A time series Analysis

Benoit Mialet

22/02/2022

In this work I used a dataset about power consumption in a building. The power consumption in Kw and the outdoor temperature in °C were monitored every 15 minutes for about 2 months. The objective of the work was to predict the power consumption of the building for the next 24 hours after the end of the dataset. Two predictions will be done:

- A first one only using past power consumption
- A second using outdoor temperature

I order to perform these predictions, I will build several models, compute then compare their prediction error, thanks to a train and test splitting of the dataset.

I) PREDICTING POWER CONSUMPTION USING PAST POWER CONSUMPTION

---

1) Data preparation

1.1) Data importation and plotting

I first import the dataset and look at the content, to identify and qualify the data

```
URL = "D:/Formations/DSTI/2022 03 - Time Series Analysis/assignment/git/Elec-train.csv"

power_df <- read.table(
  file = URL,
  header=TRUE,
  sep=";",dec=".",
  fileEncoding="Latin1",
  check.names=FALSE)

head(power_df)
```

```
##    ï»¿Timestamp Power (kW) Temp (CÂ°)
## 1 1/1/2010 1:15      165.1       10.6
## 2 1/1/2010 1:30      151.6       10.6
## 3 1/1/2010 1:45      146.9       10.6
## 4 1/1/2010 2:00      153.7       10.6
## 5 1/1/2010 2:15      153.8       10.6
## 6 1/1/2010 2:30      159.0       10.6
```

I first look at the data type and missing values.

```r
summary(power_df)
```

```
##  ï»¿Timestamp         Power (kW)      Temp (CÂ°)
##  Length:4603        Min.   :134.1   Min.   : 3.90
##  Class :character   1st Qu.:163.3   1st Qu.: 8.90
##  Mode  :character   Median :253.7   Median :11.10
##                     Mean   :231.6   Mean   :10.89
##                     3rd Qu.:277.5   3rd Qu.:12.80
##                     Max.   :355.1   Max.   :19.40
##                     NA's   :96
```

96 power values are missing (the values that need to be predicted).

I can see that the timestamp frequency is 15 minutes. My objective is to display the time by days. To create my time series object, I will have to set a frequency of 4*24 = 96 (4 observations per hour, 24 hours per day). The first timestamp is 1:15, which is the 6th timestamp starting from 0:00. I first create my time series object based on the power data. I checked on the .csv file that NA values started from row 4507 onward, so I subsample the data from row 1 to 4507.

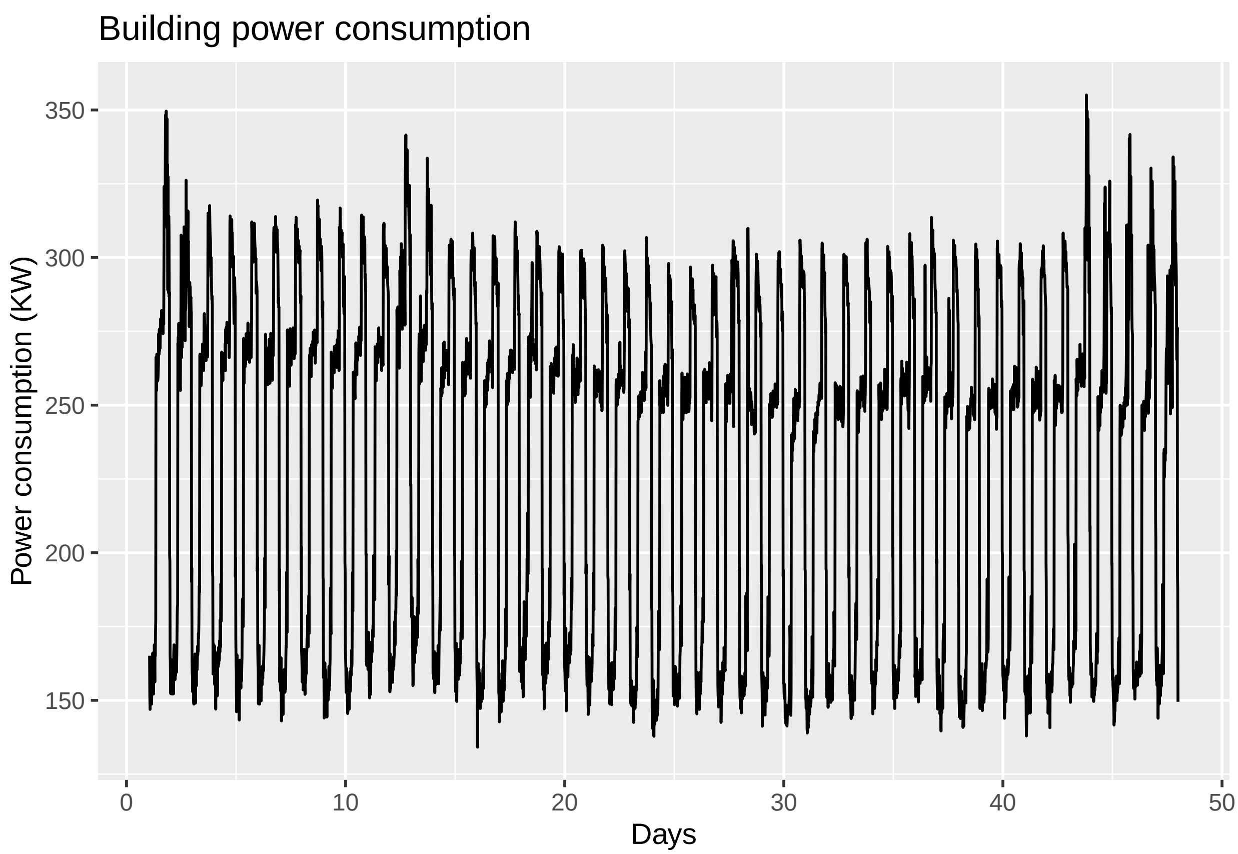
```r
power_ts <- ts(power_df[1:4507,2], frequency = 96, start=c(1,6))
head(power_ts, 10)
```

```
## Time Series:
## Start = c(1, 6)
## End = c(1, 15)
## Frequency = 96
##  [1] 165.1 151.6 146.9 153.7 153.8 159.0 157.7 163.2 151.7 148.7
```

```r
tail(power_ts, 10)
```

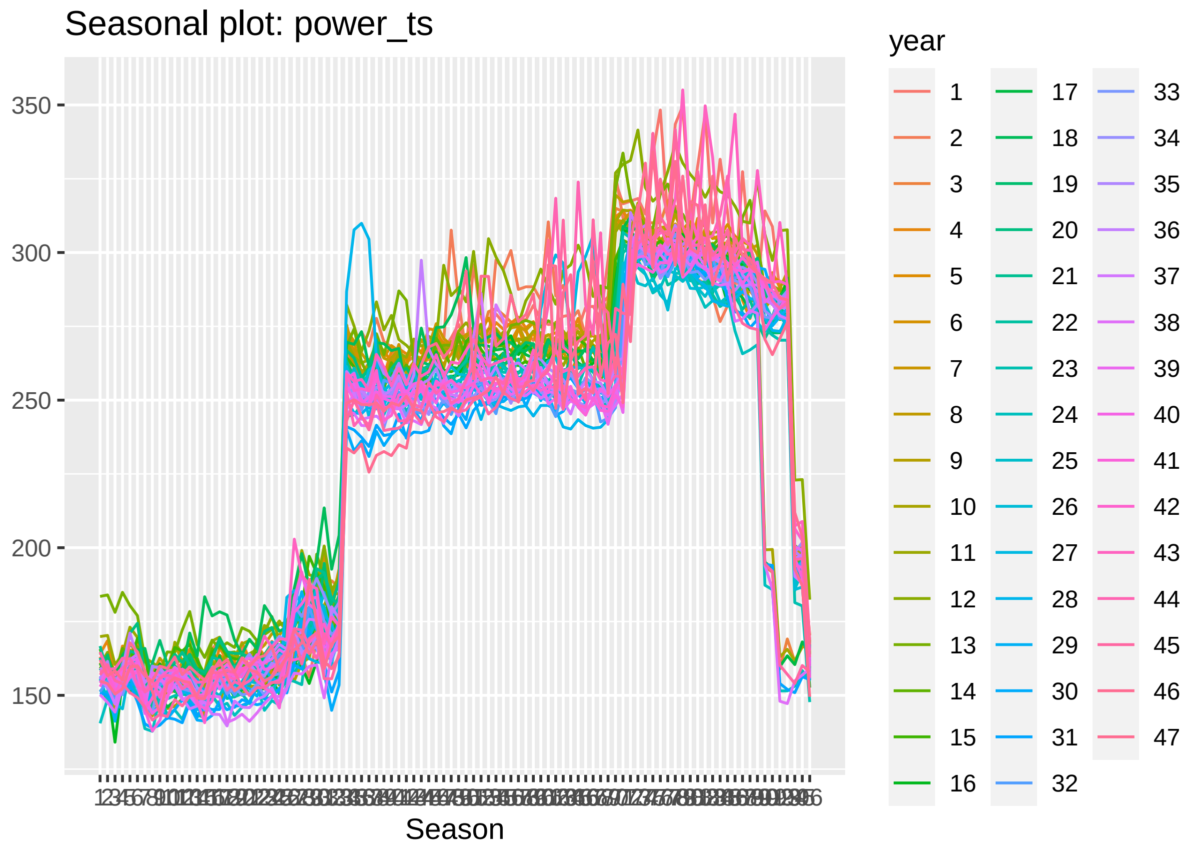
```
## Time Series:
## Start = c(47, 87)
## End = c(47, 96)
## Frequency = 96
##  [1] 299.2 297.9 292.7 270.6 265.4 270.9 276.2 192.7 187.1 149.5
```

```r
autoplot(power_ts) +
ggtitle('Building power consumption')+
xlab('Days')+
ylab('Power consumption (KW)')
```



I check my period on a seasonal plot

```
ggseasonplot(power_ts)
```

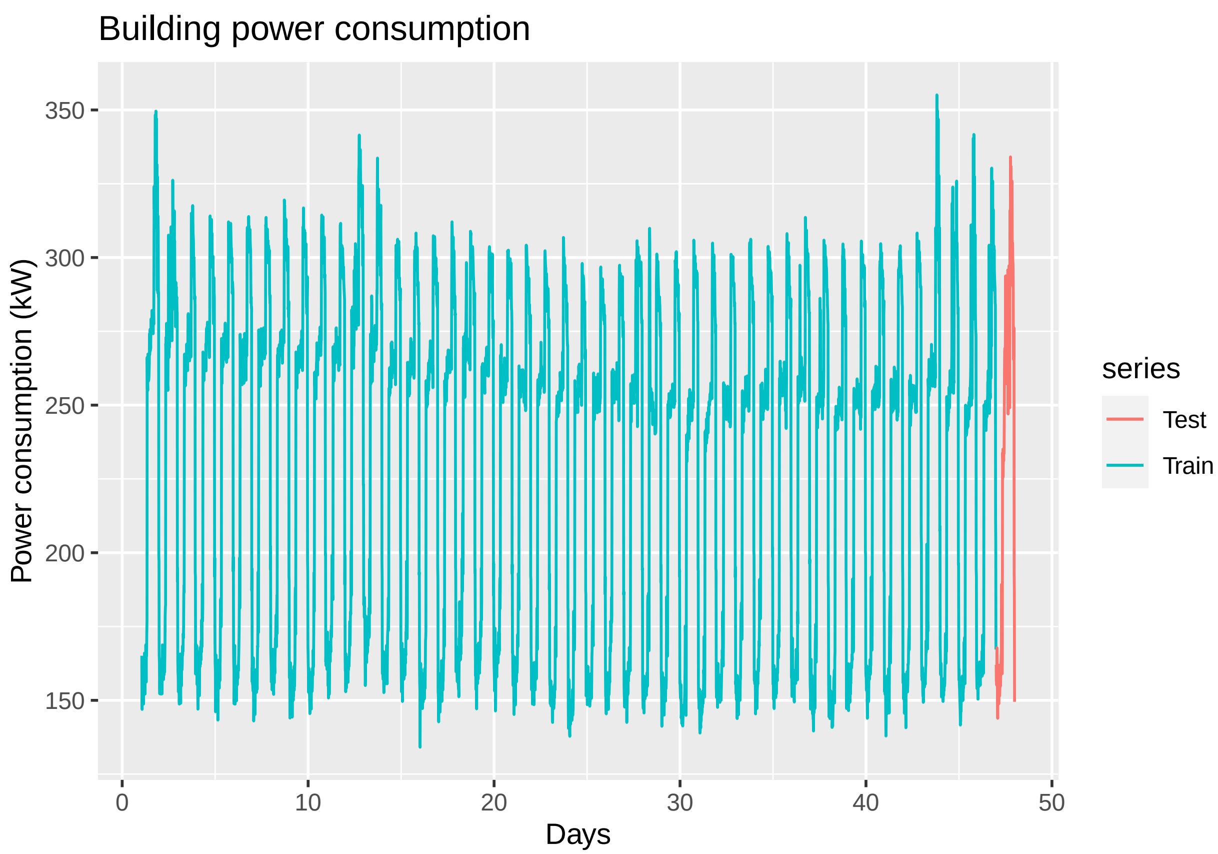


My period time seems correct. However, the end of the period seems different for some days. After checking, these days correspond to weekend days. In the further, I will only keep a period of 96 because unfortunately ARIMA models didn't work with a period of 96*7.

1.2) Splitting Train and Test data for model training

The goal is to forecast **one day** of power consumption, which corresponds to **96 (24 x 4) values**. I will thus adapt my test dataset to this length, and will take the last 96 values. The remaining previous values will be my training dataset.

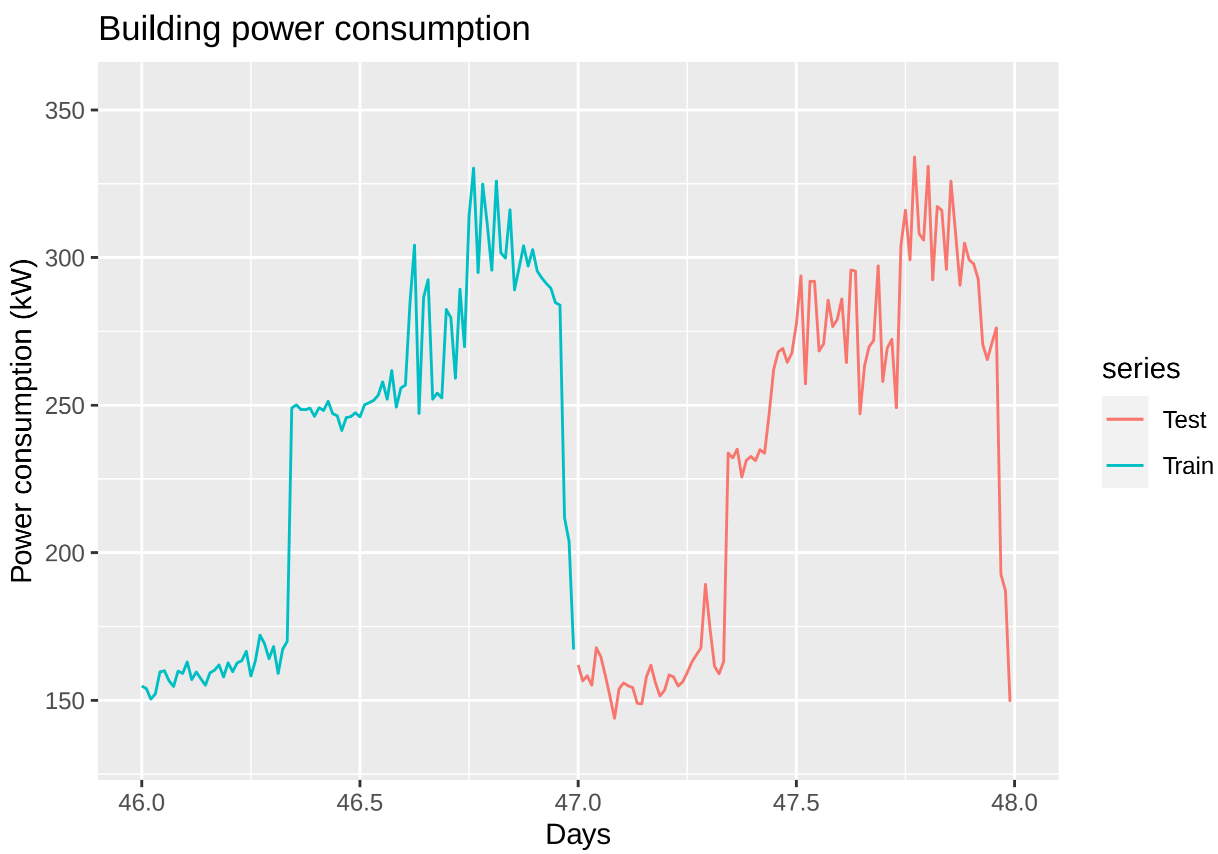
```
power_ts_train = ts(power_df[1:(46*96-5),2], frequency = 96, start=c(1,6), end=c(46,96))
power_ts_test = ts(power_df[(46*96-4):4507,2], frequency = 96, start=c(47,1), end=c(47,96) )

autoplot(power_ts_train,series='Train') +
  autolayer(power_ts_test,series='Test')+
  ggtitle ('Building power consumption') +
  xlab('Days') +
  ylab('Power consumption (kW)')
```



