicted Daily Price of Electricity (sARIMA)")+
  theme(plot.title = element_text(hjust = 0.5))
```



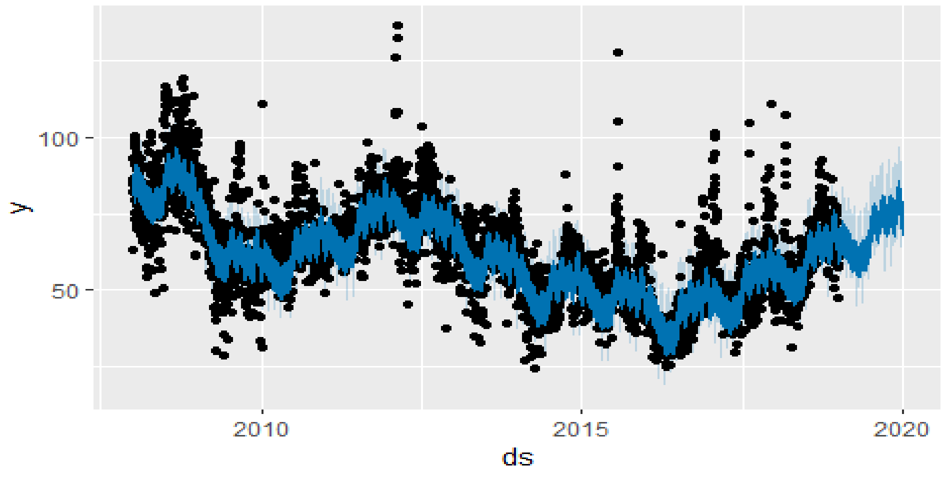
sARIMA modelling, applied at the whole daily series as it is, does not provide us with acceptable results. We store the Performance Metrics and then go on testing Prophet against the same series.

```
sARIMA_daily <- sarima(dailyPUN_ts, p = 5, d = 1, q = 1, P = 1, D = 0, Q = 1, S = 7, details =
FALSE, xreg = NULL, Model = FALSE, fixed = NULL, no.constant = TRUE)

MAE_sARIMA_daily <- mean(abs(residuals(sARIMA_daily$fit)))
MAPE_sARIMA_daily <- mean(100*abs(residuals(sARIMA_daily$fit))/dailyPUN_ts)
RMSE_sARIMA_daily <- sqrt(mean(residuals(sARIMA_daily$fit)^2))
```

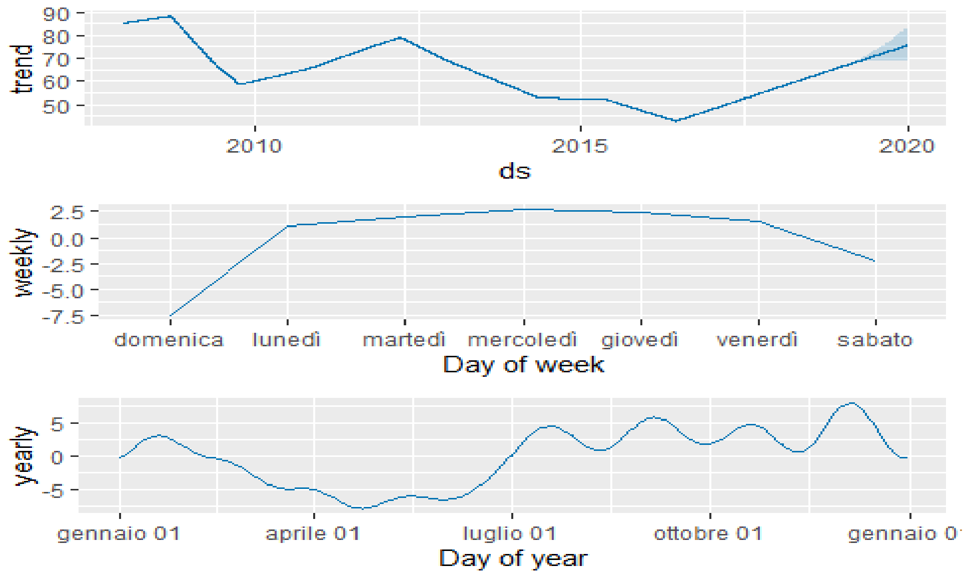
From previous figure it should be obvious that sARIMA modelling is not enough for forecasting 365 days ahead, because of the non-linearities the ARIMA model class cannot identify. Will Prophet perform any better?

LONG RUN FORECASTING: PROPHET on daily data

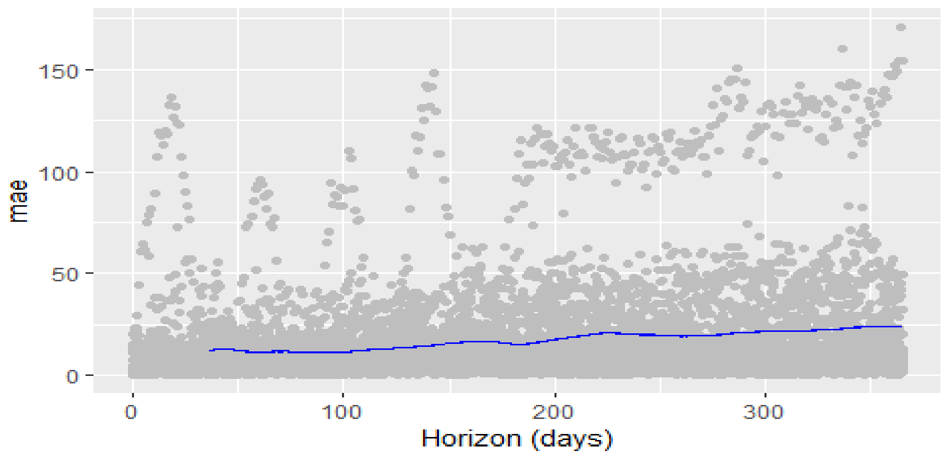
```
DailyPUN_train_df <- data.frame(dailyPUN_train$DateReal, dailyPUN_train$DailyPun)

