Electricity Consumption Forecasting

Constantin REY-COQUAIS

January 2026

Contents

1 Part 0 : Data exploration **1**

2 Part 1 Forecasting **8**

2.1 Comparison frame 8

2.2 Determinist models 10

2.3 Stochastic models 13

2.4 Neural Net 16

2.5 Forecast of the 2/17/2010 18

1 Part 0 : Data exploration

```
library(ReyCoquaisConstantin)
library(readxl)
library(ggplot2)
```

```
## Warning: le package 'ggplot2' a été compilé avec la version R 4.5.2
```

```
library(tseries)
```

```
## Warning: le package 'tseries' a été compilé avec la version R 4.5.2
```

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

```
library(patchwork)
```

```
## Warning: le package 'patchwork' a été compilé avec la version R 4.5.2
```

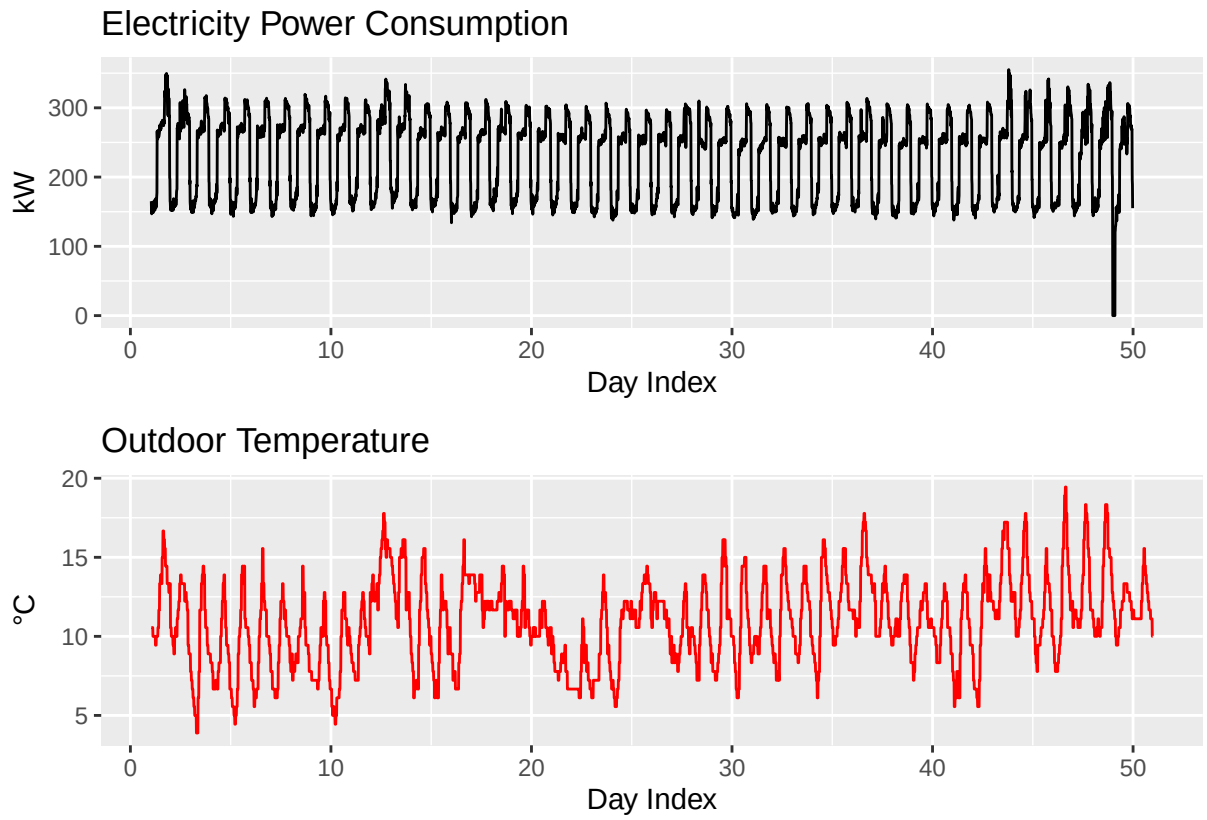
```r
library(forecast)
```

```
## Warning: le package 'forecast' a été compilé avec la version R 4.5.2
```

```r
# Config
DATA_PATH <- system.file("extdata", "Elec-train.xlsx", package = "ReyCoquaisConstantin")

# each point is separated by 15 minutes
# To study a daily seasonal pattern, we take 24*4 = 96 points into consideration
FREQ_DAILY <- 96
```

```r
# Load data
raw_data <- read_xlsx(DATA_PATH)

# Creation of Time Series objects
# Our data range from Jan, 1st 2010 1:15 am to Feb, 18th 11:45 pm
elec_ts <- ts(raw_data$`Power (kW)`, frequency = FREQ_DAILY, start = c(1, 5)) # 1:15 is the 5th measure
temp_ts <- ts(raw_data$`Temp (C°)`, frequency = FREQ_DAILY, start = c(1, 5))
```

```r
p1 <- autoplot(elec_ts) +
  labs(title = "Electricity Power Consumption", y = "kW", x = "Day Index")

p2 <- autoplot(temp_ts, colour = "red") +
  labs(title = "Outdoor Temperature", y = "°C", x = "Day Index")

p1 / p2
```



We identify an "outlier" just before the 50th day, so let's have a closer look on this area to possibly clean it up

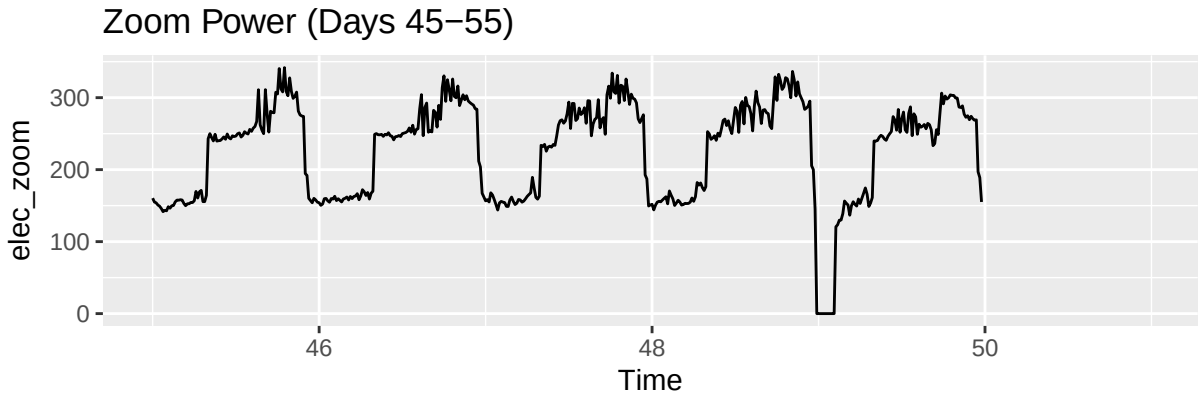
```
elec_zoom <- window(elec_ts, start = 45, end = 55)
```