Focus on training set and testing set junction

```
autoplot(power_ts_train,series='Train') +
  autolayer(power_ts_test,series='Test')+
  ggtitle ('Building power consumption') +
  xlim(c(46,48))+
  xlab('Days') +
  ylab('Power consumption (kW)')
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```



I now have the datasets to buid some models for prediction.

2) Holt-Winters Models

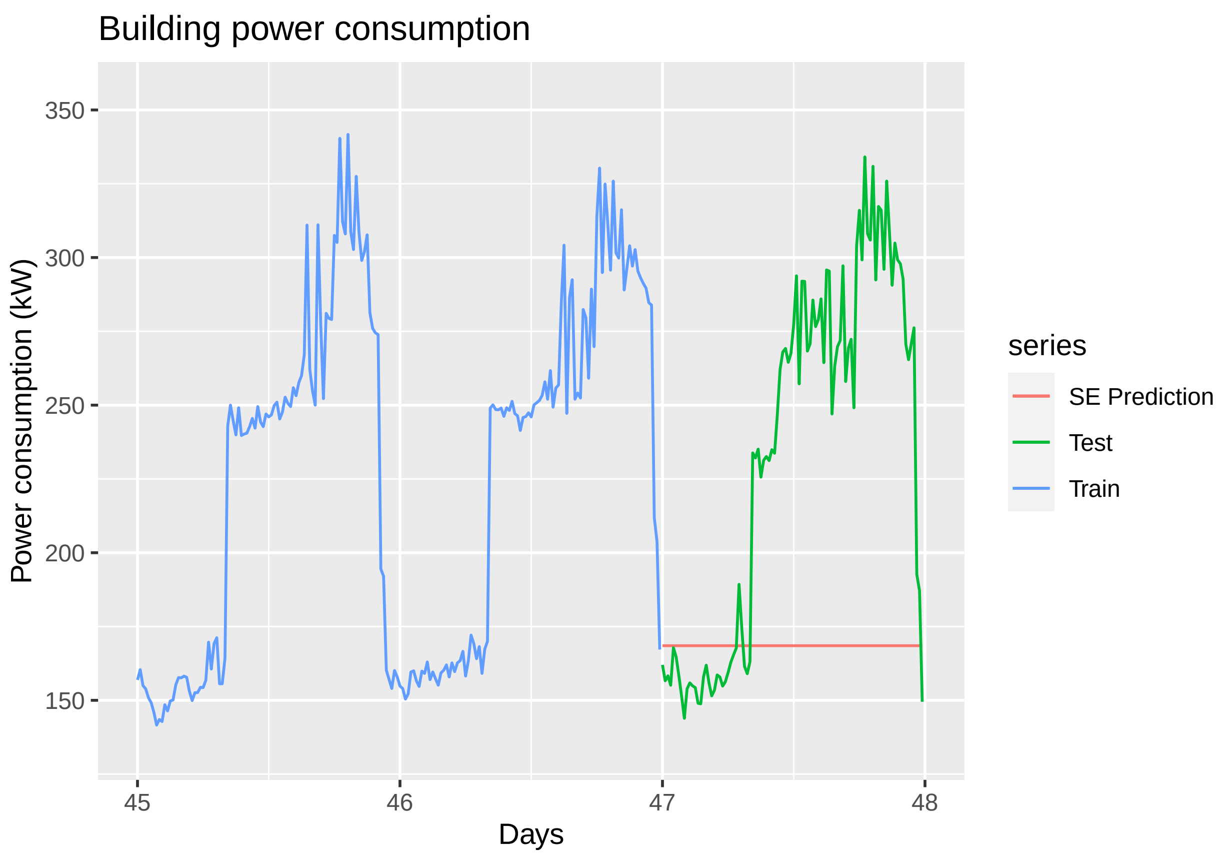
2.1) Simple exponential smoothing

Computing the predictions and mean square error

```
model_HW_SE=HoltWinters(power_ts_train,alpha=NULL,beta=FALSE,gamma=FALSE)
predict_HW_SE<-predict(model_HW_SE,n.ahead=96)

autoplot(power_ts_train,series='Train') +
  autolayer(predict_HW_SE,series='SE Prediction', PI=FALSE)+
  autolayer(power_ts_test,series='Test')+
  ggtitle ('Building power consumption') +
  xlim(c(45,48)) +
  xlab('Days') +
  ylab('Power consumption (kW)')
```

```
## Warning: Ignoring unknown parameters: PI
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```




```
#Root mean square error
model_HW_SE_RMSE = sqrt(mean((predict_HW_SE-power_ts_test)^2))
model_HW_SE_RMSE
```

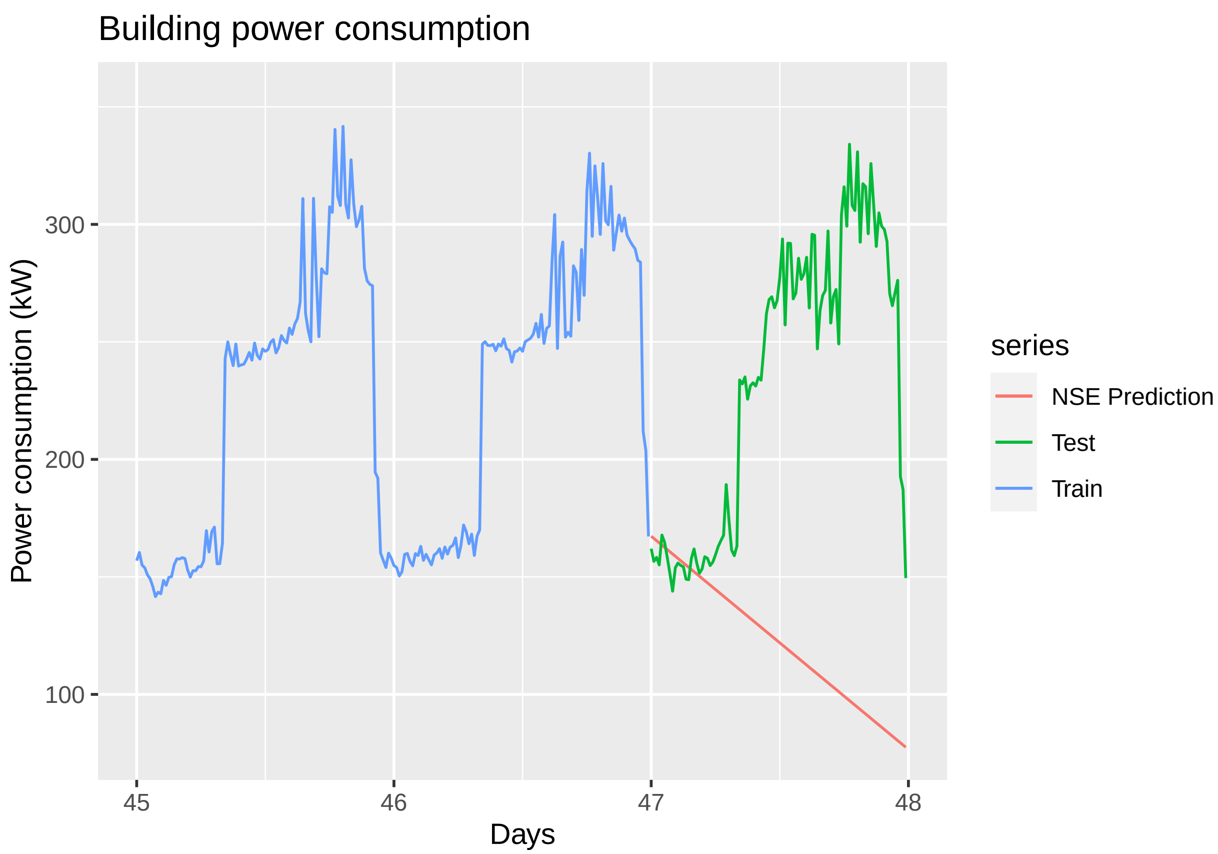
```
## [1] 88.88478
```

2.2) Non seasonal Holt-Winters smoothing

```
model_HW_NS=HoltWinters(power_ts_train,alpha=NULL,beta=NULL,gamma=FALSE)
predict_HW_NS<-predict(model_HW_NS,n.ahead=96)

autoplot(power_ts_train,series='Train') +
  autolayer(predict_HW_NS,series='NSE Prediction', PI=FALSE)+
  autolayer(power_ts_test,series='Test')+
  ggtitle ('Building power consumption') +
  xlim(c(45,48)) +
  xlab('Days') +
  ylab('Power consumption (kW)')
```

```
## Warning: Ignoring unknown parameters: PI
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```



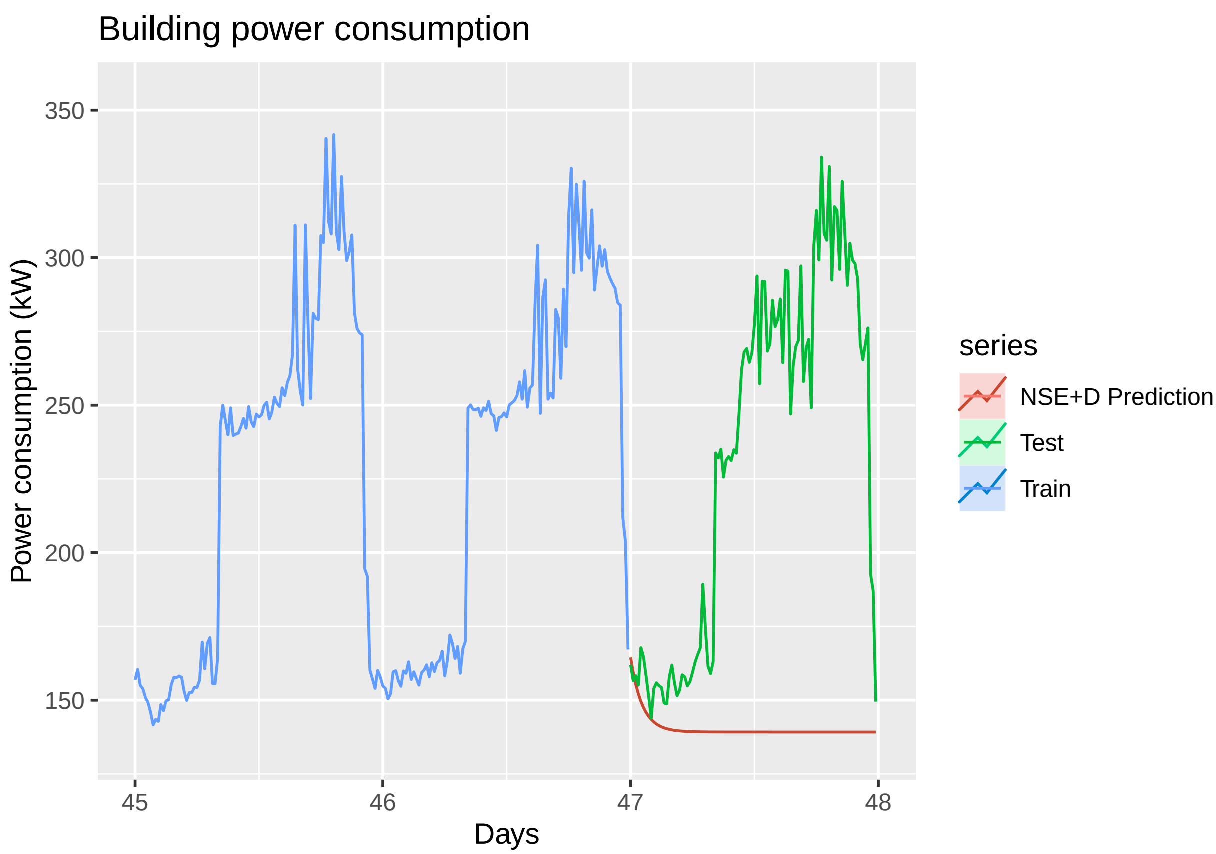

```
#Root mean square error
model_HW_NS_RMSE = sqrt(mean((predict_HW_NS-power_ts_test)^2))
model_HW_NS_RMSE
```

```
## [1] 138.6678
```

2.3) Non seasonal Holt-Winters smoothing + damping effect

```
predict_HW_NS_D=holt(power_ts_train,h=96, damped = TRUE, phi = 0.8)

autoplot(power_ts_train,series='Train') +
  autolayer(predict_HW_NS_D,series='NSE+D Prediction', PI=FALSE)+
  autolayer(power_ts_test,series='Test')+
  ggtitle ('Building power consumption') +
  xlim(c(45,48)) +
  xlab('Days') +
  ylab('Power consumption (kW)')
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```




```
#Root mean square error
model_HW_NS_D_RMSE = sqrt(mean((predict_HW_NS_D$mean-power_ts_test)^2))
model_HW_NS_D_RMSE
```

```
## [1] 111.9073
```

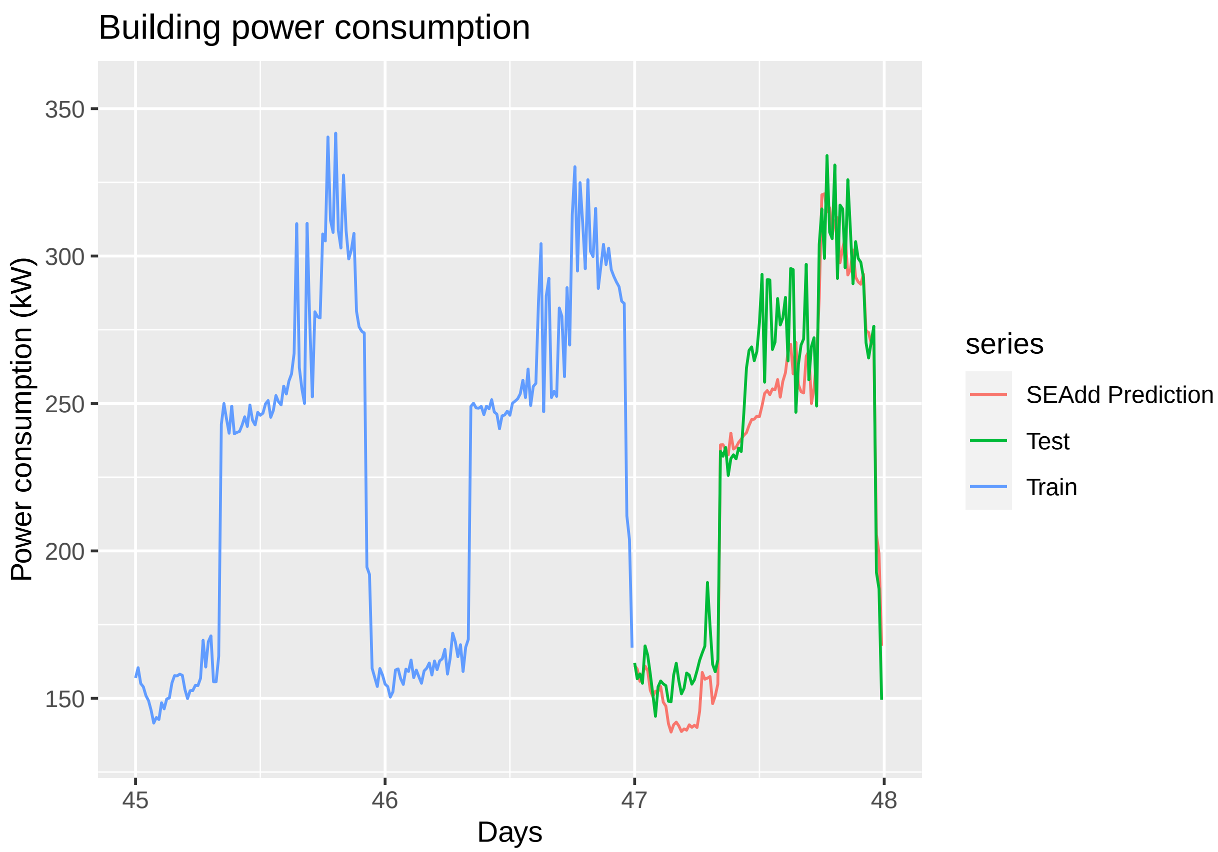
The model is more flexible with the damping effect. However, simple H-W exponential smoothing and non seasonal smoothing seem not adapted for these time series, because they ignore the effect of the period.