colnames(DailyPUN_train_df) <- c("ds", "y")

mod_prophet_daily <- prophet(df = DailyPUN_train_df, growth = "linear", yearly.seasonality = TRUE,
weekly.seasonality = TRUE, daily.seasonality = "auto", holidays = NULL, seasonality.mode =
"additive", fit = TRUE )

future_d <- make_future_dataframe(mod_prophet_daily, periods = 365, freq = "day")

fcst_daily_prophet <- predict(mod_prophet_daily, future_d)

plot(mod_prophet_daily, fcst_daily_prophet)
```



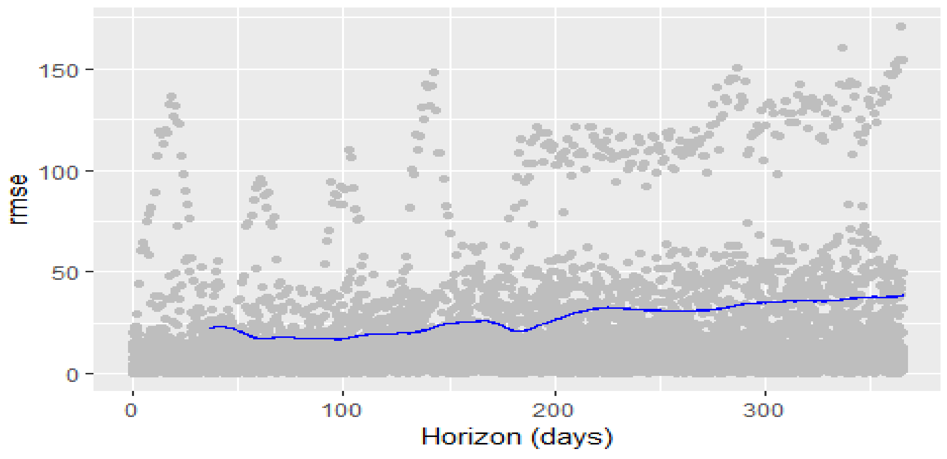
```
prophet_plot_components(mod_prophet_daily, fcst_daily_prophet)
```



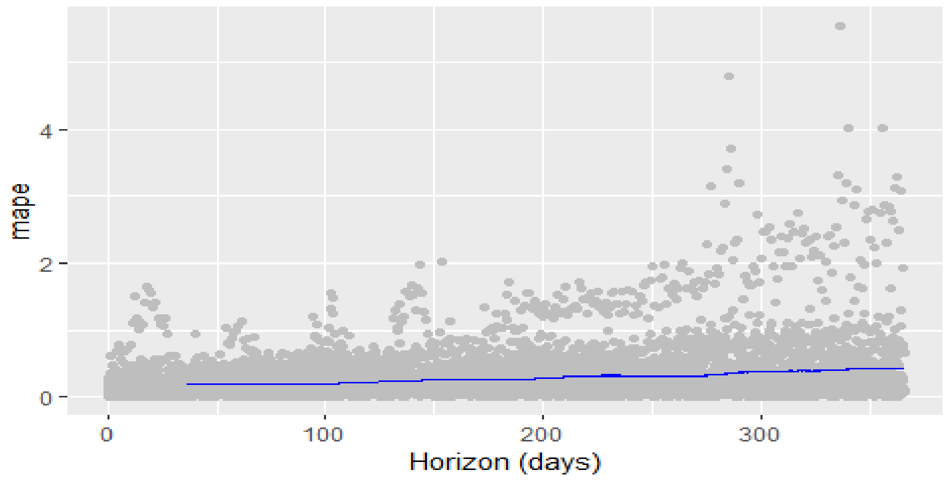
```r
prophet_cv_daily <- cross_validation(mod_prophet_daily, initial = length(DailyPUN_train_df)*0.8,
horizon = 365.25, units = 'days')