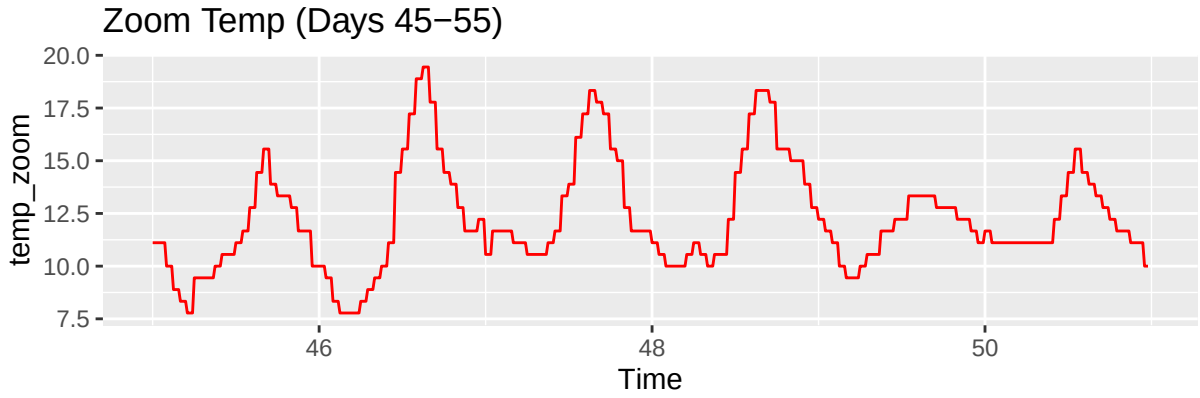
```
## Warning in window.default(x, ...): valeur de 'end' inchangée
```

```
temp_zoom <- window(temp_ts, start = 45, end = 55)
```

```
## Warning in window.default(x, ...): valeur de 'end' inchangée
```

```
p_zoom_elec <- autoplot(elec_zoom) + labs(title = "Zoom Power (Days 45-55)")
p_zoom_temp <- autoplot(temp_zoom, colour = "red") + labs(title = "Zoom Temp (Days 45-55)")

p_zoom_elec / p_zoom_temp
```





Well, according to these plots, we have a series of points that are way under the usual minimum of consumption in the dataset. It occurs on the 49th day which corresponds to the night between the 17th and 18th of February.The temperature curve does not showcase any special event.

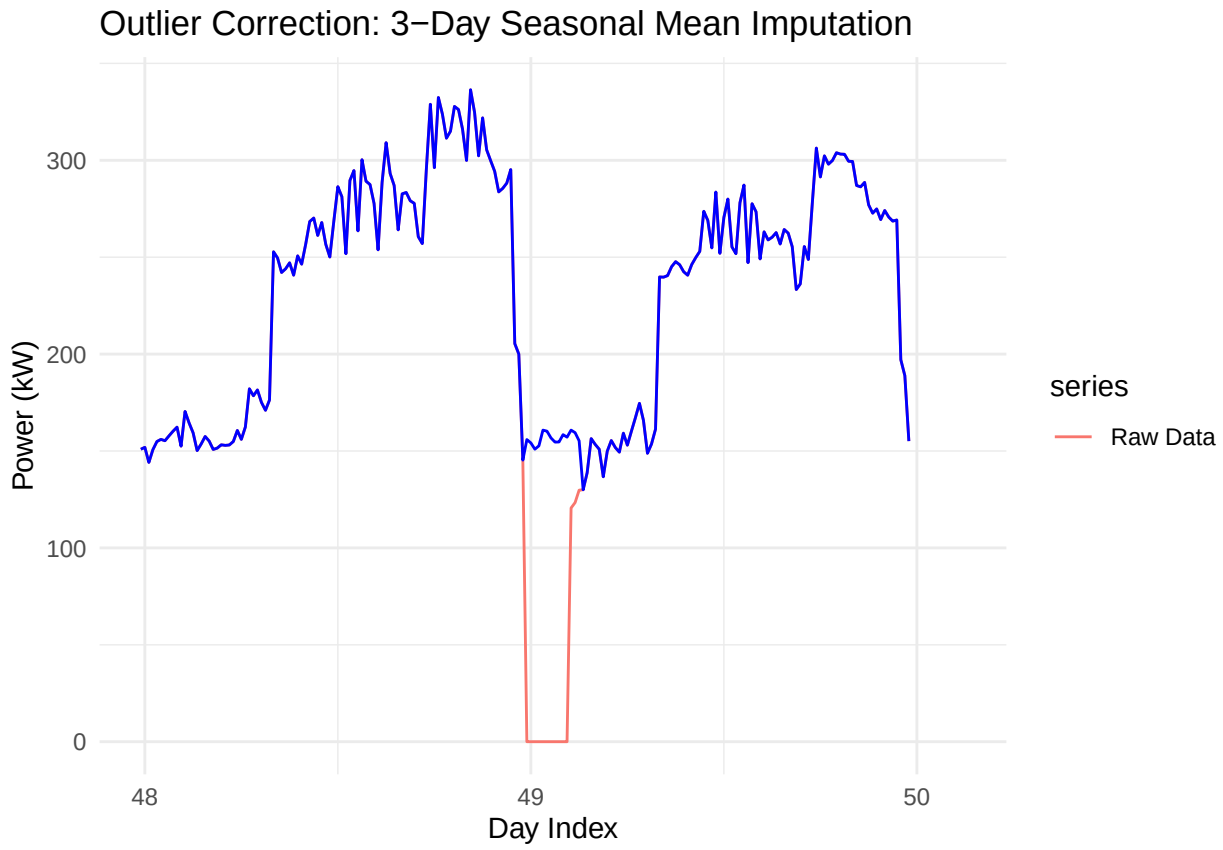
We are likely to use Exponential smoothing methods among others to predict the consumption on the 19th. Some of these methods are sensitive to the last values of the dataset to predict and I fear that these outliers might misguide our training and predictions. I think that there may have been an event such as heating failure, or electricity shortage that is responsible for these outliers and I chose to regularize them to protect the predictions of our forecasting models.