2.3) Additive seasonal Holt-Winters smoothing

```
model_HW_addSE=HoltWinters(power_ts_train,alpha=NULL,beta=NULL,gamma=NULL, seasonal = 'additive')
predict_HW_addSE<-predict(model_HW_addSE,n.ahead=96)

autoplot(power_ts_train,series='Train') +
  autolayer(predict_HW_addSE,series='SEAdd Prediction', PI=TRUE)+
  autolayer(power_ts_test,series='Test')+
  ggtitle ('Building power consumption') +
  xlim(c(45,48)) +
  xlab('Days') +
  ylab('Power consumption (kW)')
```

```
## Warning: Ignoring unknown parameters: PI
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```




```
#Root mean square error
model_HW_addSE_RMSE = sqrt(mean((predict_HW_addSE-power_ts_test)^2))
model_HW_addSE_RMSE
```

```
## [1] 16.86543
```

This models fits much better to the data.

2.4) Additive seasonal Holt-Winters smoothing with Box-Cox transformation

Unfortunately, the ets() function called by hw() function has a limit of frequency of 24. With my time series (frequency=96), it returned an error that I couldn't fix: frequency is too high for fitting:

Error : Error in ets(x, "AAA", alpha = alpha, beta = beta, gamma = gamma, phi = phi, : Frequency too high

```
# predict = hw(power_ts_train, seasonal = 'additive', lambda = 'auto', h =96)
#
# autoplot(power_ts_train,series='Train') +
#   autolayer(predict,series='SEMult Prediction', PI=TRUE)+
#   autolayer(power_ts_test,series='Test')+
```

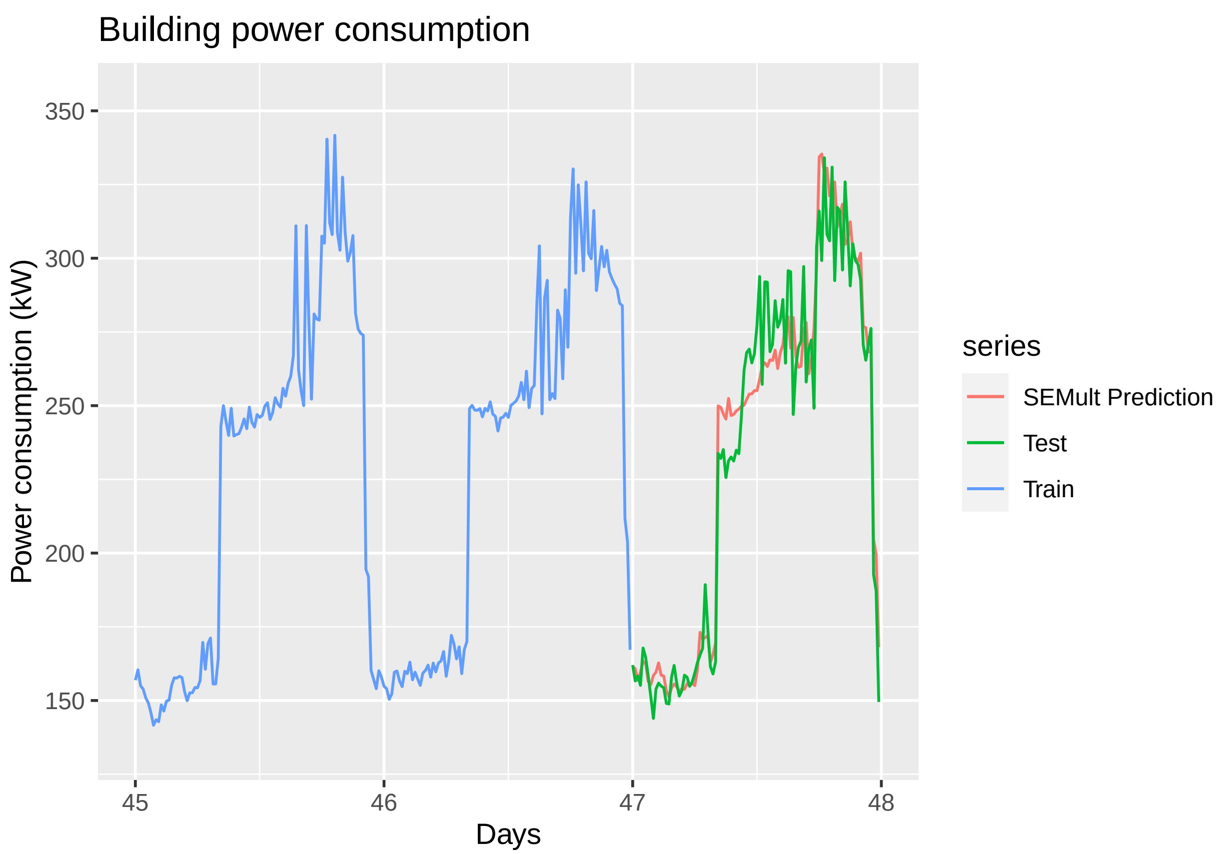
```
#   ggtitle ('Building power consumption') +
#   xlim(c(35,48)) +
#   xlab('Days') +
#   ylab('Power consumption (kW)')
```

2.5) Multiplicative seasonal Holt-Winters smoothing

```
model_HW_MultSE=HoltWinters(power_ts_train,alpha=NULL,beta=NULL,gamma=NULL, seasonal = 'multiplicative')
predict_HW_MultSE<-predict(model_HW_MultSE,n.ahead=96)

autoplot(power_ts_train,series='Train') +
  autolayer(predict_HW_MultSE,series='SEMult Prediction', PI=TRUE)+
  autolayer(power_ts_test,series='Test')+
  ggtitle ('Building power consumption') +
  xlim(c(45,48)) +
  xlab('Days') +
  ylab('Power consumption (kW)')
```

```
## Warning: Ignoring unknown parameters: PI
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```



```
#Root mean square error
model_HW_MultSE_RMSE = sqrt(mean((predict_HW_MultSE-power_ts_test)^2))
model_HW_MultSE_RMSE
```

```
## [1] 13.92376
```

Multiplicative effect has a lower error, probably because the amplitude of values during the period tend to follow a trend, which is captured by the model.

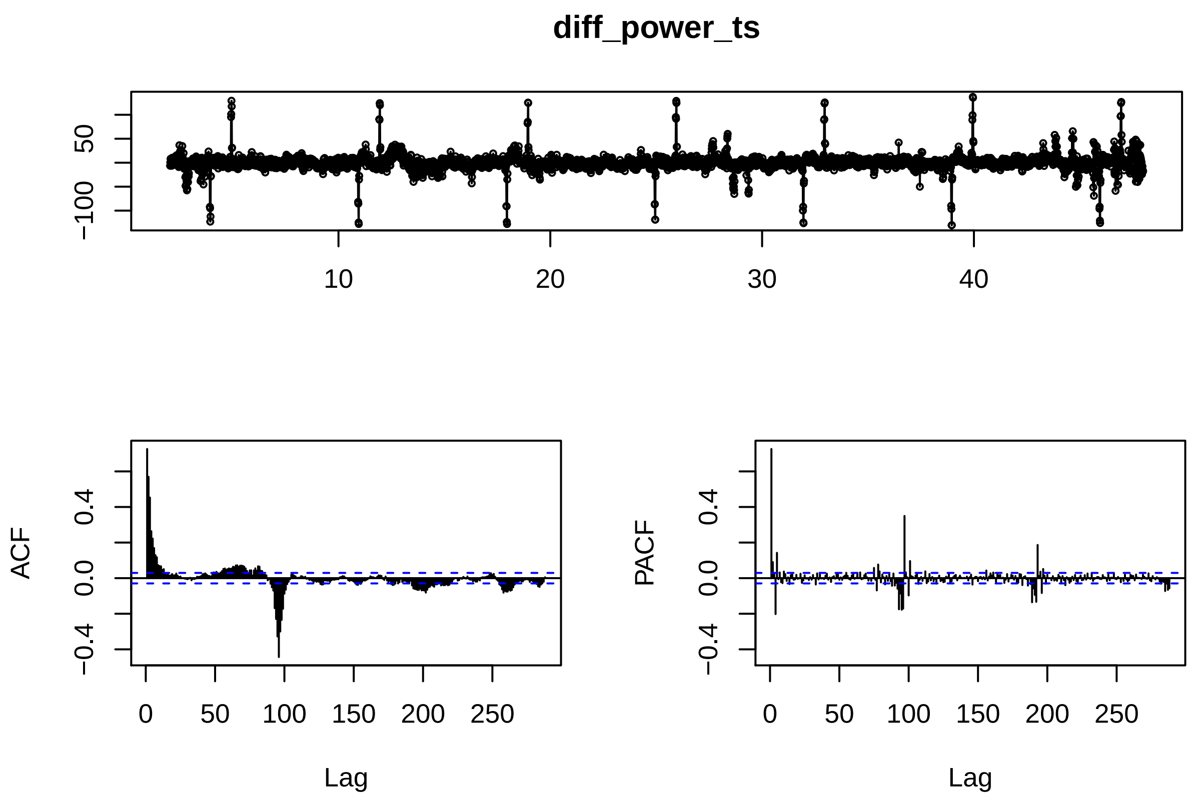
#3) ARIMA models

3.1) Removing trend and seasonal patterns by differenciation

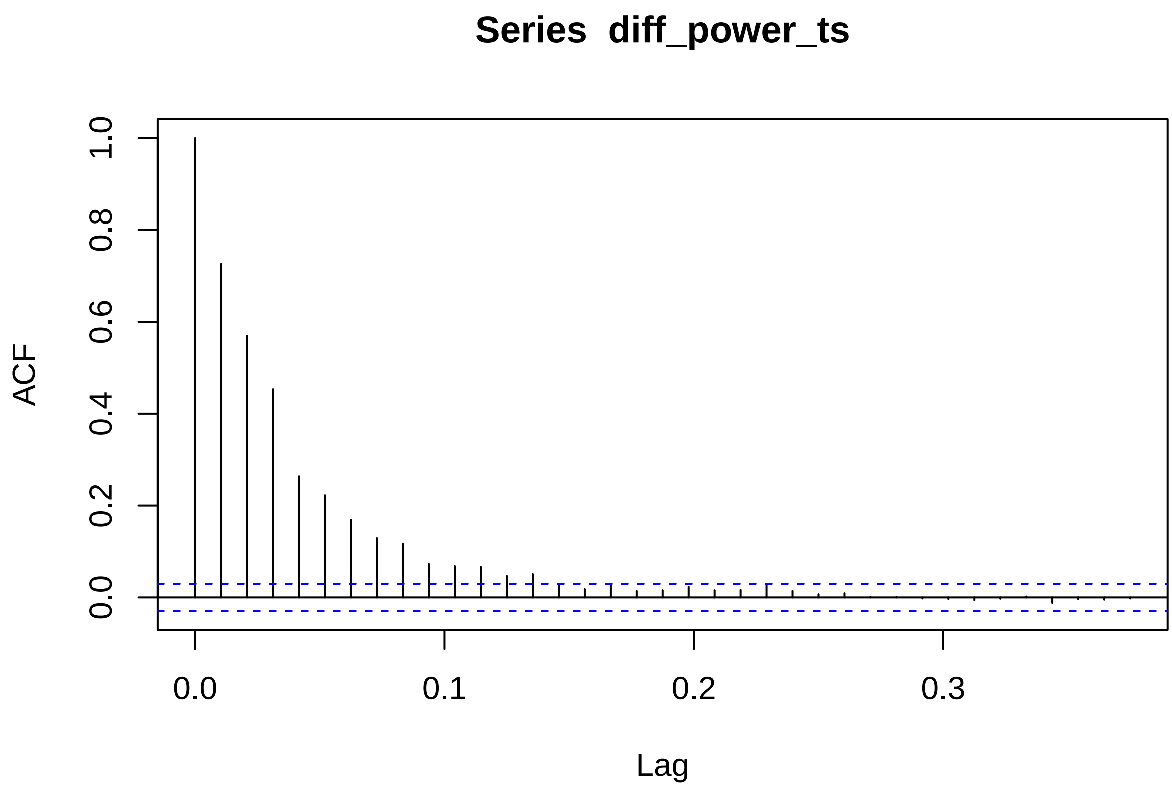
3.1.a) Based on a day period

Removing the seasonal pattern, based on a day period (24 hours * 4 quarters of hour)

```
diff_power_ts = diff(power_ts, lag = (96), differences = 1)
tsdisplay(diff_power_ts)
```

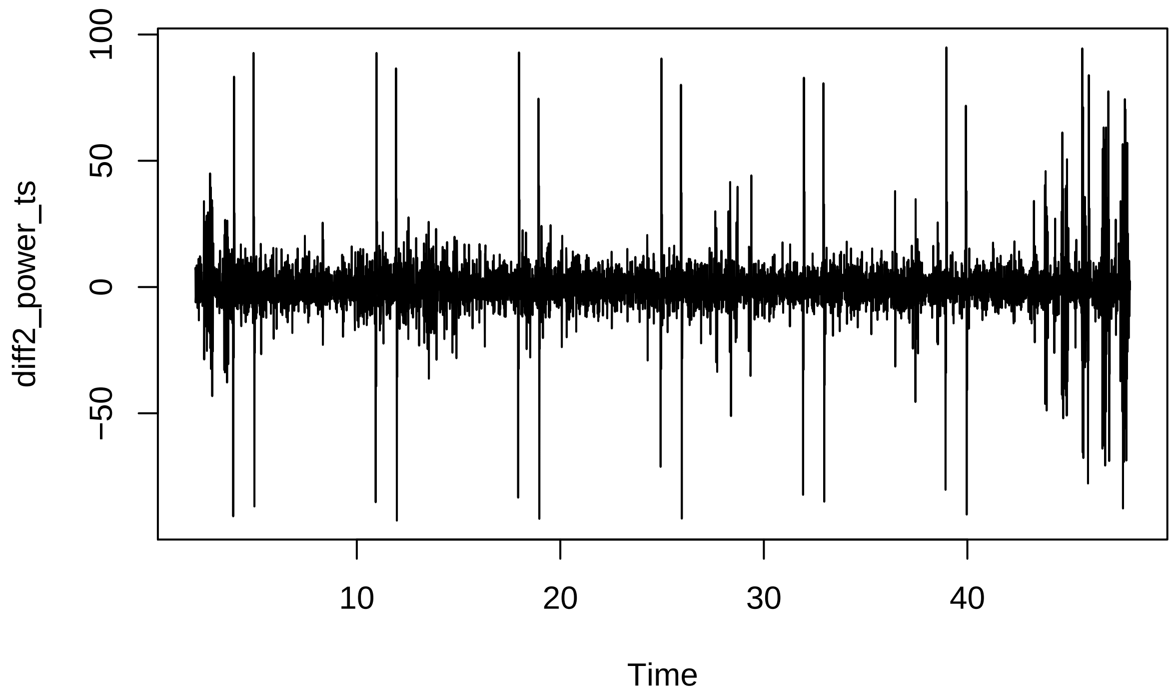



```
acf(diff_power_ts)
```

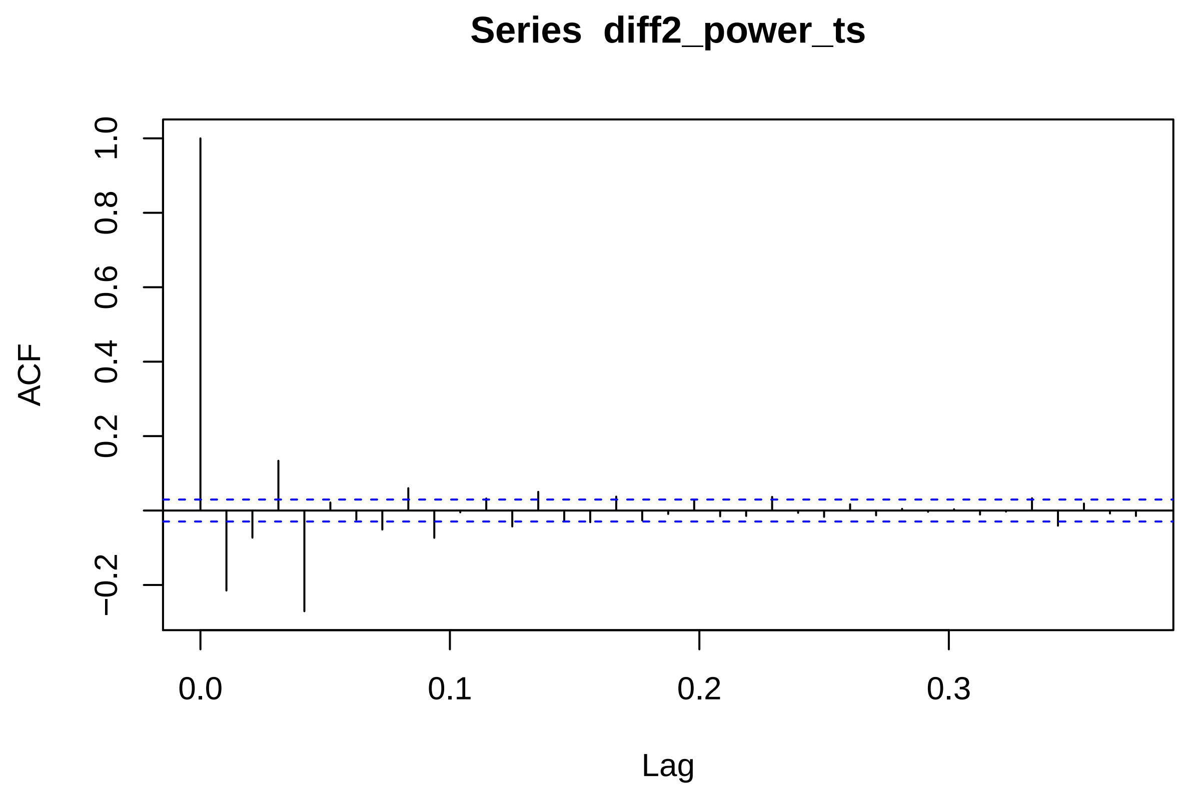


The time series still show a trend (ACF positive). I thus differentiate a second time.