head(prophet_cv_daily)

##           y         ds     yhat yhat_lower yhat_upper              cutoff
## 1 116.49714 2008-07-01 129.4688   121.3058   137.1893 2008-06-30 21:00:00
## 2 114.25514 2008-07-02 137.3492   128.8176   146.0766 2008-06-30 21:00:00
## 3 112.41138 2008-07-03 141.4542   132.6683   149.2483 2008-06-30 21:00:00
## 4 103.88157 2008-07-04 148.3361   139.8793   156.6445 2008-06-30 21:00:00
## 5  89.60028 2008-07-05 150.7524   142.4716   159.1647 2008-06-30 21:00:00
## 6  83.24725 2008-07-06 147.8932   139.7603   155.9641 2008-06-30 21:00:00

plot_cross_validation_metric(prophet_cv_daily, "mae")
```



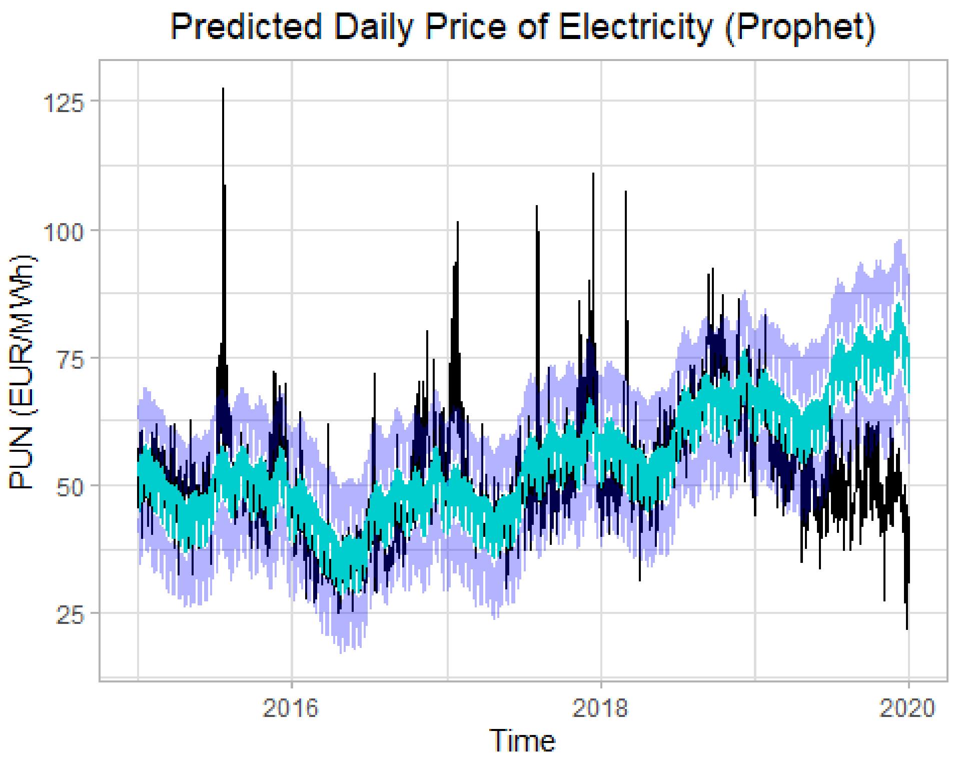
```r
plot_cross_validation_metric(prophet_cv_daily, "rmse")
```