We are going to identify the indexes of these low values. Then to regularize them, we are going to preserve the daily seasonality by imputing them the average value of the consumption of the 3 last days at this hour.

```
idx_outliers <- which(elec_ts < 130)
elec_cleaned <- elec_ts

for (i in idx_outliers) {
  # Indices for the same time in the 3 previous days
  lags <- i - (1:3 * 96)

  # Impute using the mean of these 3 points
  elec_cleaned[i] <- mean(elec_ts[lags])
}

# Let's display the identified period
plot_start <- min(time(elec_ts)[idx_outliers]) - 1
plot_end   <- max(time(elec_ts)[idx_outliers]) + 1
```

```
autoplot(window(elec_ts, start = plot_start, end = plot_end), series = "Raw Data") +
  autolayer(window(elec_cleaned, start = plot_start, end = plot_end),
            series = "Cleaned (3-day Mean)", color = "blue") +
  labs(title = "Outlier Correction: 3-Day Seasonal Mean Imputation",
       y = "Power (kW)", x = "Day Index") +
  theme_minimal()
```

```
## Warning: Removed 14 rows containing missing values or values outside the scale range
## (`geom_line()`).
```



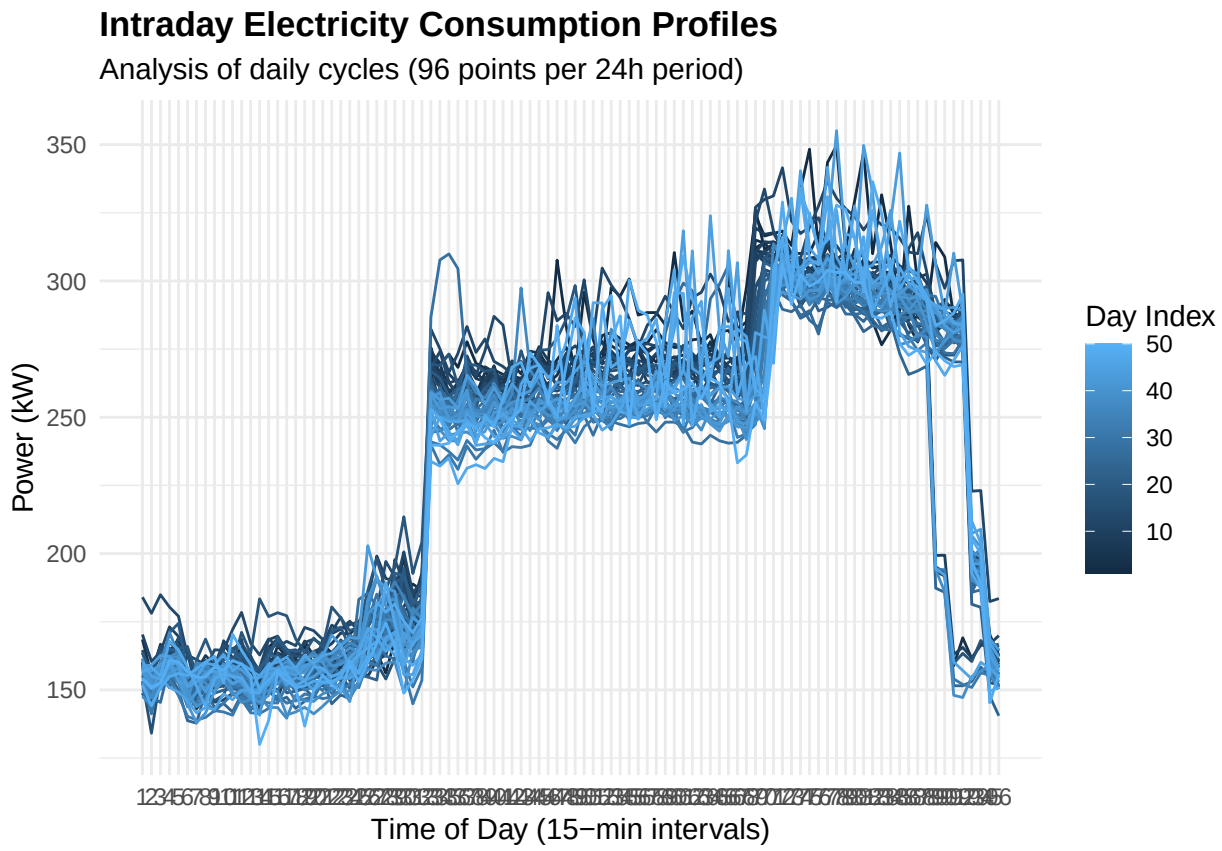
We will now work with the *elec_cleaned* time series. Let's display the season plot (with a frequency of 96, it will be the daily seasonality that will be showcased)

```
ggseasonplot(elec_cleaned, year.labels = FALSE, continuous = TRUE) +
  theme_minimal() +
  labs(
    title = "Intraday Electricity Consumption Profiles",
    subtitle = "Analysis of daily cycles (96 points per 24h period)",
    y = "Power (kW)",
    x = "Time of Day (15-min intervals)",
    color = "Day Index"
  ) +
  theme(
    legend.position = "right",
```

```
    plot.title = element_text(face = "bold")
  )
```

```
## Warning in fortify(data, ...): Arguments in `...` must be used.
## x Problematic argument:
## * na.rm = TRUE
## i Did you misspell an argument name?
```

```
## Warning: Removed 96 rows containing missing values or values outside the scale range
## (`geom_line()`).
```



We clearly see a daily pattern consisting in a plateau by night, a first jump in the morning (arround 8:00 am) plateauing again and an ultimate increase after 4:00 pm decreasing afterwards.