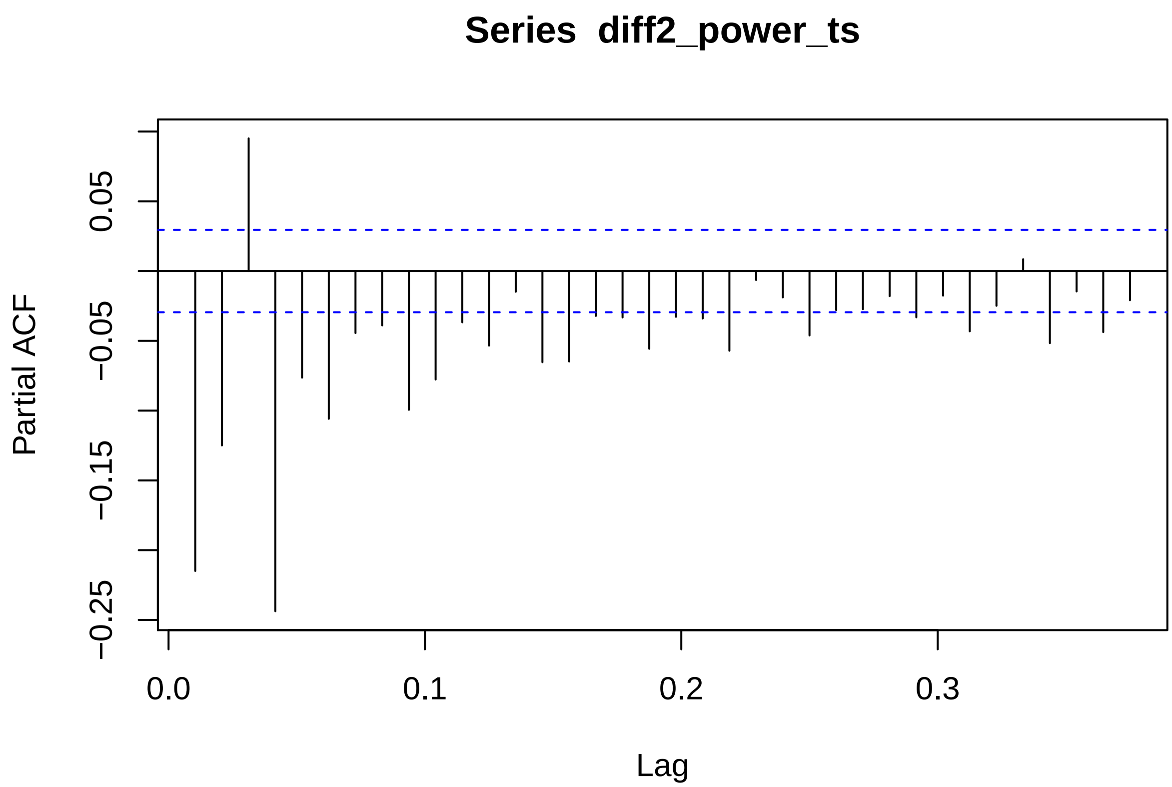
```
diff2_power_ts = diff(diff_power_ts)
plot(diff2_power_ts)
```




```
acf(diff2_power_ts)
```




```
pacf(diff2_power_ts)
```



The model now looks stationary. The exponential decreasing shape on PACF makes me prefer an ARIMA model with moving average, rather than autorecursive one. I will however test both of them.

I thus check the $\mathcal{H}_0$ hypothesis : residual time series is white noise

```
Box.test(diff2_power_ts, lag = 10, type = 'Ljung-Box')
```

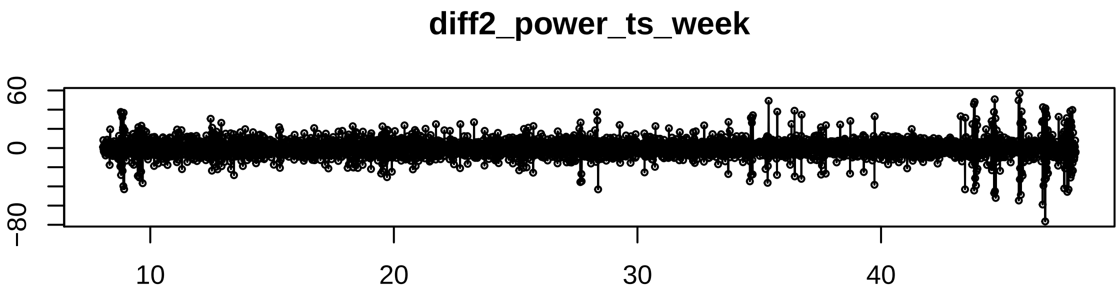
```
##
##  Box-Ljung test
##
## data:  diff2_power_ts
## X-squared = 685.64, df = 10, p-value < 2.2e-16
```

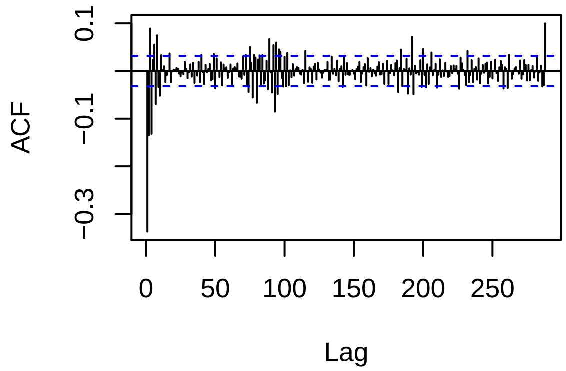
I reject this hypothesis. Thus I should be able to model the noise with an ARIMA model.

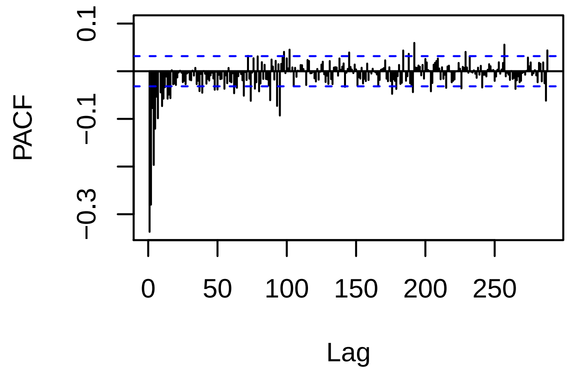
3.1.b) Based on a week period

If I Remove the seasonal pattern with a week period (7 days * 24 hours * 4 quarters of hour), then a default differentiation, I obtain a more satisfying model. Unfortunately, I couldn't go further in that direction, because as I mentioned above, Arima() function doesn't accept such long periods.

```
diff2_power_ts_week = diff(diff(power_ts, lag = (96*7), differences = 1))
tsdisplay(diff2_power_ts_week)
```





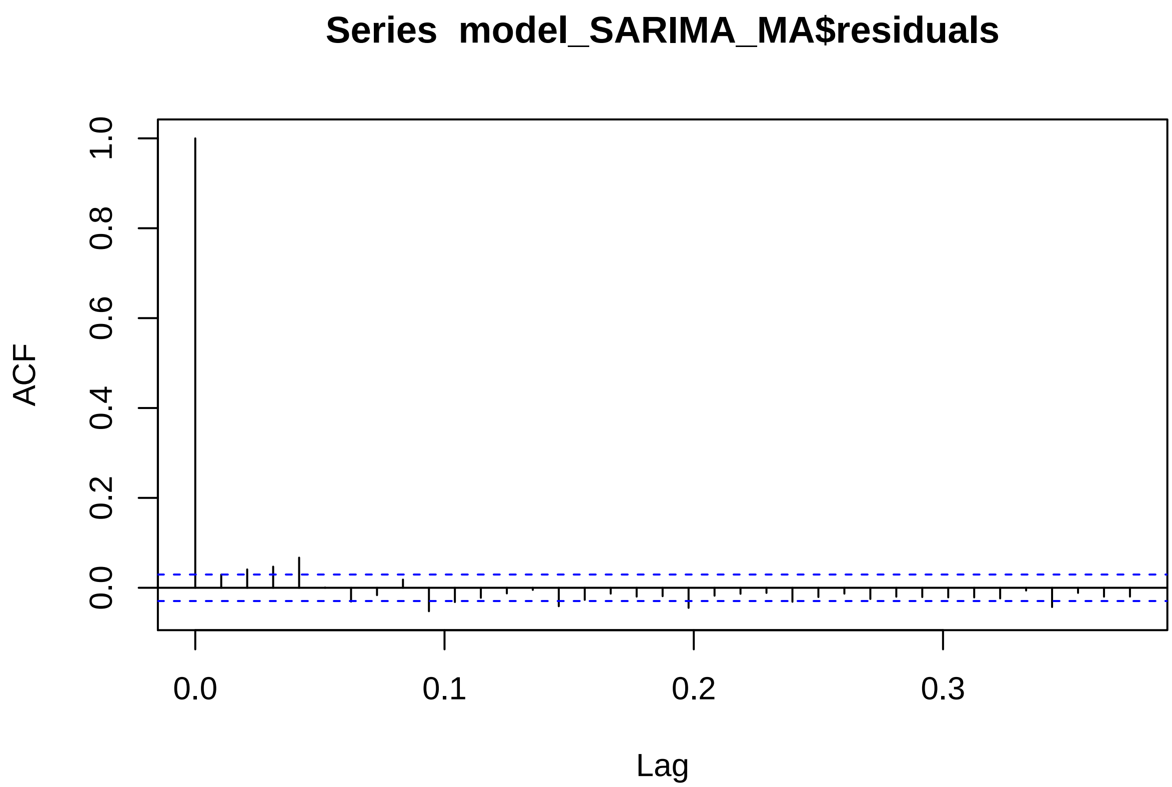


3.2) SARIMA - moving average

Based on 3.1.b), I tried an ARIMA model (moving average version) with several order parameters. The best one I found regarding the RMSE was the following one.

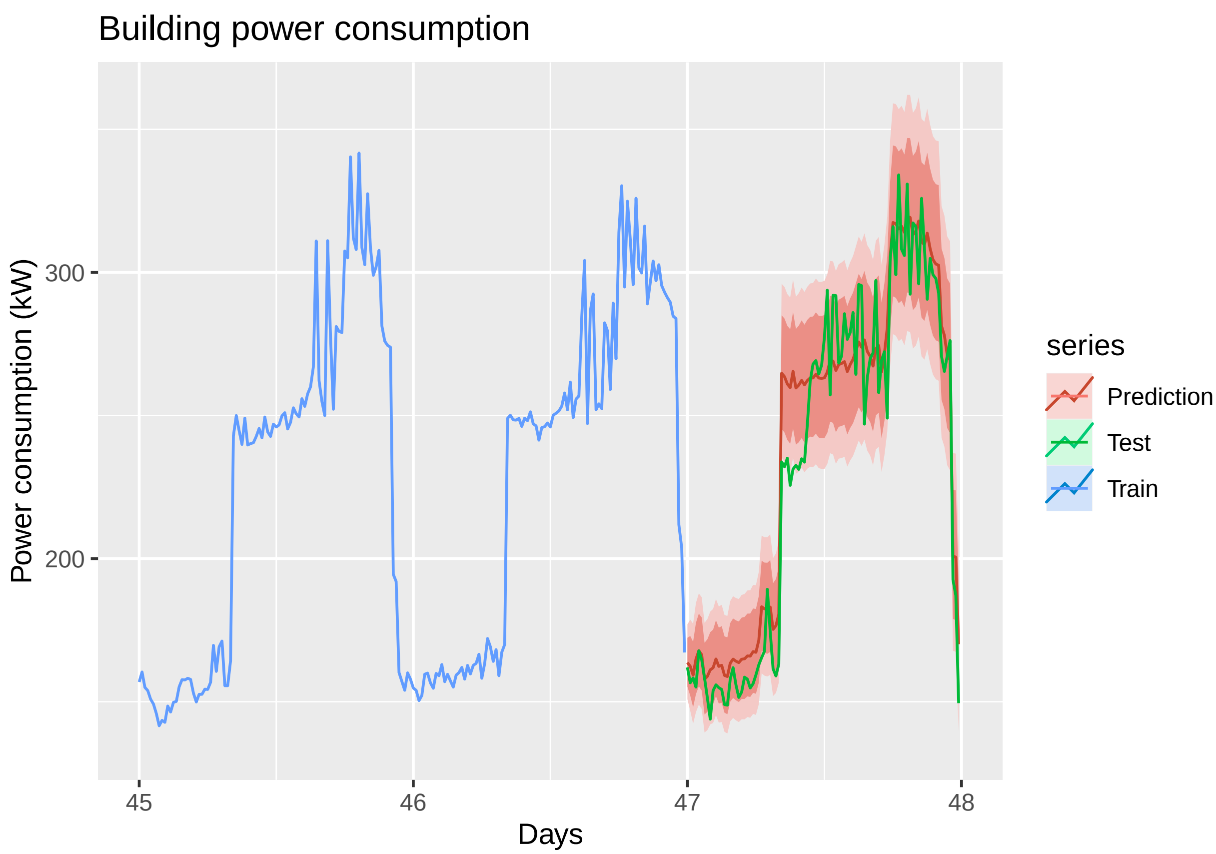
```
model_SARIMA_MA = Arima(power_ts_train, order = c(0,1,4), seasonal = c(0,1,1), lambda = 'auto')
```

```
# checkresiduals(model_SARIMA_MA)
acf(model_SARIMA_MA$residuals)
```




```
predict_SARIMA_MA<-forecast(model_SARIMA_MA, h=96)

autoplot(power_ts_train,series='Train') +
  autolayer(predict_SARIMA_MA,series='Prediction', PI=TRUE)+
  autolayer(power_ts_test,series='Test')+
  ggtitle ('Building power consumption') +
  xlim(c(45,48)) +
  xlab('Days') +
  ylab('Power consumption (kW)')
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```