```r
plot_cross_validation_metric(prophet_cv_daily, "mape")
```

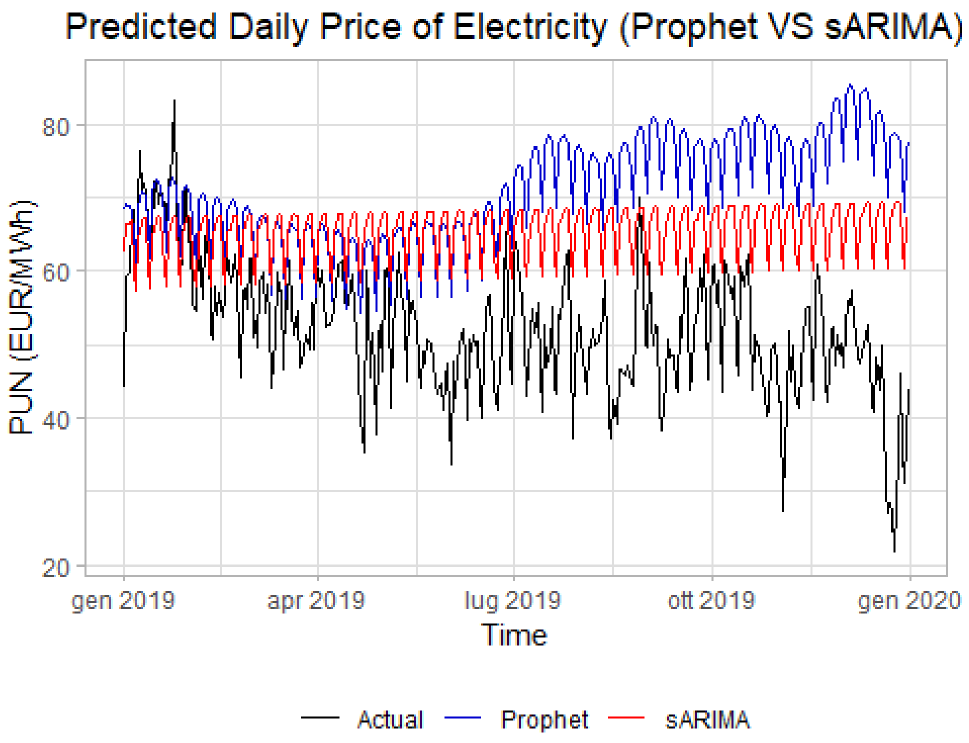


```
prophet_metrics_d <- performance_metrics(prophet_cv_daily)

head(prophet_metrics_d)

##     horizon      mse     rmse      mae      mape  coverage
## 1 876 hours 525.1325 22.91577 12.35325 0.1774979 0.6762557
## 2 879 hours 524.9195 22.91112 12.34104 0.1775225 0.6767123
## 3 882 hours 526.0414 22.93559 12.38807 0.1781381 0.6753425
## 4 885 hours 526.3136 22.94153 12.40925 0.1784345 0.6739726
## 5 888 hours 525.8022 22.93038 12.38244 0.1780422 0.6757991
## 6 891 hours 526.8511 22.95324 12.40612 0.1781945 0.6753425