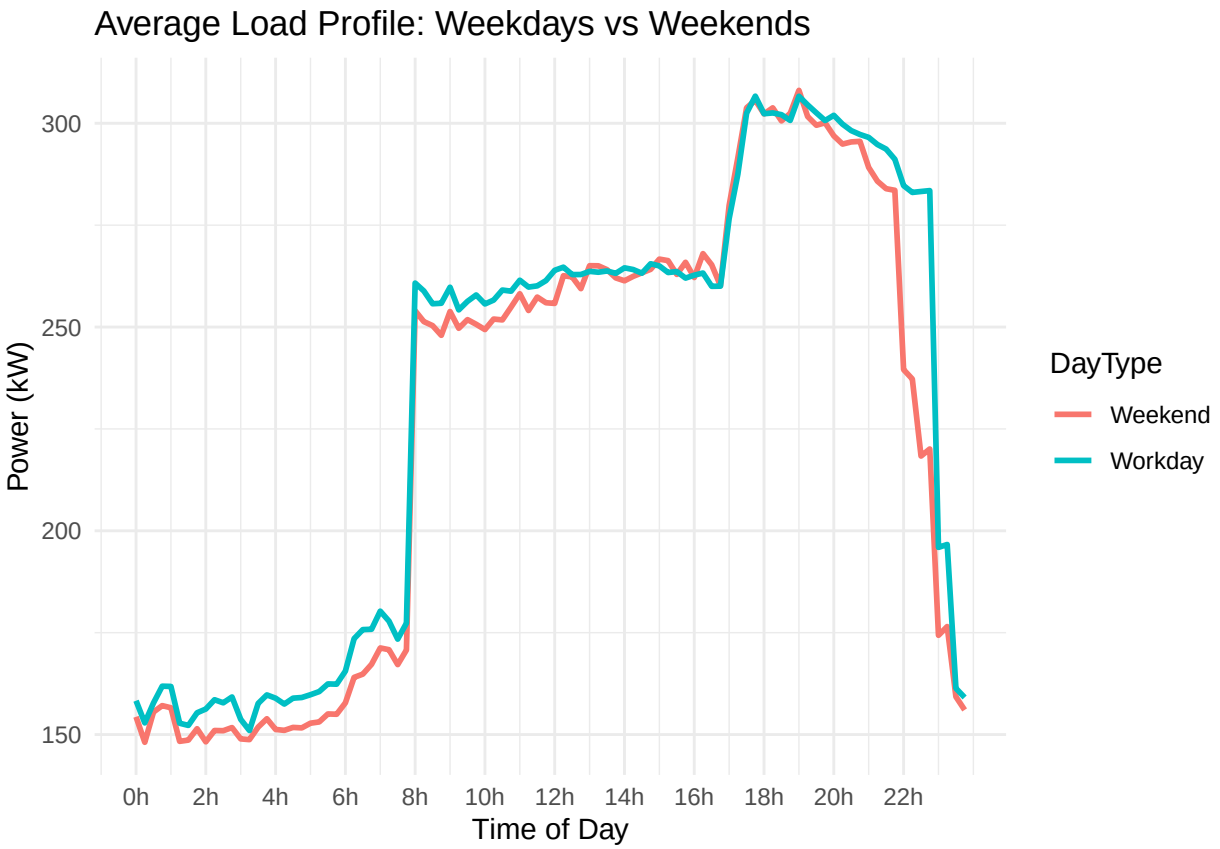
We have the impression that the first days are a bit more intense in terms of consumption. It could be linked to maybe colder temperatures in the beginning of January followed by warmer days. In fact, when plotting the same chart for temperatures, we cannot infer lots of information. We'll let the forecasting models identify these patterns.

Before moving to data modelisation, we are going to see if we identify a weekly seasonnality. In electricity consumption curves, it is common that the weekends stand out from the working days. It could guide us in our hyperparameter fine-tuning in the future (and we are curious about our data)

```
n_points <- length(elec_cleaned)

# First point is at 01:15 (index 5 of the day)
```

```r
# We initialize a sequence starting on 5 and iterating over 96
time_in_day <- ((0:(n_points - 1) + 4) %% 96) + 1

# The first day has 92 points
points_first_day <- 96 - 4

# Creation of a day vector
# Day 1 is a Friday so we put "Workday" and then we iterate
day_vector <- c(rep("Workday", points_first_day),
                rep(c("Weekend", "Weekend", "Workday", "Workday", "Workday", "Workday", "Workday"),
                    each = 96, length.out = n_points - points_first_day))

# assert the vectors have same length
day_vector <- day_vector[1:n_points]

df_analysis <- data.frame(
  Power = as.numeric(elec_cleaned),
  TimeInDay = time_in_day,
  DayType = day_vector
)

ggplot(df_analysis, aes(x = TimeInDay, y = Power, color = DayType)) +
  stat_summary(fun = mean, geom = "line", size = 1) +
  scale_x_continuous(breaks = seq(1, 96, by = 8),
                     labels = paste0(seq(0, 23, by = 2), "h")) +
  theme_minimal() +
  labs(title = "Average Load Profile: Weekdays vs Weekends",
       x = "Time of Day", y = "Power (kW)")
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.

## Warning: Removed 96 rows containing non-finite outside the scale range
## (‘stat_summary()‘).
```



From what we see on the chart, there is no significant difference between the weekends and the workdays. We'll further check with the PACF and other indicators but we seem to have the consumption of maybe an industrial or company that works 7/7 days

2 Part 1 Forecasting

2.1 Comparison frame

Let's now define the frame that will allow us to estimate the performance of the models we will implement and compare. Because we want to fine-tune their hyper-parameters, we are going to implement an **expanding window Cross-Validation** on 7 folds (for the 7 days of the week)