```r
#Root mean square error
model_SARIMA_MA_RMSE = sqrt(mean((predict_SARIMA_MA$mean-power_ts_test)^2))
model_SARIMA_MA_RMSE
```

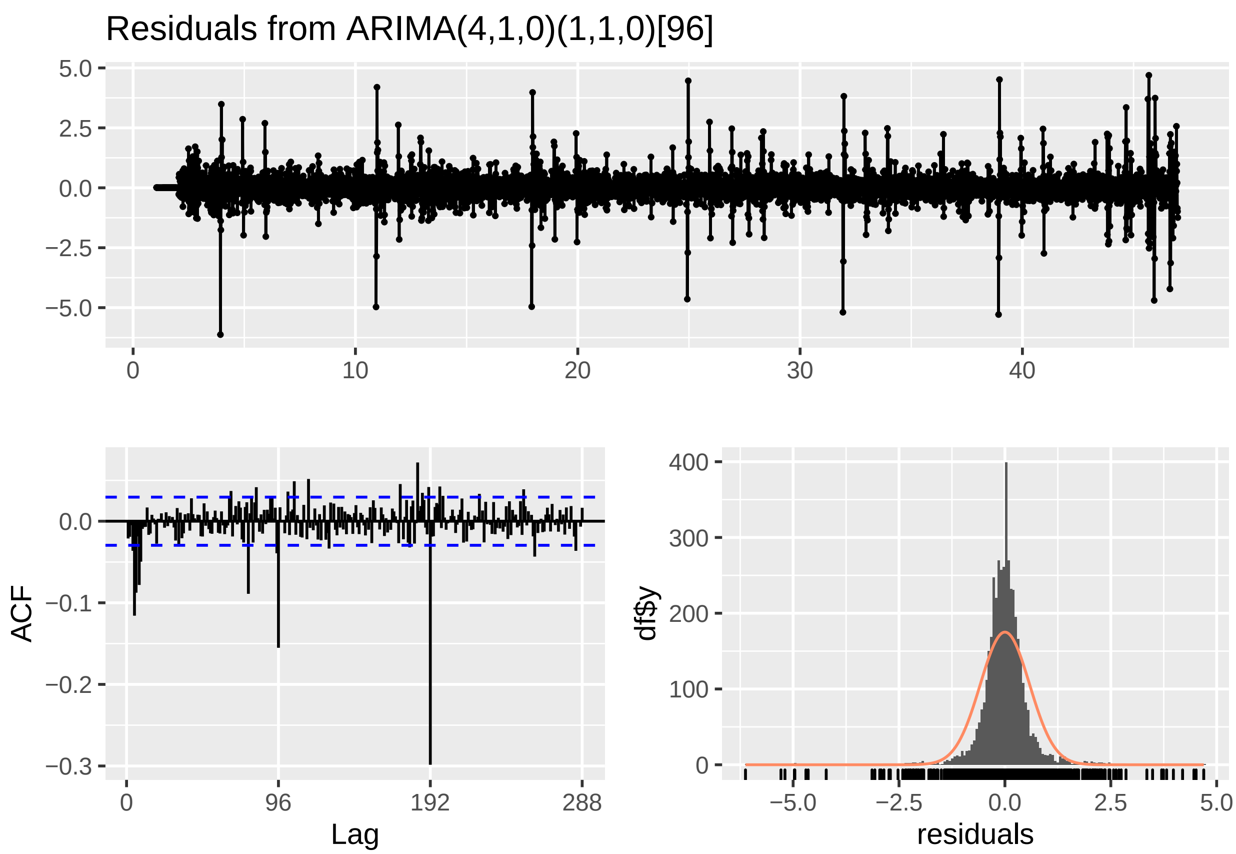
```
## [1] 15.37172
```

3.3) SARIMA - autoregressive

Based on 3.1), I built an ARIMA model (autoregressive version) with following parameters:

```r
model_SARIMA_AR = Arima(power_ts_train, order = c(4,1,0), seasonal = c(1,1,0), lambda = 'auto')
```

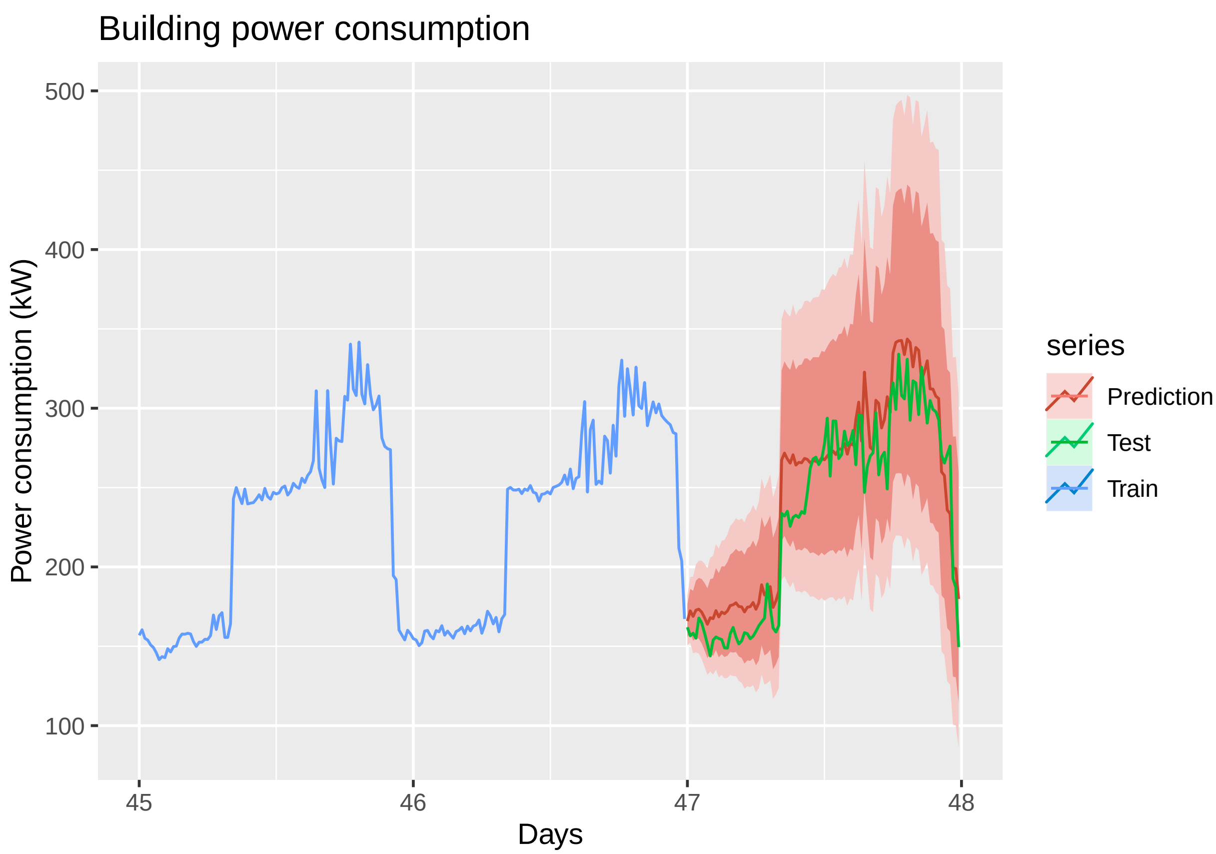
```r
checkresiduals(model_SARIMA_AR)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(4,1,0)(1,1,0)[96]
## Q* = 978.35, df = 187, p-value < 2.2e-16
##
## Model df: 5.   Total lags used: 192
```

```
predict_SARIMA_AR<-forecast(model_SARIMA_AR, h=96)

autoplot(power_ts_train,series='Train') +
  autolayer(predict_SARIMA_AR,series='Prediction', PI=TRUE)+
  autolayer(power_ts_test,series='Test')+
  ggtitle ('Building power consumption') +
  xlim(c(45,48)) +
  xlab('Days') +
  ylab('Power consumption (kW)')
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```




```
#Root mean square error
model_SARIMA_AR_RMSE = sqrt(mean((predict_SARIMA_AR$mean-power_ts_test)^2))
model_SARIMA_AR_RMSE
```

```
## [1] 23.49247
```

As suggested, the moving average SARIMA model fits better than the autorecursive one.

4) Neural Network Models

4.1) Autoregressive NN

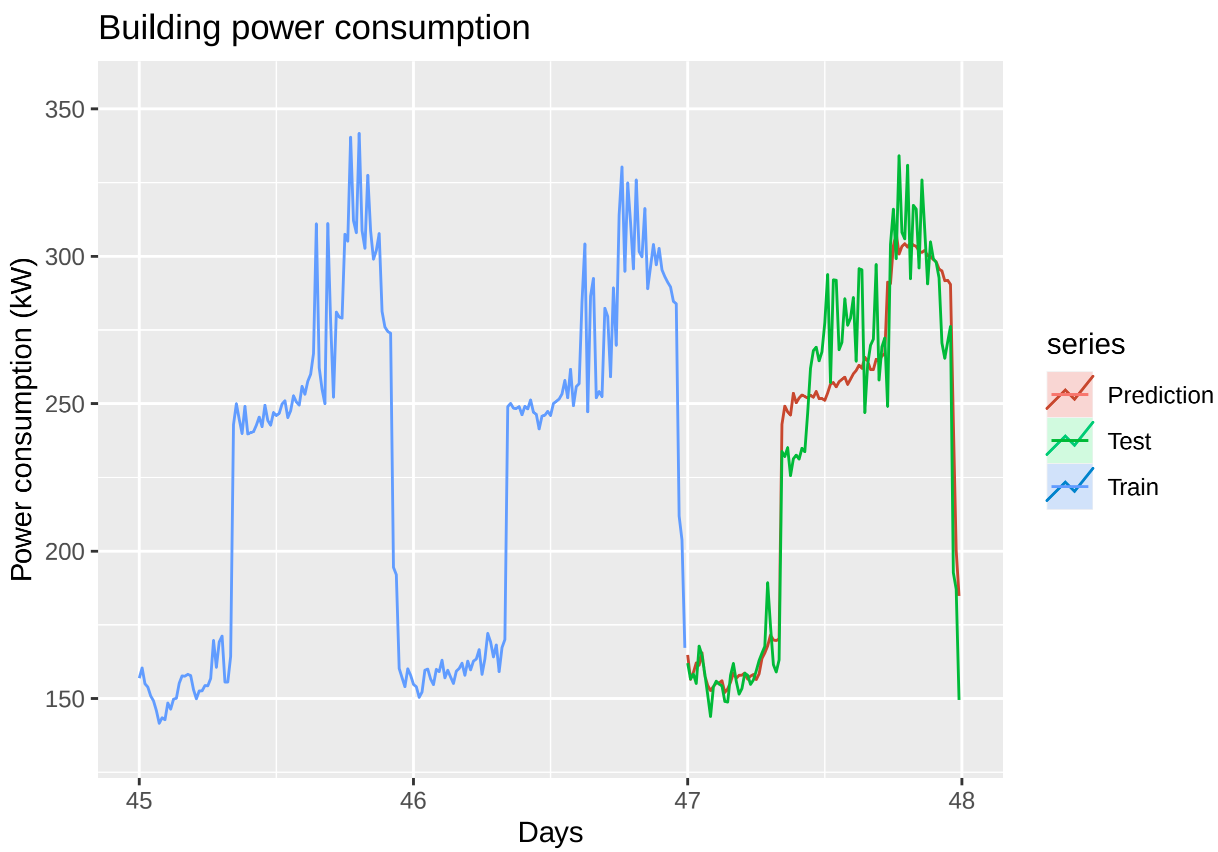
I also try to predict consumption with a neural network. I tried several architectures. I only keep here the best one I found

```
model_NNAR = nnetar(power_ts_train, p=40, P=1, size = 20)
```

```
predict_NNAR<-forecast(model_NNAR, h=96)

autoplot(power_ts_train,series='Train') +
  autolayer(predict_NNAR,series='Prediction', PI=TRUE)+
  autolayer(power_ts_test,series='Test')+
  ggtitle ('Building power consumption') +
  xlim(c(45,48)) +
```

```
  xlab('Days') +
  ylab('Power consumption (kW)')
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```




```
#Root mean square error
model_NNAR_RMSE = sqrt(mean((predict_NNAR$mean-power_ts_test)^2))
model_NNAR_RMSE
```

```
## [1] 16.90124
```

5) RMSE comparison and selection of the best model

```
cat('RMSE Holt-Winters simple exponential:',model_HW_SE_RMSE,'\n')
```

```
## RMSE Holt-Winters simple exponential: 88.88478
```

```
cat('RMSE Non seasonal Holt-Winters:',model_HW_NS_RMSE,'\n')
```

```
## RMSE Non seasonal Holt-Winters: 138.6678
```

```
cat('RMSE Non seasonal Holt-Winters + dampling:',model_HW_NS_D_RMSE,'\n')
```

```
## RMSE Non seasonal Holt-Winters + dampling: 111.9073
```

```
cat('RMSE additive seasonal Holt-Winters:',model_HW_addSE_RMSE,'\n')
```

```
## RMSE additive seasonal Holt-Winters: 16.86543
```

```
cat('RMSE multiplicative seasonal Holt-Winters:',model_HW_MultSE_RMSE,'\n')
```

```
## RMSE multiplicative seasonal Holt-Winters: 13.92376
```

```
cat('RMSE SARIMA - moving average:',model_SARIMA_MA_RMSE,'\n')
```

```
## RMSE SARIMA - moving average: 15.37172
```

```
cat('RMSE SARIMA - autoregressive:',model_SARIMA_AR_RMSE,'\n')
```

```
## RMSE SARIMA - autoregressive: 23.49247
```

```
cat('RMSE Autoregressive neural network:',model_NNAR_RMSE,'\n')
```

```
## RMSE Autoregressive neural network: 16.90124
```

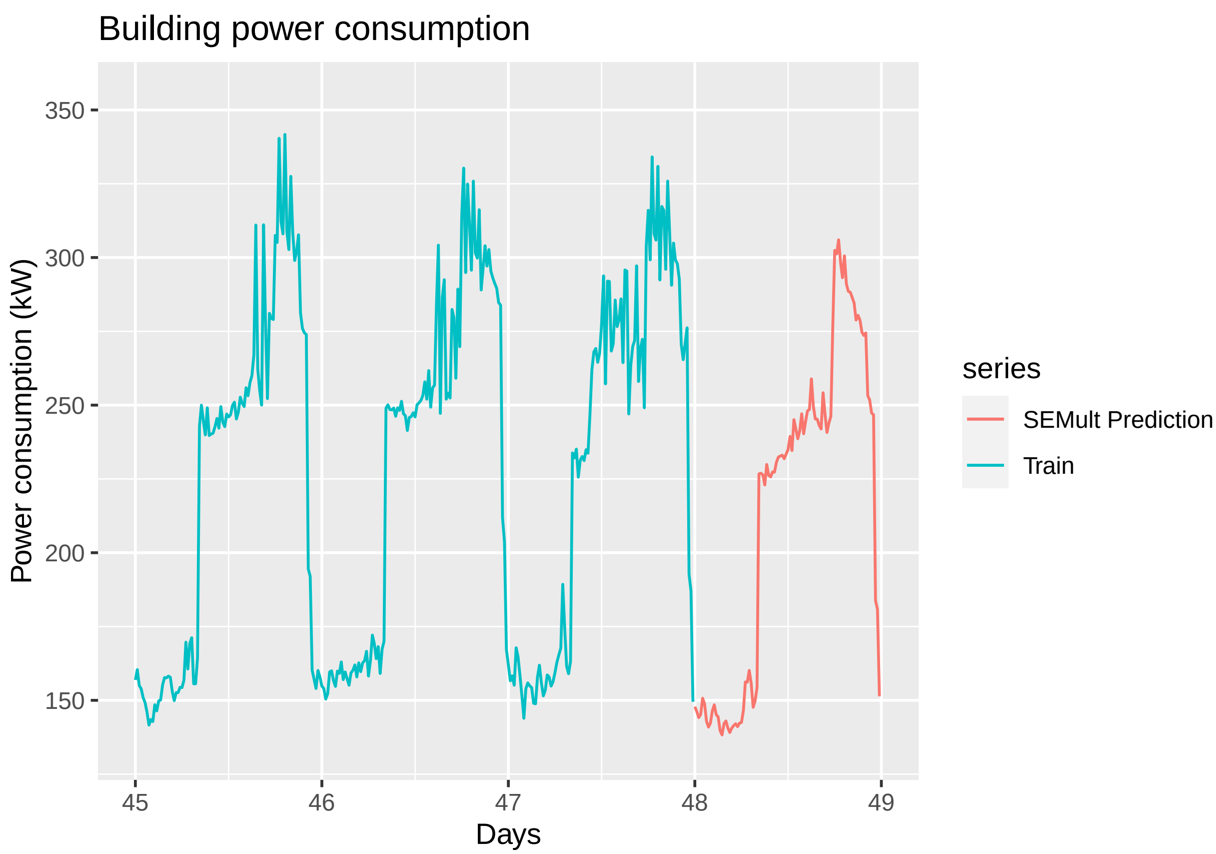
The model I will use for prediction is thus the multiplicative seasonal Holt-Winters.

6) New training on the whole dataset and Prediction

```
model_HW_MultSE_final=HoltWinters(power_ts,alpha=NULL,beta=NULL,gamma=NULL, seasonal = 'multiplicative')
predict_HW_MultSE_final<-predict(model_HW_MultSE_final,n.ahead=96)

autoplot(power_ts,series='Train') +
  autolayer(predict_HW_MultSE_final,series='SEMult Prediction', PI=FALSE)+
  ggtitle ('Building power consumption') +
  xlim(c(45,49)) +
  xlab('Days') +
  ylab('Power consumption (kW)')
```

```
## Warning: Ignoring unknown parameters: PI
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```



II) PREDICTING POWER CONSUMPTION USING OUT-DOOR TEMPERATURE

---

1) Data preparation

1.1) Data importation and plotting

As for power consumption data, I transform temperature data into a time series, with the same frequency, because outdoor temperature also follows a day cycle.

