Time Series - second submission

Lorena Romeo

2025-01-01

Purpose of the Project

The project aims to return two forecasts, each with exactly two columns (one column per forecast) and 96 rows:

1. the first one without using outdoor temperature,
2. the second one using outdoor temperature.

Data investigation

Load all necessary libraries

```
library(readxl)
library(openxlsx)
library(tibble)
library(lubridate)
library(dplyr)
library(xts)
library(ggplot2)
library(forecast)
library(stats)
library(tseries)
library(bootUR)
library(urca)
library(zoo)
library(randomForest)
library(xgboost)
library(writexl)
library(openxlsx)
library(vars)
library(Metrics)
```

Load Data

```
data_row = read_excel("C:/Users/Utilisateur/OneDrive/Desktop/2023-11-Elec-train.xlsx")
head(data_row)
```

```
## # A tibble: 6 x 3
##   Timestamp          `Power (kW)` `Temp (C°)`
##   <chr>                     <dbl>       <dbl>
## 1 40179.052083333336         165.        10.6
## 2 1/1/2010 1:30              152.        10.6
## 3 1/1/2010 1:45              147.        10.6
## 4 1/1/2010 2:00              154.        10.6
## 5 1/1/2010 2:15              154.        10.6
## 6 1/1/2010 2:30              159         10.6
```

```
#str(data_row)
```

The dataset contains time-series data with readings of power consumption and temperature at specific time intervals. The intervals are measured every 15 minutes, from 1/1/2010 1:15 to 2/20/2010 23:45 :

- Timestamp: Represents the date and time of the recorded data
- Power (kW): The power consumption in kilowatts at the specific timestamp
- Temp (C): The temperature in Celsius at the specific timestamp

In addition, outdoor air temperature is available for 2/21/2010. The goal is to forecast electricity consumption (kW) for 2/21/2010.

Data cleansing

I will effectuate some manipulation to clean and prepare the data for modelling.

Timestamp handling

During the load a Timestamp was not handled properly in R. In order to have full control over how each type of timestamp is handled it is appropriate to create a custom function.

```
convert_timestamp = function(ts) {
  if (grepl("^[0-9]+\\.[0-9]+$", ts)) {
       excel_date = as.numeric(ts)
    origin = as.Date("1899-12-30")
    date = origin + excel_date
    return(as.POSIXct(date))
  } else {
    return(parse_date_time(ts, orders = "m/d/y H:M", tz = "UTC"))
  }
}

data_row$Timestamp = sapply(data_row$Timestamp, convert_timestamp)

data_row$Timestamp = as.POSIXct(data_row$Timestamp, origin = "1970-01-01", tz = "UTC")
```

We check the results of the dataset after this Timestamp transformation and we can see that now the Timestamp column is handled properly.

```
# head(data_row)
```

Prepare the dataset

I rename the variables to be more user-friendly

```
#complete dataset
elec <- data_row %>%
  rename(
    Power_kW = `Power (kW)`,
    Temp_C = `Temp (C°)`
  )

#head(elec)
```