MAE_prophet_d <- mean(prophet_metrics_d$mae)
MAPE_prophet_d <- mean(prophet_metrics_d$mape)
RMSE_prophet_d <- mean(prophet_metrics_d$rmse)

fcst_daily_prophet$Year <- year(fcst_daily_prophet$ds)

fcst_daily_prophet$ds <- as.Date(fcst_daily_prophet$ds)

ggplot() +
  geom_line(data = subset(PUN_db_wide, Year >= 2015), aes(x = DateReal, y = DailyPun)) +
  geom_line(data = subset(fcst_daily_prophet, Year >= 2015), aes(x = ds, y = yhat), color='cyan3')+
  geom_line(data = subset(fcst_daily_prophet, Year >= 2015), aes(x = ds, y = yhat_lower),
color='blue', alpha=0.3)+
  geom_line(data = subset(fcst_daily_prophet, Year >= 2015), aes(x = ds, y = yhat_upper),
color='blue', alpha=0.3)+
  theme_light()+
  xlab("Time")+
  ylab("PUN (EUR/MWh)")+
  ggtitle("Predicted Daily Price of Electricity (Prophet)")+
  theme(plot.title = element_text(hjust = 0.5))
```



Prophet performs better than sARIMA on long-run forecasting in terms of evaluation metrics but, both in additive and multiplicative seasonality mode, fails to forecast electricity prices as it should.

```r
dailyPUN_test <- data.frame(dailyPUN_test$DateReal, dailyPUN_test$DailyPun)

colnames(dailyPUN_test) <- c("DateReal", "DailyPun")

dailyPUN_test$Pred_sar <- fcst_daily.test_df$Pred_PUN

prophet_2019_daily <- subset(fcst_daily_prophet, Year >= 2019)

dailyPUN_test$Pred_Prophet <- prophet_2019_daily$yhat

ggplot(data = dailyPUN_test, aes(x = DateReal)) +
  geom_line(aes(y = DailyPun, colour = "Actual")) +
  geom_line(aes(y = Pred_Prophet, colour = "Prophet")) +
  geom_line(aes(y = Pred_sar, colour = "sARIMA")) +
  scale_colour_manual("",
                      breaks = c("Actual", "Prophet", "sARIMA"),
                      values = c("black", "mediumblue", "red")) +
  xlab("Time") +
  ylab("PUN (EUR/MWh)")+
  ggtitle("Predicted Daily Price of Electricity (Prophet VS sARIMA)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5), legend.position = "bottom")
```



Working around the Long-Casting task: for loops and sliding windows

One way we figured out to address the long-casting task and obtain believable predictions is to use a sliding window of one month of daily observations and running our models through it, forecasting three days ahead, dropping the first three observations and repeating. This comes at cost of losing a degree of automation and increasing complexity a little but, as the reader will notice, our strategy is worth exploring.

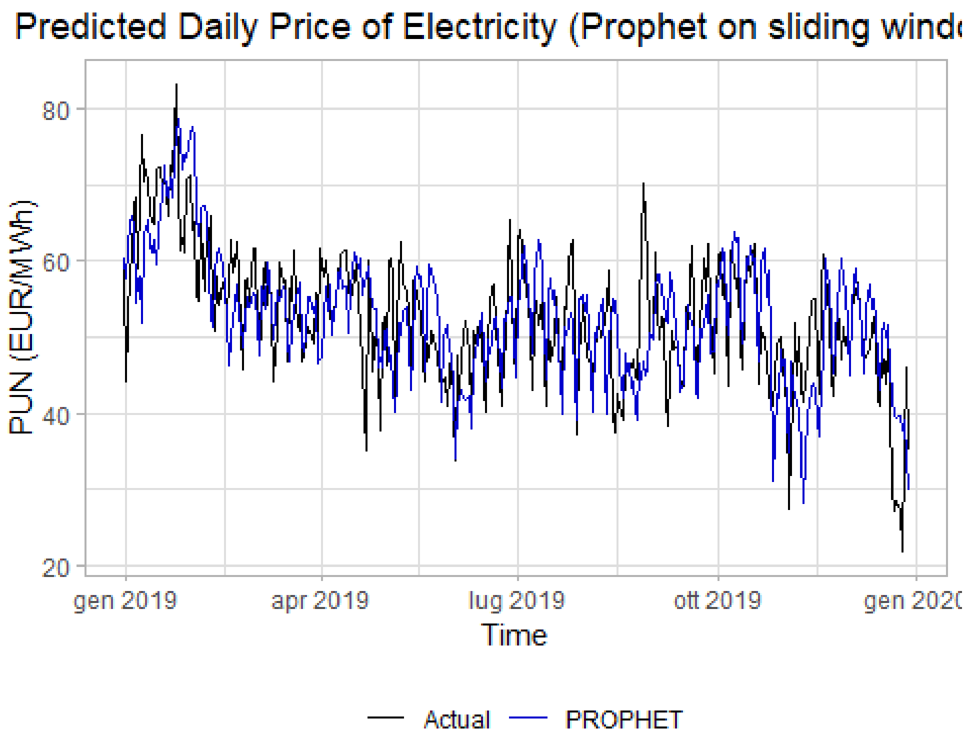
```
db_2018 <- subset(PUN_db_wide, Year == 2018)