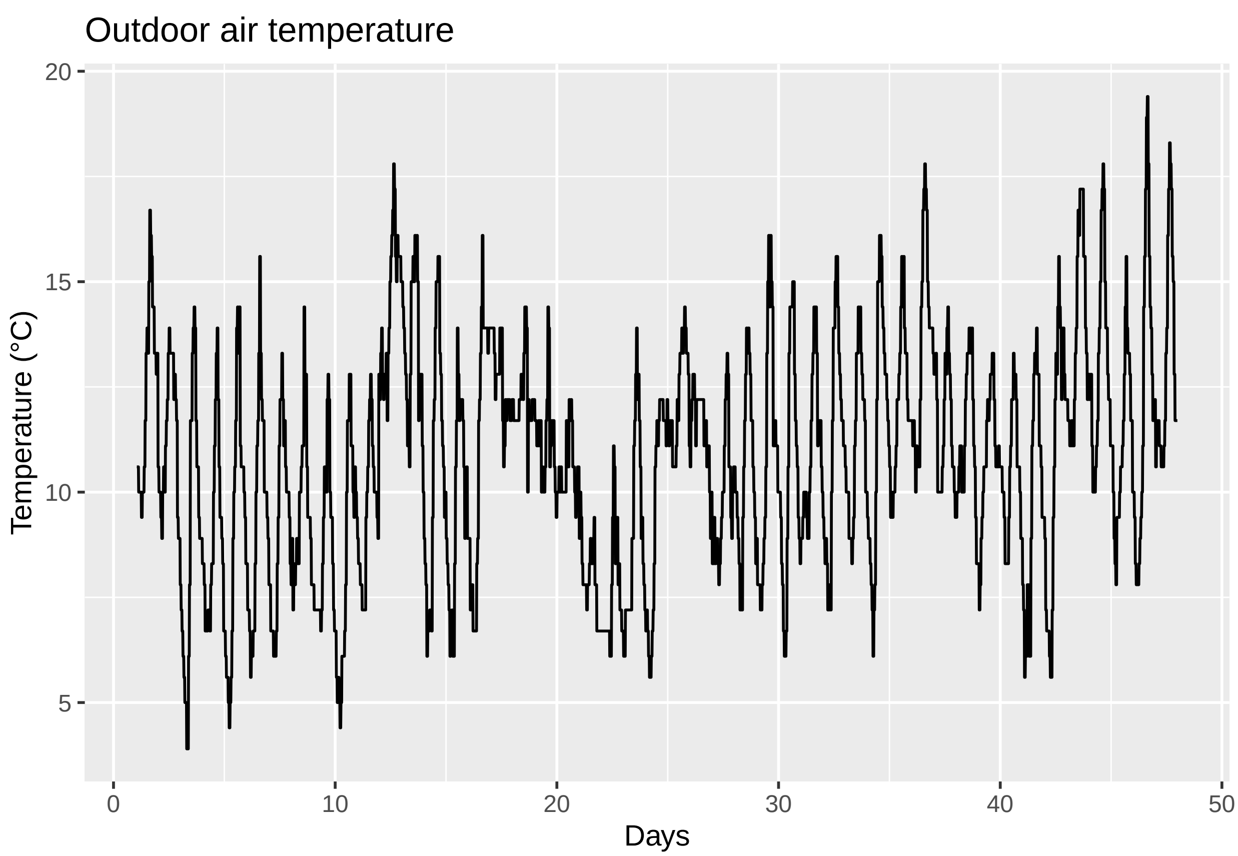
```
temp_ts <- ts(power_df[1:4507,3], frequency = 96, start=c(1,6))
head(temp_ts, 10)
```

```
## Time Series:
## Start = c(1, 6)
## End = c(1, 15)
## Frequency = 96
##  [1] 10.6 10.6 10.6 10.6 10.6 10.6 10.6 10.6 10.0 10.0
```

```r
tail(temp_ts, 10)
```

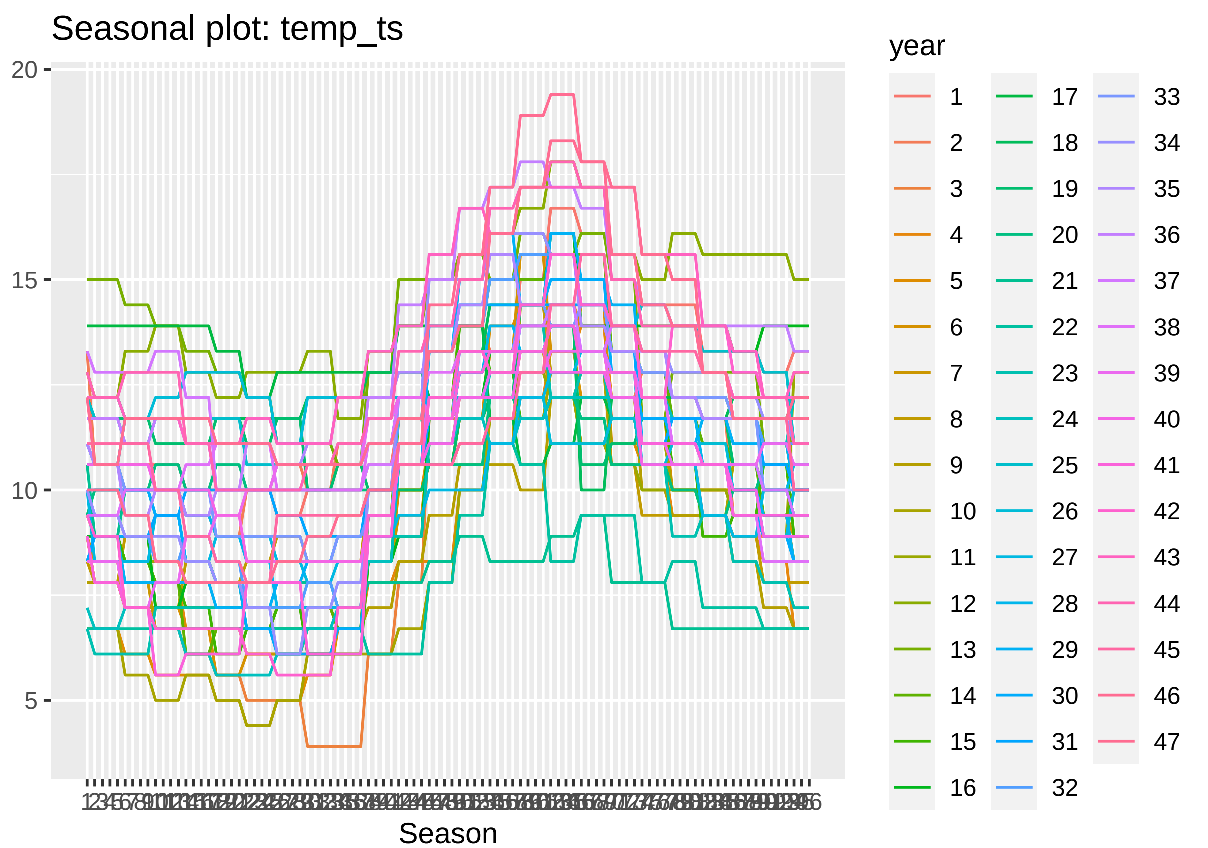
```
## Time Series:
## Start = c(47, 87)
## End = c(47, 96)
## Frequency = 96
##  [1] 11.7 11.7 11.7 11.7 11.7 11.7 11.7 11.7 11.7 11.7
```

```r
autoplot(temp_ts) +
ggtitle('Outdoor air temperature')+
xlab('Days')+
ylab('Temperature (°C)')
```



The season plot shows the behavior of temperature according to the hour of the day:

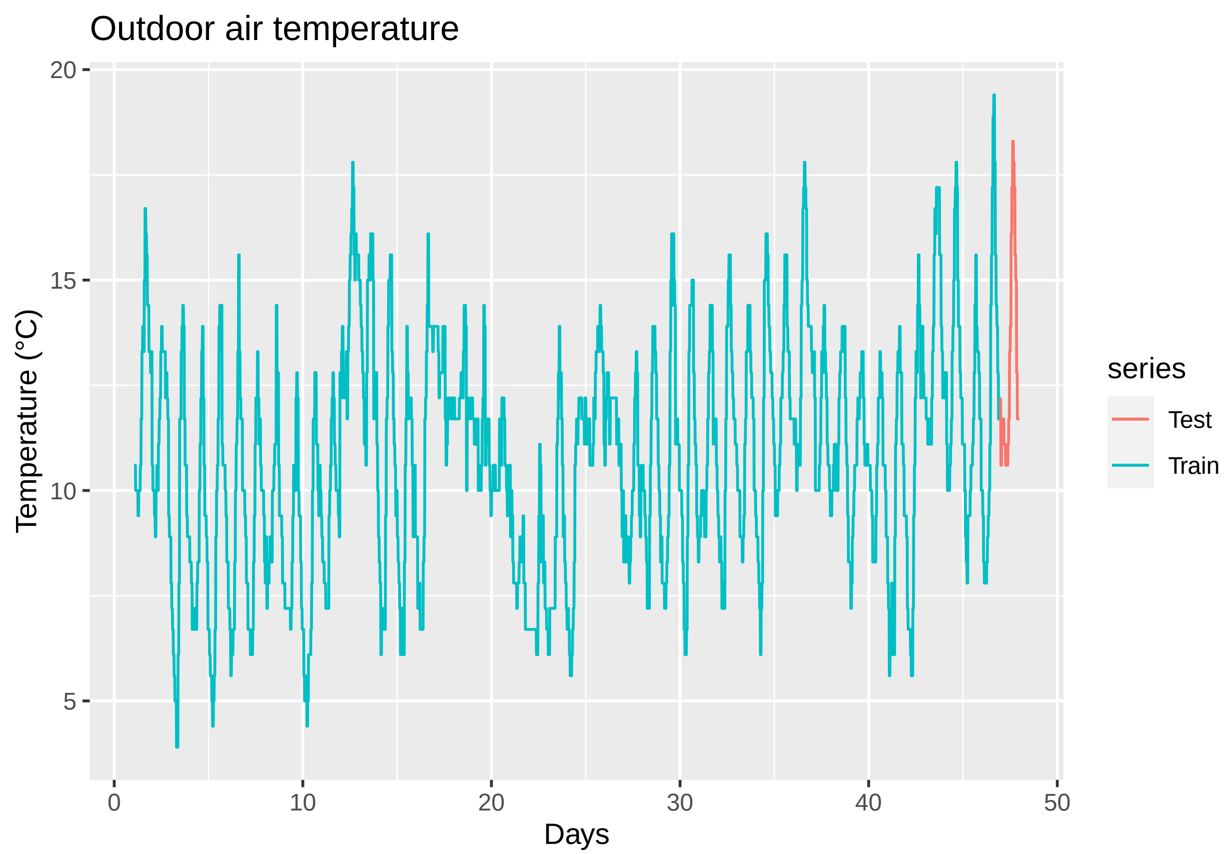
```r
ggseasonplot(temp_ts)
```



1.2) Splitting Train and Test data for model training

As previously, the objective is to predict the power consumption for **24 hours**. As for consumption data, I will thus thus take the last day values (96 values) as my test dataset. The remaining previous values will be my training dataset.

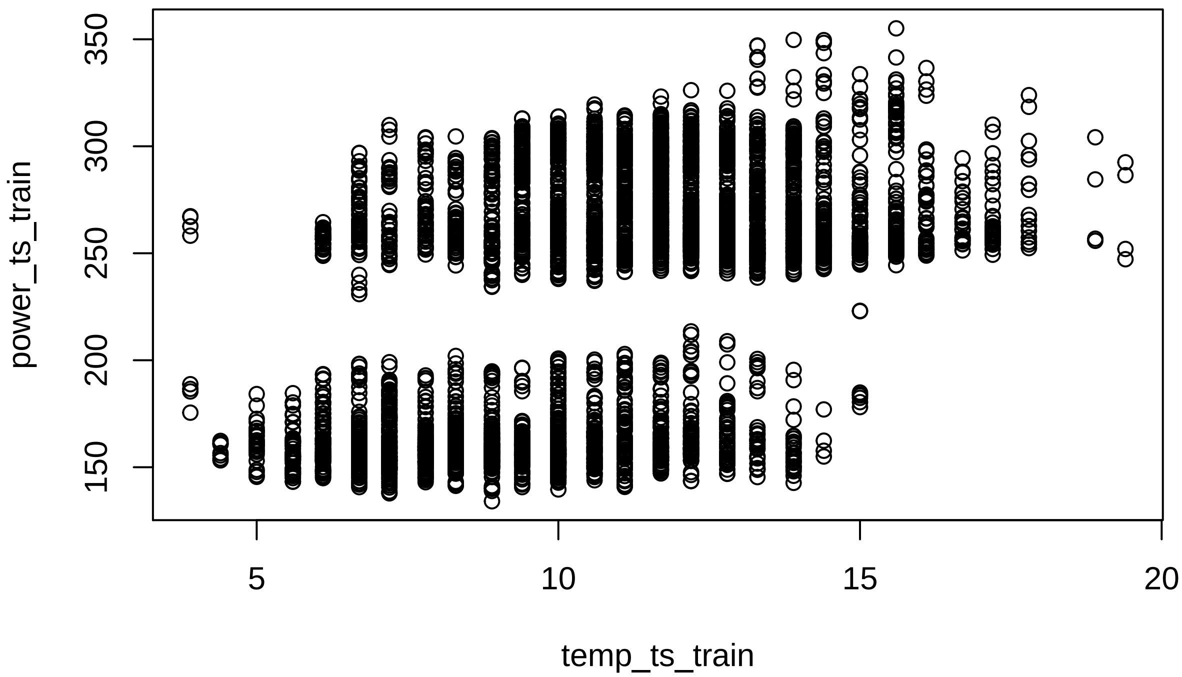
```
temp_ts_train = ts(power_df[1:(46*96-5),3], frequency = 96, start=c(1,6), end=c(46,96))
temp_ts_test = ts(power_df[(46*96-4):4507,3], frequency = 96, start=c(47,1), end=c(47,96) )

autoplot(temp_ts_train,series='Train') +
  autolayer(temp_ts_test,series='Test')+
  ggtitle('Outdoor air temperature')+
  xlab('Days')+
  ylab('Temperature (°C)')
```



2) Regression model

I try to build a first model to predict power consumption, taking into account outdoor temperature. I start to check visually if the relation between both variables looks linear

```
plot(temp_ts_train, power_ts_train)
```



I consider it is the case even there is a strong difference between day and night consumption. I also include in the model the effect of time through a trend and a seasonal effect.

```
linear_model = tslm(power_ts_train~temp_ts_train+trend+season)
summary(linear_model)
```