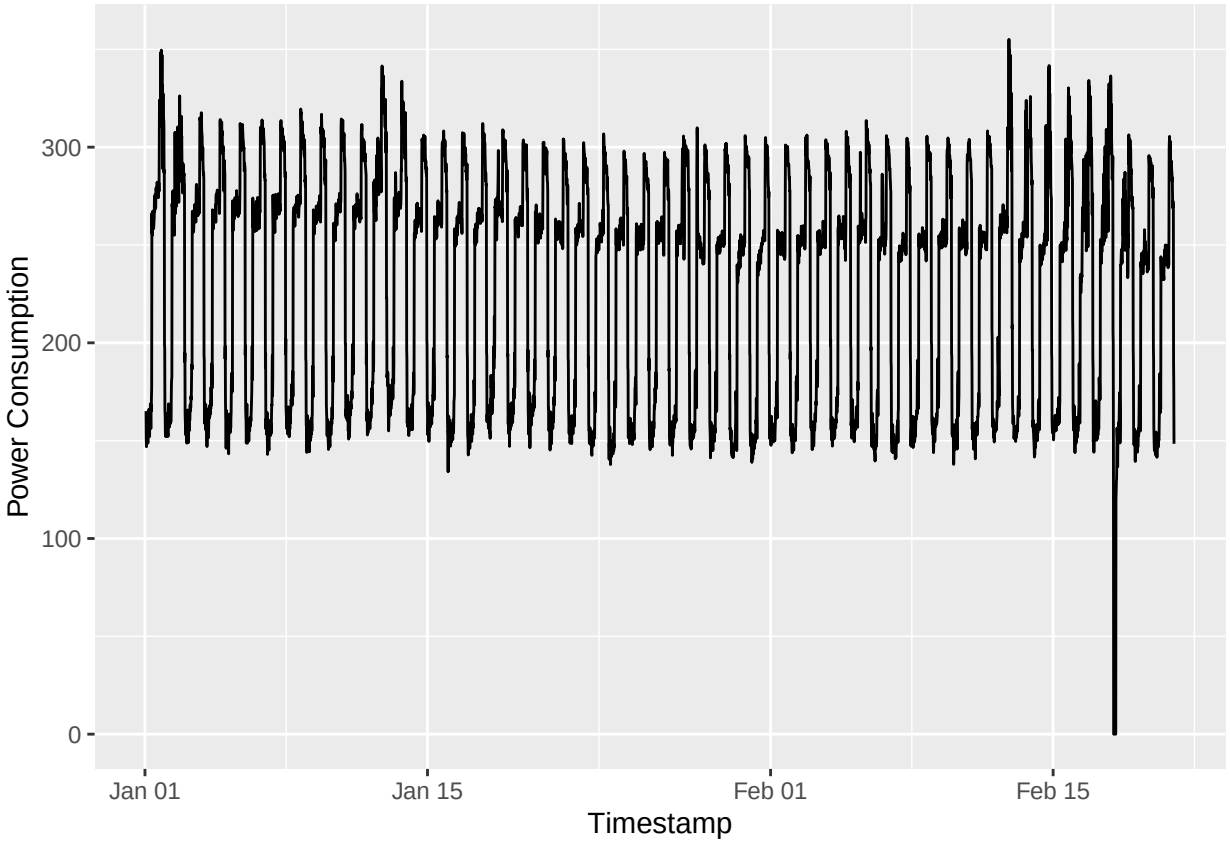
I remove the rows of the day that I want to forecast, since it can interfere with the good functioning of the code.

```
elec <- elec %>%
  filter(!is.na(Power_kW))
```

Plot the data

Now I want to plot the data to have a first view of it

```
ggplot(elec, aes(x = Timestamp, y = Power_kW)) +
geom_line() +
labs( x = "Timestamp", y = "Power Consumption")
```



From the plot I can see that there is a low peak (outlier) on day 18 February. Let's investigate.

```
outlier <- elec %>%
  filter(Power_kW == 0)

print(outlier)
```

```
## # A tibble: 11 x 3
##    Timestamp           Power_kW Temp_C
##    <dttm>                 <dbl>  <dbl>
##  1 2010-02-18 00:00:00        0   12.8
##  2 2010-02-18 00:15:00        0   12.2
##  3 2010-02-18 00:30:00        0   12.2
##  4 2010-02-18 00:45:00        0   12.2
##  5 2010-02-18 01:00:00        0   12.2
##  6 2010-02-18 01:15:00        0   11.7
##  7 2010-02-18 01:30:00        0   11.7
##  8 2010-02-18 01:45:00        0   11.7
##  9 2010-02-18 02:00:00        0   11.7
## 10 2010-02-18 02:15:00        0   11.1
## 11 2010-02-18 02:30:00        0   11.1
```

It is visible that for 2 hours and 15 minutes the Power consumption was at 0. This could be due to a power outage happened during this period.