db_2018 <- data.frame(db_2018$DateReal, db_2018$DailyPun)

colnames(db_2018) <- c("DateReal", "DailyPun")

db_2018$month <- month(db_2018$DateReal)

dec18 <- subset(db_2018, month == 12)

row.names(dec18) <- NULL

db_2019 <- subset(PUN_db_wide, Year == 2019)

db_2019 <- data.frame(db_2019$DateReal, db_2019$DailyPun)

colnames(db_2019) <- c("DateReal", "DailyPun")

db_2019$month <- month(db_2019$DateReal)

row.names(db_2019) <- NULL

daily_set <- rbind(dec18, db_2019)
```

```r
daily_set <- daily_set[1:2]
```

Prophet on a sliding window

We first initialize a counter and the needed data frames in which we'll store the new predictions. Then we employ a while loop to recursively run prophet. The results can be observed in the graph below.

```r
c <- 0
fcst_test_prophet <- data.frame()
running <- data.frame()
pred_running <- data.frame()

while(c < nrow(daily_set) - 33){

  sample_p <- daily_set[(1+c):(30+c), ]

  colnames(sample_p) <- c("ds", "y")

  mod_prophet_test <- prophet(df = sample_p, growth = "linear", yearly.seasonality = "auto",
weekly.seasonality = TRUE,
                               daily.seasonality = "auto", holidays = NULL, seasonality.mode =
"additive", fit = TRUE )

  future_test <- make_future_dataframe(mod_prophet_test, periods = 3, freq = "day")

  fcst_test_prophet <- predict(mod_prophet_test, future_test)

  running <- fcst_test_prophet[31:33, c("ds", "yhat")] #these are the three rolling predictions..

  pred_running <- rbind(pred_running, running)

  c <- c+3
}

daily_set <- daily_set[31:nrow(daily_set), ]

row.names(daily_set) <- NULL

daily_set <- daily_set[1:363, ]

pred_running <- cbind(pred_running, daily_set$DailyPun)

colnames(pred_running) <- c("date", "prophet_pred", "DailyPUN")