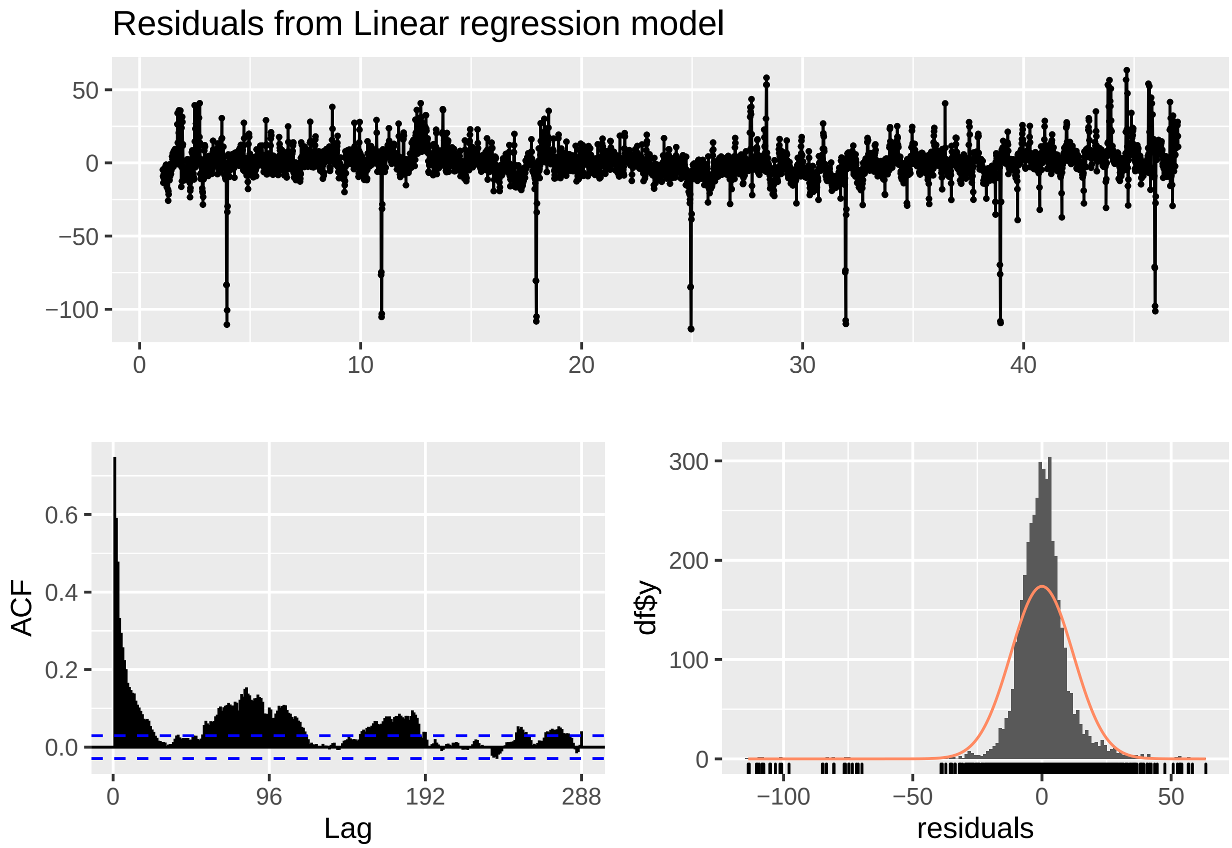
```
##
## Call:
## tslm(formula = power_ts_train ~ temp_ts_train + trend + season)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -113.672   -5.027    0.039    4.845   63.410
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    1.536e+02  2.050e+00  74.959  < 2e-16 ***
## temp_ts_train  1.275e+00  9.566e-02  13.326  < 2e-16 ***
## trend         -3.695e-03  1.507e-04 -24.517  < 2e-16 ***
## season2       -4.662e-01  2.584e+00  -0.180  0.85684
## season3       -6.316e+00  2.584e+00  -2.444  0.01456 *
## season4       -4.121e-01  2.584e+00  -0.159  0.87329
## season5        2.627e+00  2.584e+00   1.017  0.30940
## season6        2.633e+00  2.571e+00   1.024  0.30576
## season7       -6.518e+00  2.571e+00  -2.536  0.01126 *
## season8       -6.681e+00  2.571e+00  -2.599  0.00938 **
```

```
## season9       -3.358e+00  2.571e+00  -1.306  0.19149
## season10      -3.747e+00  2.571e+00  -1.457  0.14512
## season11      -9.624e-01  2.571e+00  -0.374  0.70817
## season12      -1.996e+00  2.571e+00  -0.776  0.43766
## season13      -4.898e-01  2.571e+00  -0.191  0.84891
## season14      -4.825e+00  2.572e+00  -1.876  0.06072 .
## season15      -6.206e+00  2.572e+00  -2.413  0.01587 *
## season16      -6.720e-01  2.572e+00  -0.261  0.79389
## season17       9.904e-01  2.572e+00   0.385  0.70022
## season18       1.891e-01  2.573e+00   0.073  0.94143
## season19      -7.268e-01  2.573e+00  -0.282  0.77762
## season20       8.552e-01  2.573e+00   0.332  0.73966
## season21       5.023e-01  2.573e+00   0.195  0.84523
## season22       1.478e+00  2.574e+00   0.574  0.56580
## season23       2.295e+00  2.574e+00   0.892  0.37266
## season24       4.314e+00  2.574e+00   1.676  0.09383 .
## season25       3.837e+00  2.574e+00   1.491  0.13611
## season26       7.293e+00  2.574e+00   2.833  0.00463 **
## season27       1.491e+01  2.574e+00   5.794 7.36e-09 ***
## season28       1.622e+01  2.574e+00   6.301 3.25e-10 ***
## season29       1.646e+01  2.574e+00   6.395 1.78e-10 ***
## season30       2.149e+01  2.574e+00   8.351  < 2e-16 ***
## season31       2.029e+01  2.574e+00   7.884 4.00e-15 ***
## season32       1.586e+01  2.574e+00   6.160 7.92e-10 ***
## season33       1.956e+01  2.574e+00   7.598 3.66e-14 ***
## season34       1.033e+02  2.573e+00  40.148  < 2e-16 ***
## season35       1.011e+02  2.573e+00  39.302  < 2e-16 ***
## season36       9.853e+01  2.573e+00  38.291  < 2e-16 ***
## season37       9.796e+01  2.573e+00  38.070  < 2e-16 ***
## season38       1.010e+02  2.570e+00  39.306  < 2e-16 ***
## season39       9.568e+01  2.570e+00  37.230  < 2e-16 ***
## season40       9.782e+01  2.570e+00  38.062  < 2e-16 ***
## season41       9.872e+01  2.570e+00  38.413  < 2e-16 ***
## season42       9.498e+01  2.571e+00  36.944  < 2e-16 ***
## season43       9.589e+01  2.571e+00  37.294  < 2e-16 ***
## season44       9.726e+01  2.571e+00  37.828  < 2e-16 ***
## season45       9.762e+01  2.571e+00  37.969  < 2e-16 ***
## season46       9.931e+01  2.576e+00  38.555  < 2e-16 ***
## season47       9.744e+01  2.576e+00  37.826  < 2e-16 ***
## season48       9.812e+01  2.576e+00  38.093  < 2e-16 ***
## season49       9.875e+01  2.576e+00  38.335  < 2e-16 ***
## season50       9.833e+01  2.583e+00  38.065  < 2e-16 ***
## season51       1.017e+02  2.583e+00  39.365  < 2e-16 ***
## season52       1.006e+02  2.583e+00  38.958  < 2e-16 ***
## season53       9.903e+01  2.583e+00  38.337  < 2e-16 ***
## season54       1.005e+02  2.588e+00  38.842  < 2e-16 ***
## season55       1.008e+02  2.588e+00  38.942  < 2e-16 ***
## season56       1.005e+02  2.588e+00  38.826  < 2e-16 ***
## season57       9.922e+01  2.588e+00  38.332  < 2e-16 ***
## season58       9.945e+01  2.595e+00  38.327  < 2e-16 ***
## season59       1.001e+02  2.595e+00  38.559  < 2e-16 ***
## season60       1.003e+02  2.595e+00  38.662  < 2e-16 ***
## season61       1.010e+02  2.595e+00  38.926  < 2e-16 ***
## season62       1.007e+02  2.597e+00  38.765  < 2e-16 ***
```

```
## season63       1.007e+02  2.597e+00  38.765  < 2e-16 ***
## season64       9.974e+01  2.597e+00  38.403  < 2e-16 ***
## season65       9.963e+01  2.597e+00  38.360  < 2e-16 ***
## season66       9.917e+01  2.592e+00  38.267  < 2e-16 ***
## season67       1.010e+02  2.592e+00  38.956  < 2e-16 ***
## season68       9.906e+01  2.592e+00  38.226  < 2e-16 ***
## season69       9.726e+01  2.592e+00  37.529  < 2e-16 ***
## season70       1.168e+02  2.583e+00  45.223  < 2e-16 ***
## season71       1.294e+02  2.583e+00  50.096  < 2e-16 ***
## season72       1.418e+02  2.583e+00  54.917  < 2e-16 ***
## season73       1.440e+02  2.583e+00  55.747  < 2e-16 ***
## season74       1.422e+02  2.576e+00  55.224  < 2e-16 ***
## season75       1.411e+02  2.575e+00  54.769  < 2e-16 ***
## season76       1.406e+02  2.575e+00  54.592  < 2e-16 ***
## season77       1.404e+02  2.575e+00  54.503  < 2e-16 ***
## season78       1.463e+02  2.574e+00  56.839  < 2e-16 ***
## season79       1.433e+02  2.574e+00  55.677  < 2e-16 ***
## season80       1.407e+02  2.574e+00  54.653  < 2e-16 ***
## season81       1.397e+02  2.574e+00  54.292  < 2e-16 ***
## season82       1.410e+02  2.572e+00  54.822  < 2e-16 ***
## season83       1.376e+02  2.572e+00  53.484  < 2e-16 ***
## season84       1.372e+02  2.572e+00  53.330  < 2e-16 ***
## season85       1.373e+02  2.572e+00  53.393  < 2e-16 ***
## season86       1.349e+02  2.571e+00  52.475  < 2e-16 ***
## season87       1.331e+02  2.571e+00  51.787  < 2e-16 ***
## season88       1.318e+02  2.571e+00  51.276  < 2e-16 ***
## season89       1.301e+02  2.571e+00  50.628  < 2e-16 ***
## season90       1.130e+02  2.570e+00  43.957  < 2e-16 ***
## season91       1.112e+02  2.570e+00  43.252  < 2e-16 ***
## season92       1.054e+02  2.570e+00  41.029  < 2e-16 ***
## season93       1.059e+02  2.570e+00  41.190  < 2e-16 ***
## season94       3.096e+01  2.570e+00  12.049  < 2e-16 ***
## season95       3.258e+01  2.570e+00  12.679  < 2e-16 ***
## season96       3.231e+00  2.570e+00   1.257  0.20870
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.26 on 4313 degrees of freedom
## Multiple R-squared:  0.9555, Adjusted R-squared:  0.9545
## F-statistic: 955.5 on 97 and 4313 DF,  p-value: < 2.2e-16
```

Coefficients for outdoor temperature, trend and most of the seasons (mostly season #26 onward) are highly significant. I now check the residuals:

```
checkresiduals(linear_model)
```



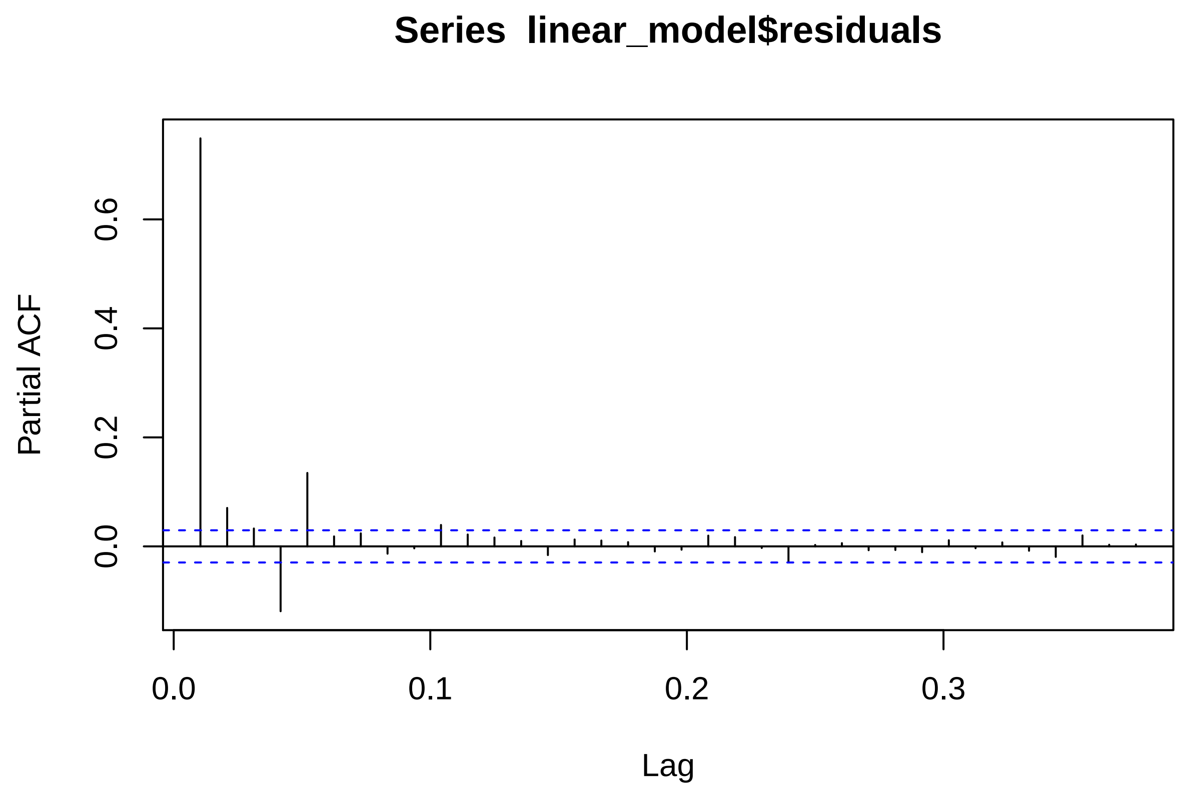

```
##
##  Breusch-Godfrey test for serial correlation of order up to 192
##
## data:  Residuals from Linear regression model
## LM test = 2693, df = 192, p-value < 2.2e-16
```

There is still a trend, probably non linear. The residuals are correlated, so the required assumption for linear regression is not verified. I thus try to fit a SARIMA model based on the PACF.

3) Dynamic regression model

I first look at the PACF to see which order is the most significant.