We are going to evaluate them on the Root Mean Squared Error (RMSE), to penalize the extreme values and on the Mean Absolute Pourcentage Error (MAPE) for more interpretable indicator.

```
elec_train <- window(elec_cleaned,end=c(48,96))
elec_test <- window(elec_cleaned, start = c(49, 1), end = c(49, 96))

temp_train <- window(temp_ts,end=c(48,96))
temp_test <- window(temp_ts, start = c(49, 1), end = c(49, 96))

# Vérification (doit afficher 96)
print(paste("Nouvelle taille elec_test :", length(elec_test)))
```

```
## [1] "Nouvelle taille elec_test : 96"
```

```r
# Performance Metrics Function we'll call after each forecast
accuracy_metrics <- function(forecast_val, real_val) {
  rmse_val <- sqrt(mean((real_val - forecast_val)^2))
  mape_val <- mean(abs((real_val - forecast_val) / real_val)) * 100
  return(c(RMSE = rmse_val, MAPE = mape_val))
}

cross_validate <- function(ts_data, forecast_func, h = 96, n_folds = 7) {
  n <- length(ts_data)
  mapes <- c()
  rmses <- c()

  for (i in 1:n_folds) {
    # This fold's training ends i days (i*h points) before the last data point
    train_end <- n - i * h

    train_set <- subset(ts_data, end = train_end)
    test_set  <- subset(ts_data, start = train_end + 1, end = train_end + h)

    fit <- forecast_func(train_set, h)

    metrics <- accuracy_metrics(fit$mean, test_set)
    mapes <- c(mapes, metrics["MAPE"])
    rmses <- c(rmses, metrics["RMSE"])
  }

  return(c(MAPE = mean(mapes), RMSE = mean(rmses)))
}
```

We then define an auxiliary structure, a function to display a "leaderboard" of our trained models and a function to plot the prediction on the last day of data

```r
model_comparison <- data.frame(
  Model = character(),
  MAPE = numeric(),
  RMSE = numeric(),
  stringsAsFactors = FALSE
)

# Helper to register results
register_model <- function(df, name, metrics) {
  new_row <- data.frame(Model = name, MAPE = metrics["MAPE"], RMSE = metrics["RMSE"])
  return(rbind(df, new_row) |> dplyr::arrange(MAPE))
}

plot_leader <- function(test_set, forecast_obj, model_name, metrics) {

  # Construction du sous-titre avec les métriques arrondies
  sub_text <- paste0(model_name, " | MAPE: ", round(metrics["MAPE"], 2),
                     "% | RMSE: ", round(metrics["RMSE"], 2))

  autoplot(test_set, series = "Real Data", size = 0.8) +
    autolayer(forecast_obj, series = model_name, PI = FALSE, size = 1) +
```

```
    scale_color_manual(values = c("Real Data" = "black", model_name = "blue"),
                       breaks = c("Real Data", model_name)) +
    labs(
      title = "Champion Model Performance vs Reality",
      subtitle = sub_text,
      x = "Time Index",
      y = "Power (kW)",
      color = "Series"
    ) +
    theme_minimal() +
    theme(legend.position = "bottom",
          plot.title = element_text(face = "bold"))
}
```

2.2 Determinist models

Because we have a high frequency here *96*, the classic *hw()* models from *forecast* cannot be used (they are capped to a frequency < 24). We will be using the stlf() of *forecast* with the *ets* methods

```
# First, a baseline which just copies the day before to predict
fit_snaive <- function(train, h) { snaive(train, h = h) }

# 2. Holt-Winters via STL (AAA stands for Additive for the Error, the Trend and the Seasonality)
fit_hw_add <- function(train, h) {
  stlf(train, method = "ets", etsmodel = "AAA", h = h)
}

# 3. Holt-Winters via STL (Multiplicative for the Error and Seasonality)
fit_hw_mult <- function(train, h) {
  stlf(train, method = "ets", etsmodel = "MAM", h = h)
}

# 4. Additive Holt-Winters Damped
fit_hw_damped <- function(train, h) {
  stlf(train, method = "ets", etsmodel = "AAA", damped = TRUE, h = h)
}
```

Evaluation

```
# Baseline
metrics_snaive <- cross_validate(elec_train, fit_snaive)
model_comparison <- register_model(model_comparison, "Seasonal Naive", metrics_snaive)

# Holt-Winters Additive
metrics_hw_add <- cross_validate(elec_train, fit_hw_add)
```

```
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
```

```
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
```

```
model_comparison <- register_model(model_comparison, "HW Additive", metrics_hw_add)

# Holt-Winters Multiplicative
metrics_hw_mult <- cross_validate(elec_train, fit_hw_mult)
```

```
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
```

```
model_comparison <- register_model(model_comparison, "HW Multiplicative", metrics_hw_mult)

# Holt-Winters Additive Damped
metrics_hw_damped <- cross_validate(elec_train, fit_hw_damped)
```

```
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
```

```
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
```

```
model_comparison <- register_model(model_comparison, "HW Additive Damped", metrics_hw_damped)


print(model_comparison)
```