ggplot(data = pred_running, aes(x = date)) +
  geom_line(aes(y = DailyPUN, colour = "Actual")) +
  geom_line(aes(y = prophet_pred, colour = "PROPHET")) +
  scale_colour_manual("",
                      breaks = c("Actual", "PROPHET"),
                      values = c("black", "mediumblue")) +
  xlab("Time") +
  ylab("PUN (EUR/MWh)")+
  ggtitle("Predicted Daily Price of Electricity (Prophet on sliding window)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5), legend.position = "bottom")
```



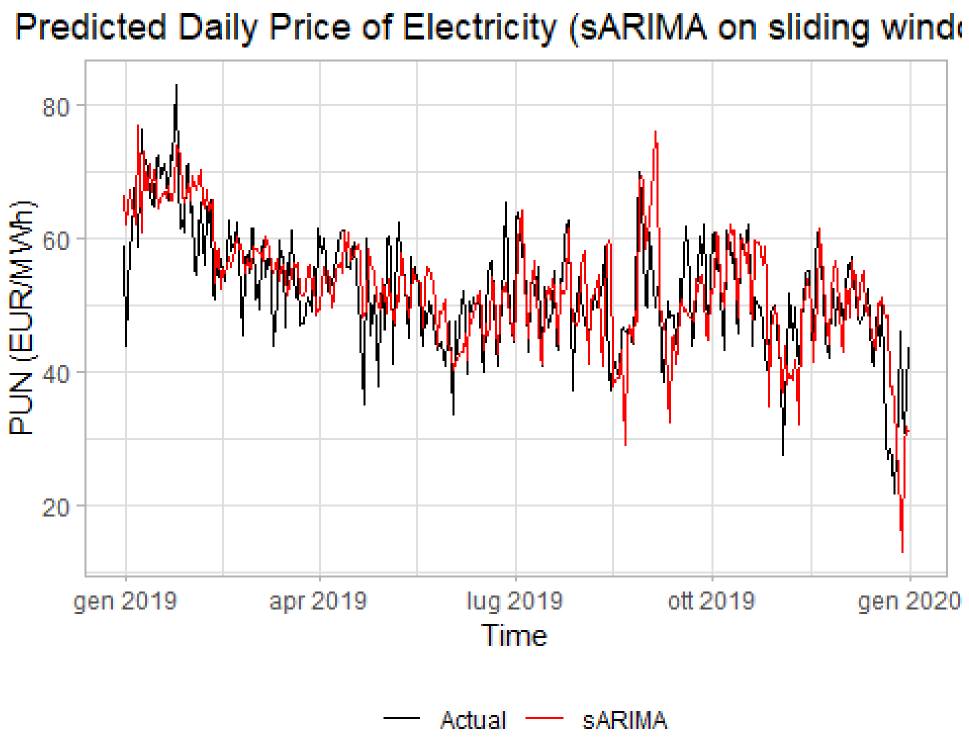
As can be seen, our method works fairly well for prophet. We store the Performance Metrics from the predictions and then procede with a similar method for sARIMA modelling.

```
PROPHET_roll_acc <- accuracy(object = pred_running$prophet_pred, x = pred_running$DailyPUN, test =
NULL)
```

sARIMA on a sliding window of observations

Preparing the subset to run through with sARIMA: here, we don't have the chance to use *auto.arima()* and then decide what to do. Our approach is then to utilize a simple ARIMA(1,1,1)x(1,0,1)[7] since we are running through 31 observations at a time.
Once again, we initialize the ausiliary data frames and then run the for loop, obtaining satisfactory results for our Long-casting purposes.

```
daily_set_sar <- rbind(dec18, db_2019)

daily_set_sar <- daily_set_sar[1:2]

sample_sar <- data.frame()
predictions_sar <- data.frame()

for (i in seq(1, nrow(daily_set_sar)-30, 3)){     #1/3 of actual values

  sample_sar <- daily_set_sar$DailyPun[i:c(29+i)]

  dates <- daily_set_sar$DateReal[i:c(29+i)]

  sample_ts <-as.ts(sample_sar, start_date = dates[1], freq = 365)

  sARIMA <- arima(sample_ts, order=c(1,1,1), seasonal=list(order=c(1,0,1), period=7), include.mean =
FALSE, method="ML")

  sARIMA_fcst <- forecast(sARIMA, h =3)
  predictions_sar <- rbind(predictions_sar, as.data.frame(as.numeric(sARIMA_fcst$mean)))
}

daily_set_sar <- daily_set_sar[31:nrow(daily_set_sar), ]

row.names(daily_set_sar) <- NULL

daily_set_sar <- cbind(daily_set_sar, predictions_sar)

colnames(daily_set_sar) <- c("date", "daily_PUN", "predictions_sar")