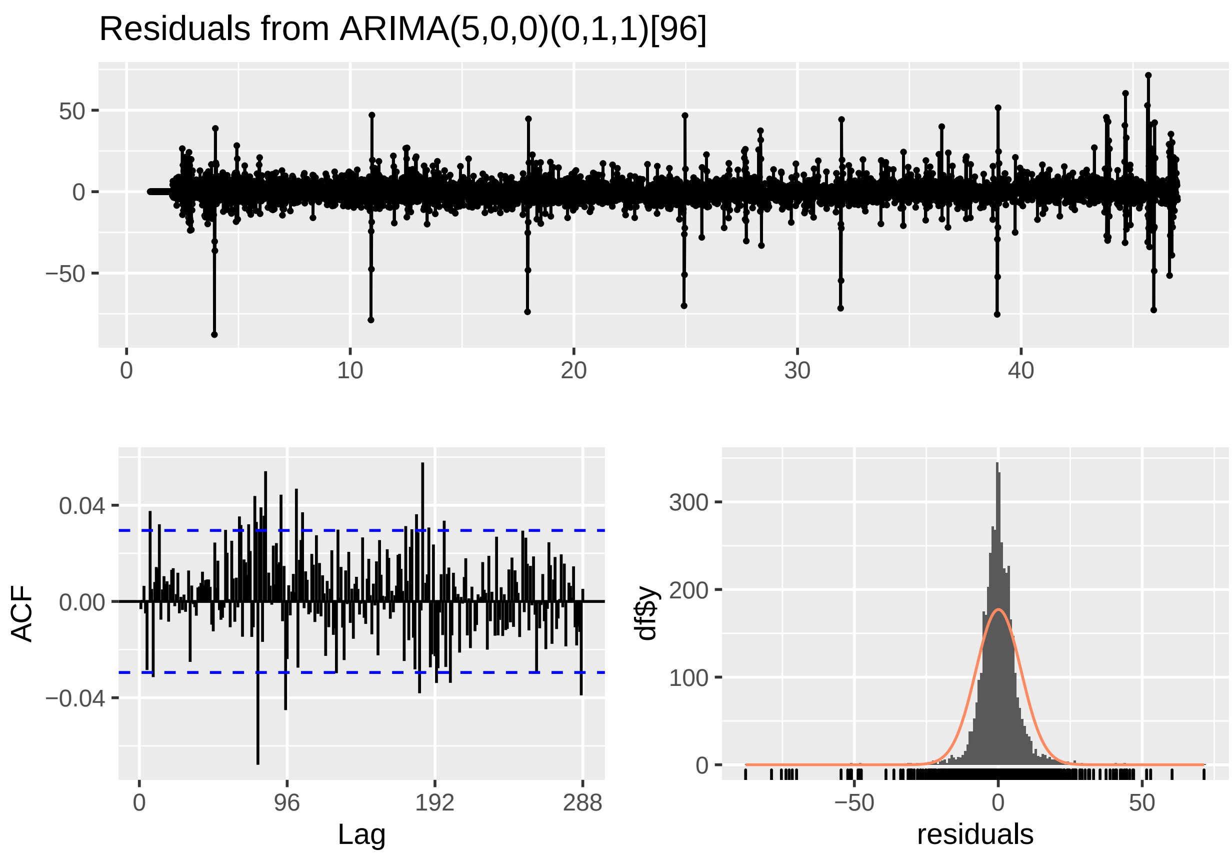
```
pacf(linear_model$residuals)
```



I note the 5th one. So I try to build a SARIMA model on residuals

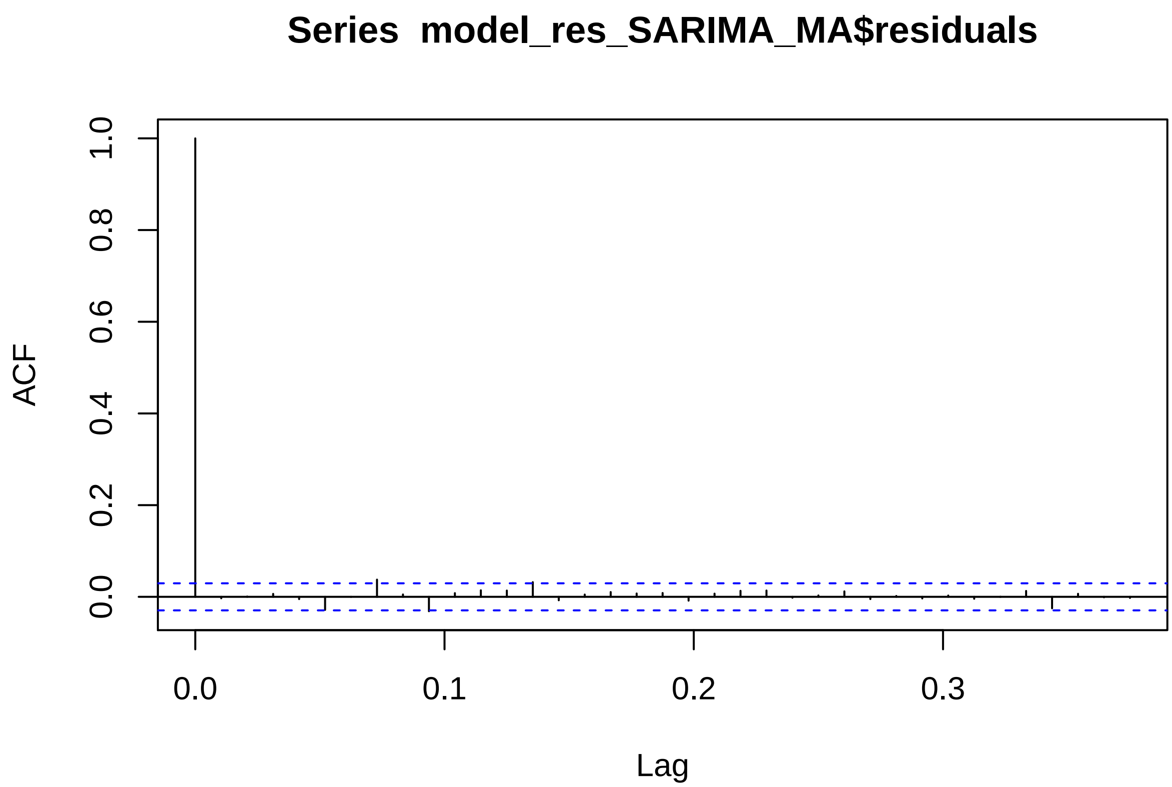
```
model_res_SARIMA_MA = Arima(linear_model$residuals, order = c(5,0,0), seasonal = c(0,1,1))
```

```
checkresiduals(model_res_SARIMA_MA)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(5,0,0)(0,1,1)[96]
## Q* = 316.31, df = 186, p-value = 8.041e-09
##
## Model df: 6.   Total lags used: 192
```

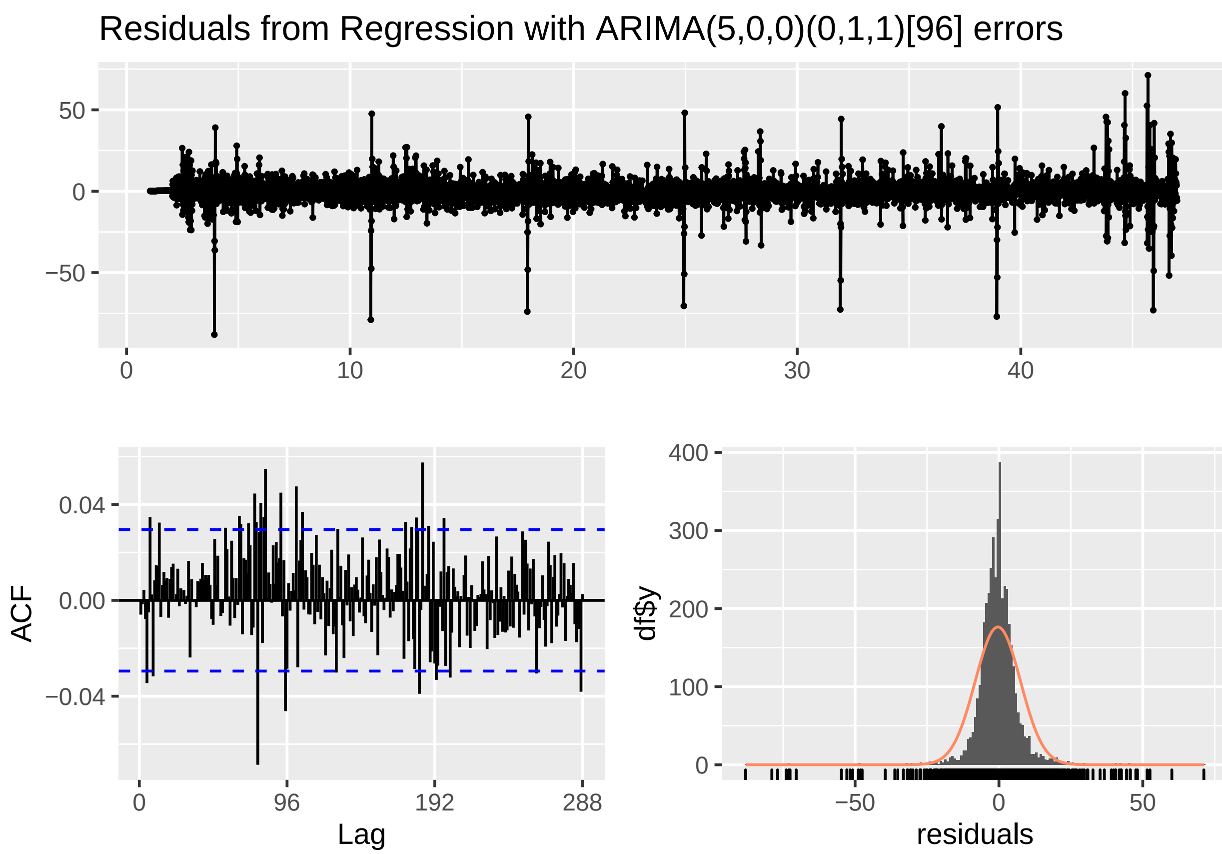
```
acf(model_res_SARIMA_MA$residuals)
```



The Box test is still significant but I didn't succeed to improve it. I use the previous ARIMA model on residuals to build a dynamic regression model with the time series.

```
model_res_SARIMA=Arima(power_ts_train, xreg = temp_ts_train, order = c(5,0,0), seasonal = c(0,1,1))
```

```
checkresiduals(model_res_SARIMA)
```

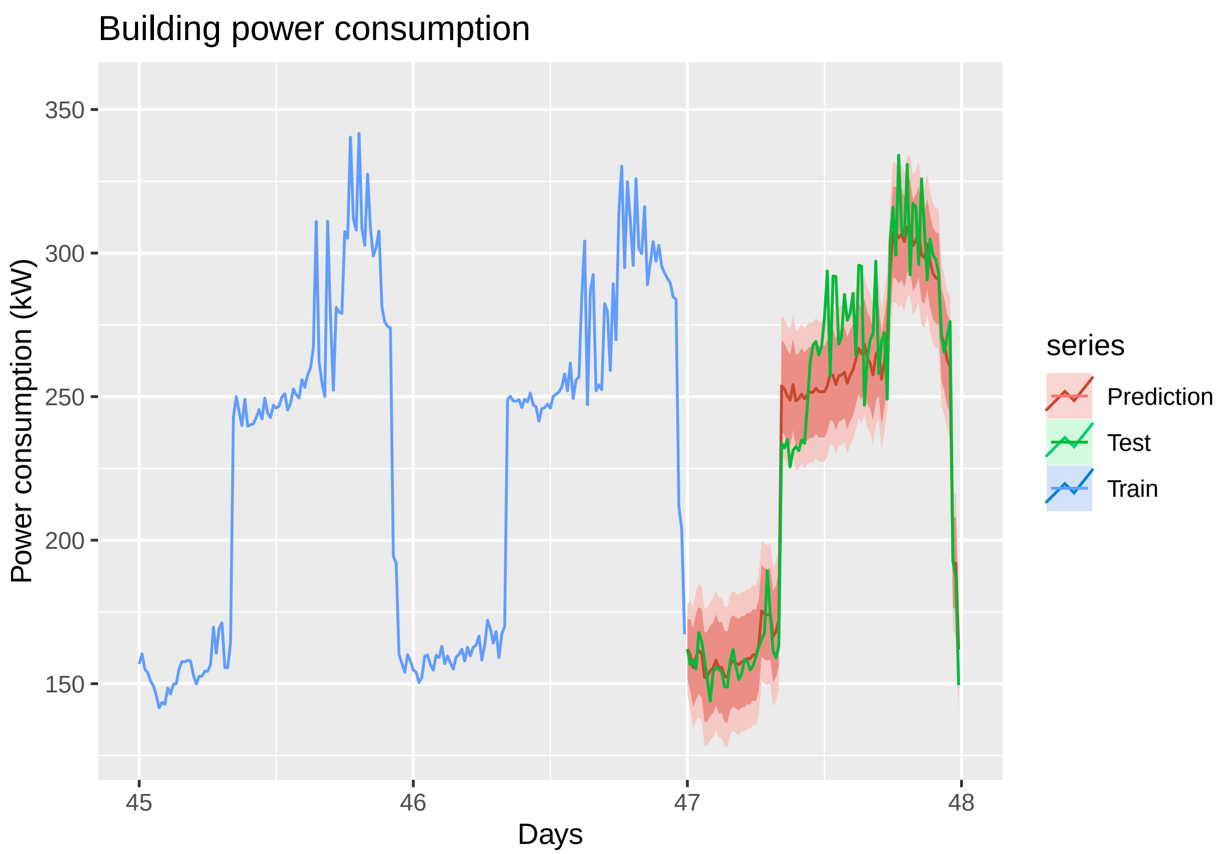



```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(5,0,0)(0,1,1)[96] errors
## Q* = 324, df = 185, p-value = 1.139e-09
##
## Model df: 7.   Total lags used: 192
```

I now compute mu RMSE to be compared with other models.

```
predict_model_res_SARIMA = forecast(model_res_SARIMA,h=96,xreg=temp_ts_test)

autoplot(power_ts_train,series='Train') +
  autolayer(predict_model_res_SARIMA,series='Prediction', PI=TRUE)+
  autolayer(power_ts_test,series='Test')+
  ggtitle ('Building power consumption') +
  xlim(c(45,48)) +
  xlab('Days') +
  ylab('Power consumption (kW)')
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```




```
#Root mean square error
model_res_SARIMA_RMSE = sqrt(mean((predict_model_res_SARIMA$mean-power_ts_test)^2))
model_res_SARIMA_RMSE
```

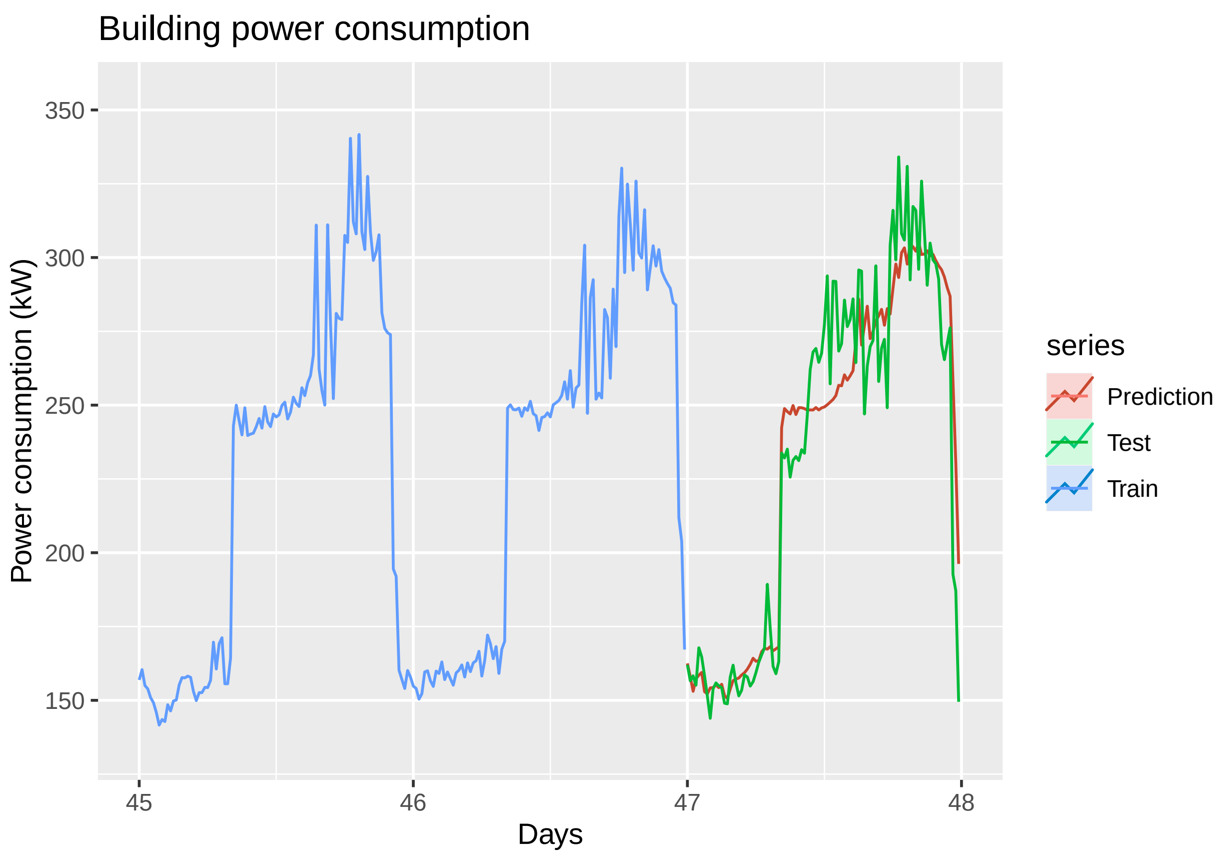
```
## [1] 14.82086
```

3) Neural network model

I use the same methodology as above to build this model.

```
model_nnar_temp = nnetar(power_ts_train, xreg = temp_ts_train)
predict_nnar_temp = forecast(model_nnar_temp,h=96, xreg = temp_ts_test)

autoplot(power_ts_train, series='Train') +
  autolayer(predict_nnar_temp, series='Prediction', PI=TRUE)+
  autolayer(power_ts_test, series='Test')+
  ggtitle ('Building power consumption') +
  xlim(c(45,48)) +
  xlab('Days') +
  ylab('Power consumption (kW)')
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```




```
#Root mean square error
model_nnar_temp_RMSE = sqrt(mean((predict_nnar_temp$mean-power_ts_test)^2))
model_nnar_temp_RMSE
```

```
## [1] 18.60264
```

The RMSE is less good than for dynamic regression model. I will thus use the latter to predict power consumption.

4) New training on the whole dataset and Prediction

I use the same methodology as above: I retrain the model on the whole dataset then perform my prediction.

```
model_res_SARIMA_final=Arima(power_ts, xreg = temp_ts, order = c(5,0,0), seasonal = c(0,1,1))

predict_res_SARIMA_final<-forecast(model_res_SARIMA_final,h=96, xreg = temp_ts)

autoplot(power_ts,series='Train') +
  autolayer(predict_res_SARIMA_final,series='Prediction', PI=TRUE)+
  ggtitle ('Building power consumption') +
  xlim(c(45,49)) +
  xlab('Days') +
  ylab('Power consumption (kW)')
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```