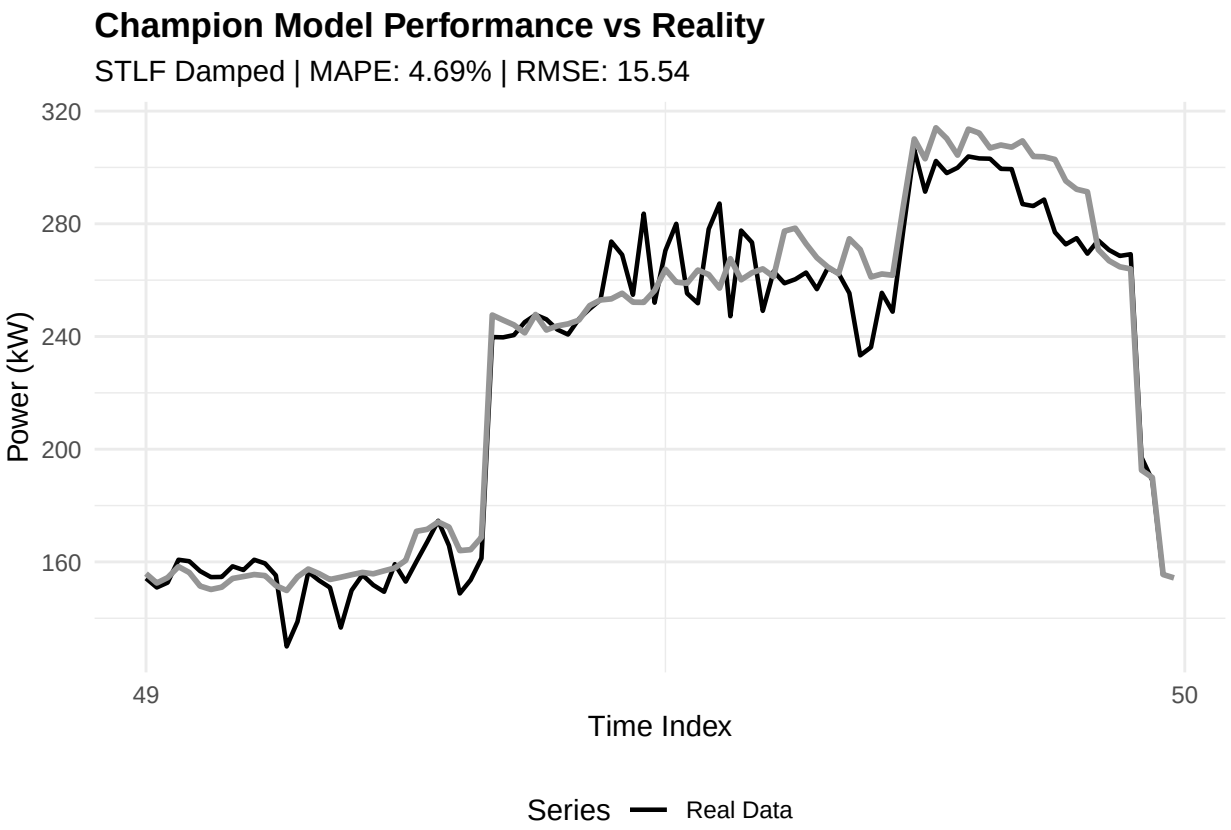
```
##                    Model     MAPE     RMSE
## MAPE3 HW Additive Damped 4.693817 15.54019
## MAPE1        HW Additive 4.714741 15.59412
## MAPE      Seasonal Naive 5.045776 18.07972
## MAPE2  HW Multiplicative 6.092145 17.37818
```

The Damped Holt-Winters seems to perform slightly better thant the others, let's visualise its prediction on the last day of data

```
h <- 96
leader_fit <- stlf(elec_train, method = "ets", etsmodel = "AAA", damped = TRUE, h = h)
```

```
## Warning in forecast.stl(object, method = method, etsmodel = etsmodel,
## forecastfunction = forecastfunction, : The ETS model must be non-seasonal. I'm
## ignoring the seasonal component specified.
```

```
leader_metrics <- model_comparison[1, c("MAPE", "RMSE")]

plot_leader(elec_test, leader_fit, "STLF Damped", leader_metrics)
```



2.3 Stochastic models

We will be implementing an auto.arima to see its performance and then we are going to try to fine-tune an arima ourselves to see if we can outperform thanks to the analysis of the auto-correlations.

```
# Wrapper for standard auto.arima
fit_auto_sarima <- function(train, h) {
  # approximation=TRUE and stepwise=TRUE for higher convergence speed
  fit <- auto.arima(train, seasonal = TRUE, stepwise = TRUE, approximation = TRUE)
  forecast(fit, h = h)
}

metrics_auto <- cross_validate(elec_train, fit_auto_sarima)
model_comparison <- register_model(model_comparison, "Auto SARIMA", metrics_auto)
```

```
print(model_comparison)
```

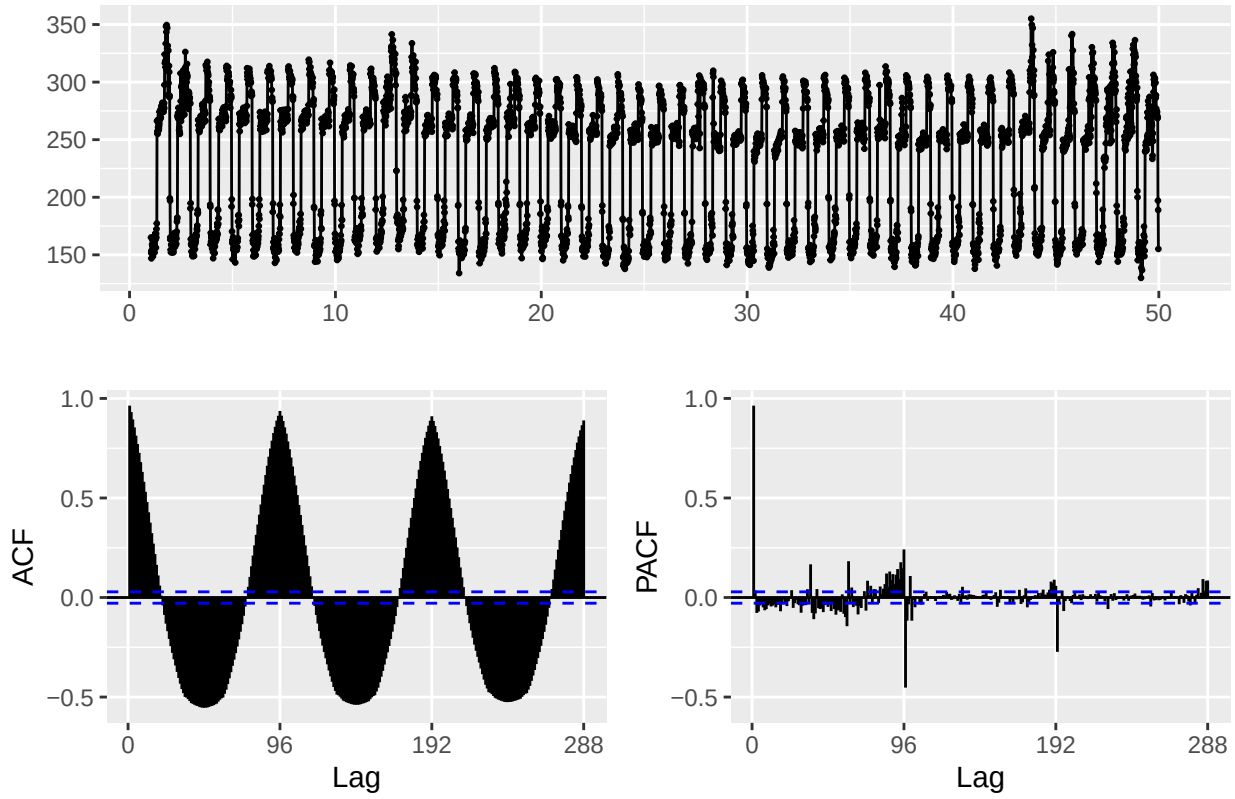
```
##                     Model     MAPE     RMSE
## MAPE3 HW Additive Damped 4.693817 15.54019
## MAPE1        HW Additive 4.714741 15.59412
## MAPE      Seasonal Naive 5.045776 18.07972
## MAPE4        Auto SARIMA 5.050779 18.08266
## MAPE2  HW Multiplicative 6.092145 17.37818
```

After an eternity of fitting, it performs under the naive model... Maybe the high frequency combined with very high auto-correlations complicated its learning.