ggplot(data = daily_set_sar, aes(x = date)) +
  geom_line(aes(y = daily_PUN, colour = "Actual")) +
  geom_line(aes(y = predictions_sar, colour = "sARIMA")) +
  scale_colour_manual("",
                      breaks = c("Actual", "sARIMA"),
                      values = c("black", "red")) +
  xlab("Time") +
  ylab("PUN (EUR/MWh)")+
  ggtitle("Predicted Daily Price of Electricity (sARIMA on sliding window)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5), legend.position = "bottom")
```



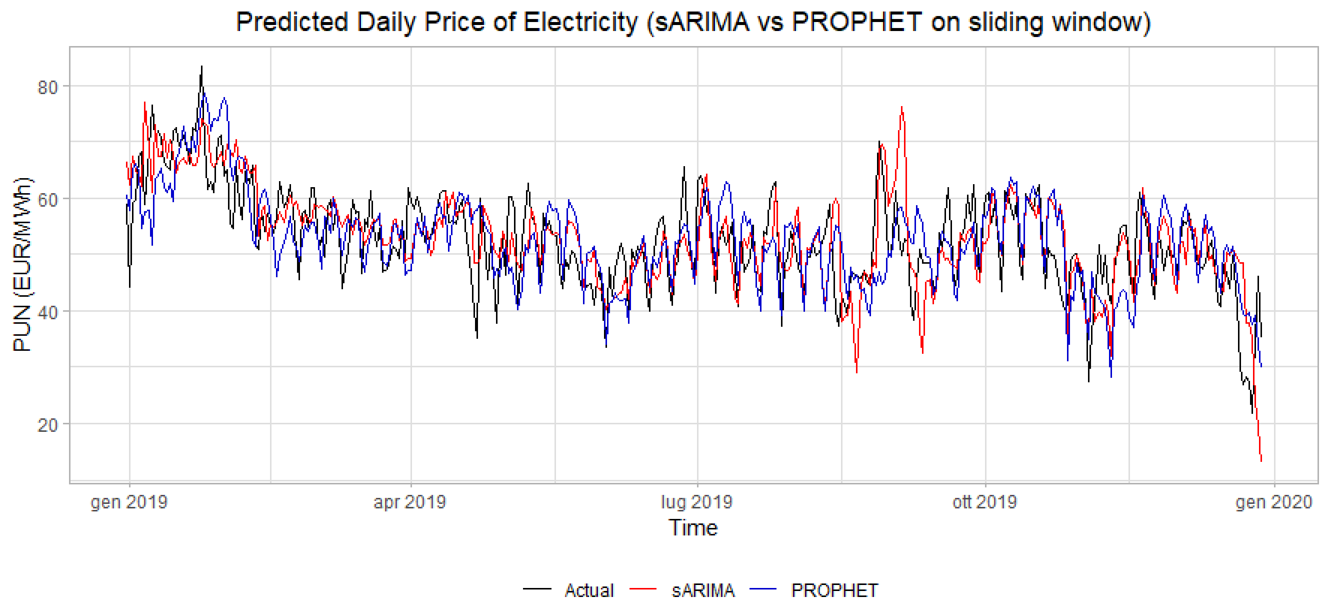
As with Prophet, we can store the resulting Performance Metrics..

```
sARIMA_roll_acc <- accuracy(object = daily_set_sar$predictions_sar, x = daily_set_sar$daily_PUN,
test = NULL)
```

.. and then plot the predictions obtained from the two models.

```
daily_set_sar <- daily_set_sar[1:(nrow(daily_set_sar)-3), ]

daily_df_fin <- cbind(daily_set_sar, pred_running$prophet_pred)
```

```r
colnames(daily_df_fin) <- c("date", "DailyPUN", "sARIMA_pred", "Prophet_pred")

ggplot(data = daily_df_fin, aes(x = date)) +
  geom_line(aes(y = DailyPUN, colour = "Actual")) +
  geom_line(aes(y = sARIMA_pred, colour = "sARIMA")) +
  geom_line(aes(y = Prophet_pred, colour = "PROPHET")) +
  scale_colour_manual("",
                      breaks = c("Actual", "sARIMA", "PROPHET"),
                      values = c("black", "red", "mediumblue")) +
  xlab("Time") +
  ylab("PUN (EUR/MWh)")+
  ggtitle("Predicted Daily Price of Electricity (sARIMA vs PROPHET on sliding window)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5), legend.position = "bottom")
```



Our job is done, as both sARIMA and Prophet are now able to forecast with a satisfactory degree of accuracy. sARIMA is slightly better according to MAPE, but Prophet has probably the edge here, because of the lesser effort it requires, in terms both of learning curve and looping automatization.

Thanks for reading!