Let's now analyze our time series to identify hyper-parameters to test for our sarima :

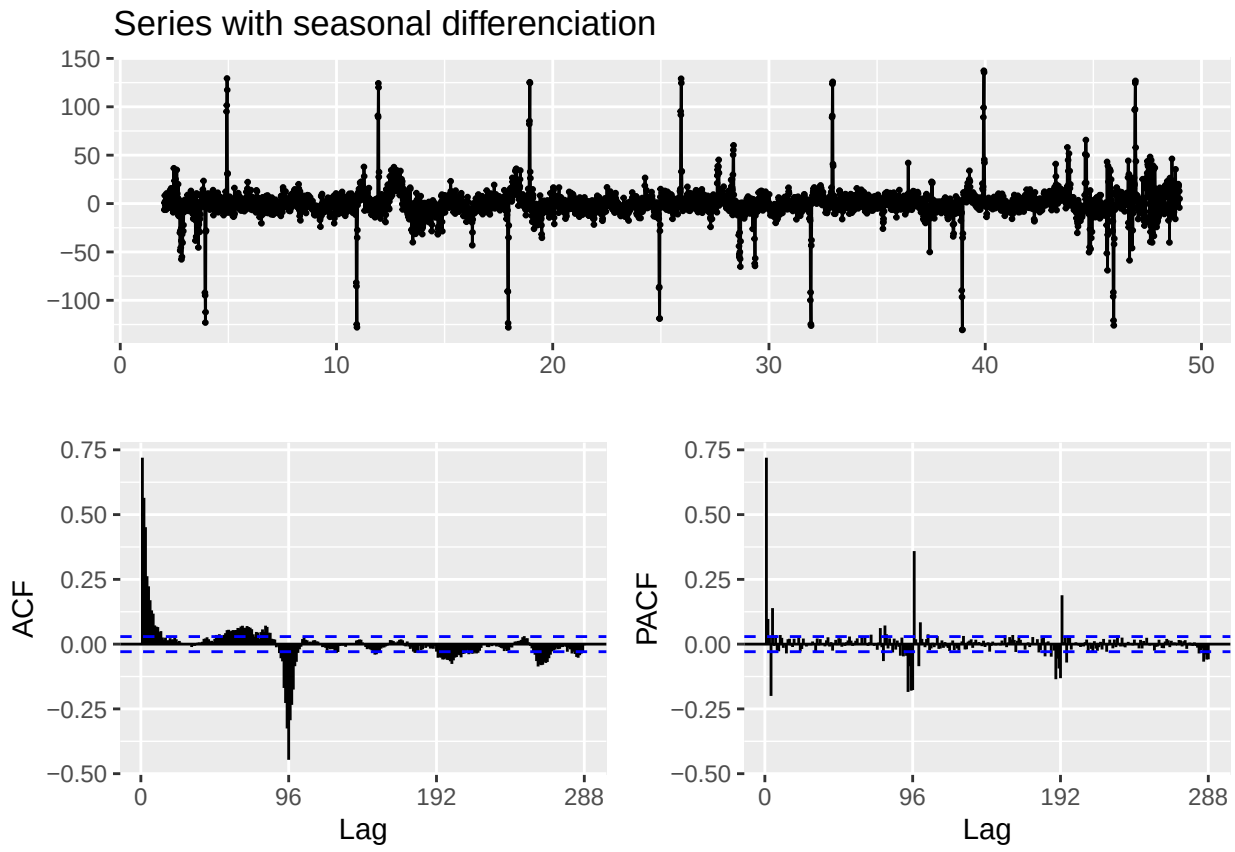
```
ggtsdisplay(elec_cleaned)
```

```
## Warning: Removed 96 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



Almost perfect sinusoide curve for the ACF and high values every 96 points in the PACF, we need to stabilize the series by differenciating its seasonal part.

```
elec_train |> diff(lag = 96) |> ggtsdisplay(main = "Series with seasonal differenciation")
```



Decreasing exponnentials on ACF associated with significant apex on lags 1,2,3,4 on PACF => AR(4) Decreasing exponentials on PACF (on the seasonal part) associated with 1 signinficant value at lag 96 on ACF => MA(1) on seasonal part.

Let's implement and benchmark our SARIMA

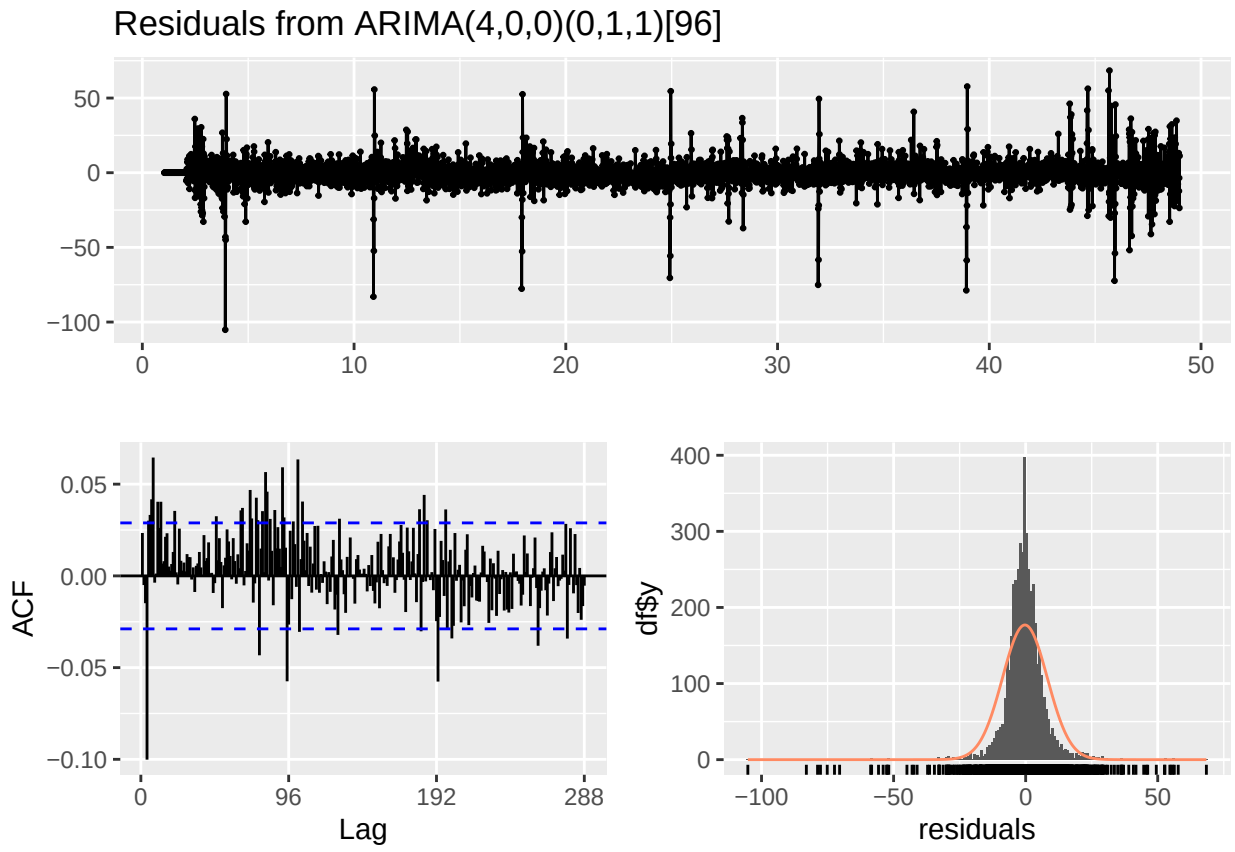
```
fit_sarima_manual <- function(train, h) {
  fit <- Arima(train,
               order = c(4, 0, 0),
               seasonal = list(order = c(0, 1, 1), period = 96),# 1 for the seasonal differenciation we
               method = "CSS") # CSS is quicker for the tests
  return(forecast(fit, h = h))
}

# CV
metrics_sarima_man <- cross_validate(elec_train, fit_sarima_manual)
model_comparison <- register_model(model_comparison, "SARIMA(4,0,0)(0,1,1)[96]", metrics_sarima_man)
```

```
print(model_comparison)
```

```
##                           Model     MAPE     RMSE
## MAPE5 SARIMA(4,0,0)(0,1,1)[96] 4.071004 14.81783
## MAPE3       HW Additive Damped 4.693817 15.54019
## MAPE1              HW Additive 4.714741 15.59412
## MAPE            Seasonal Naive 5.045776 18.07972
## MAPE4              Auto SARIMA 5.050779 18.08266
## MAPE2        HW Multiplicative 6.092145 17.37818
```

We have a slightly better leader but let's check if it has captured all the information of the series

```
final_sarima_fit <- Arima(elec_train,
                          order = c(4, 0, 0),
                          seasonal = list(order = c(0, 1, 1), period = 96),
                          method = "CSS")

checkresiduals(final_sarima_fit)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(4,0,0)(0,1,1)[96]
## Q* = 430.93, df = 187, p-value < 2.2e-16
##
## Model df: 5.   Total lags used: 192
```

We clearly don't have the white noise we wanted. There is still a high seasonal pattern that we struggle to catch. In fact, trying to increase the orders I do not succeed in capturing the pattern. We will try further on using Fourier Models as regressors to catch it.

2.4 Neural Net

```r
fit_nnar <- function(train, h) {
  # We let the function select p and P automatically based on AIC
  # We force 20 repeats to stabilize the output
  fit <- nnetar(train, repeats = 20)

  forecast(fit, h = h)
}
```

```r
# CV
metrics_nnar <- cross_validate(elec_train, fit_nnar)

# 2. Registration
model_comparison <- register_model(model_comparison, "NNAR (Auto)", metrics_nnar)

print(model_comparison)
```

```
##                           Model     MAPE     RMSE
## MAPE5 SARIMA(4,0,0)(0,1,1)[96] 4.071004 14.81783
## MAPE3       HW Additive Damped 4.693817 15.54019
## MAPE1              HW Additive 4.714741 15.59412
## MAPE6              NNAR (Auto) 5.007842 18.52538
## MAPE            Seasonal Naive 5.045776 18.07972
## MAPE4              Auto SARIMA 5.050779 18.08266
## MAPE2        HW Multiplicative 6.092145 17.37818
```

Our Neural network performs terribly. We can try and "help" him by inputing him the values we found from our SARIMA analysis :

```r
fit_nnar_optimized <- function(train, h) {
  fit <- nnetar(train, p = 4, P = 1, size = 5, repeats = 10)

  forecast(fit, h = h)
}

# CV
metrics_nnar <- cross_validate(elec_train, fit_nnar_optimized)
model_comparison <- register_model(model_comparison, "NNAR(4,1,5)[96]", metrics_nnar)
print(model_comparison)
```

```
##                           Model     MAPE     RMSE
## MAPE5 SARIMA(4,0,0)(0,1,1)[96] 4.071004 14.81783
## MAPE3       HW Additive Damped 4.693817 15.54019
## MAPE1              HW Additive 4.714741 15.59412
## MAPE6              NNAR (Auto) 5.007842 18.52538
## MAPE            Seasonal Naive 5.045776 18.07972
## MAPE4              Auto SARIMA 5.050779 18.08266
## MAPE7          NNAR(4,1,5)[96] 5.123540 17.95757
## MAPE2        HW Multiplicative 6.092145 17.37818
```

Slightly better but not much

2.5 Forecast of the 2/17/2010

Let's now train our best model i.e. *SARIMA(4,0,0)(0,1,1)[96]* over the whole dataset and predict the 2/17/2020 consumption