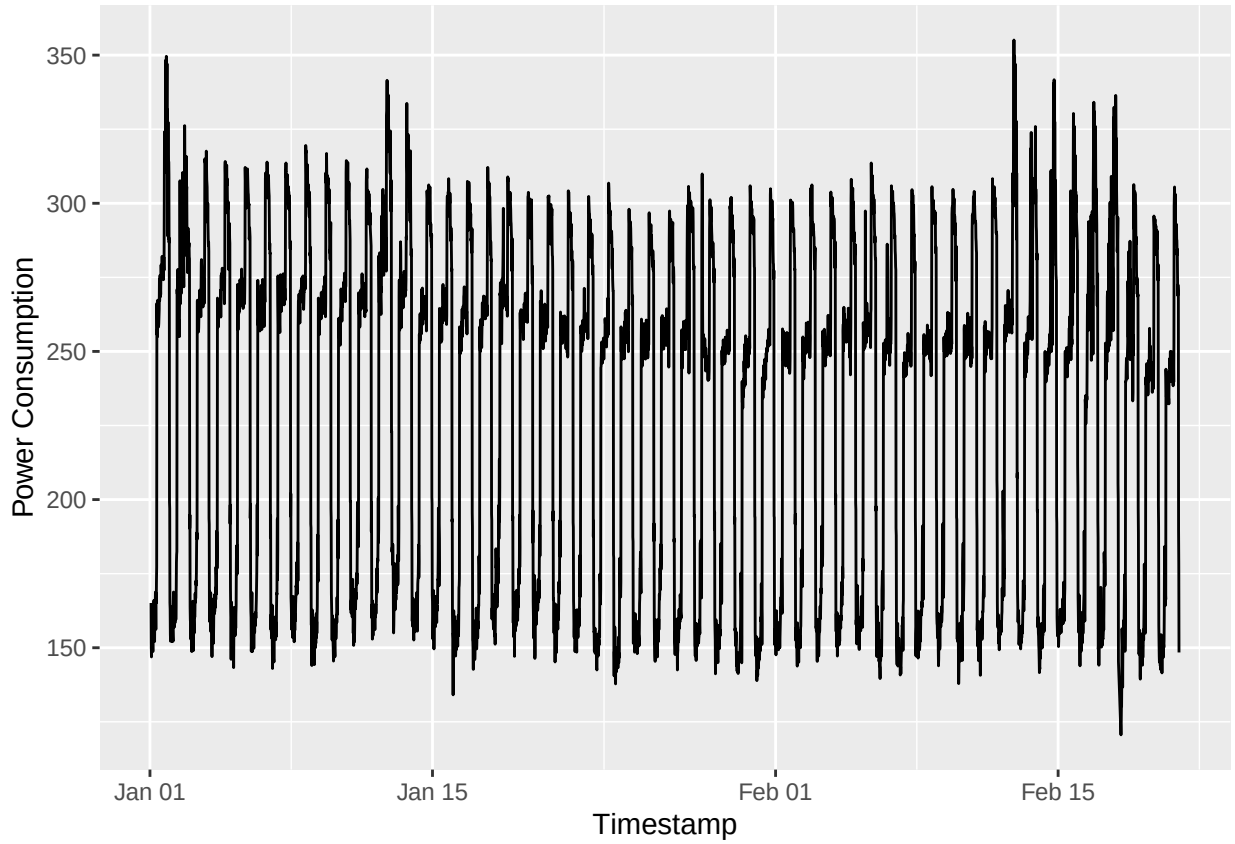
This could impact the forecast, therefore it would be better to replace the zero value with NA. I decided to treat zero values as missing data since they do not represent actual consumption.

```r
#interpolation
elec <- elec %>%
  mutate(
    Power_kW = na.approx(ifelse(Power_kW == 0, NA, Power_kW))
  )
```

Plot adjusted data

The plot with the imputation of missing values using interpolation changed the 0 values to NA. Now, the plot no longer shows the outliers because they have been replaced with interpolated values. The data is now clean.

```r
ggplot(elec, aes(x = Timestamp, y = Power_kW)) +
geom_line() +
labs( x = "Timestamp", y = "Power Consumption")
```



Now that the data is clean I can transform the data in ts object and investigate the seasonality.

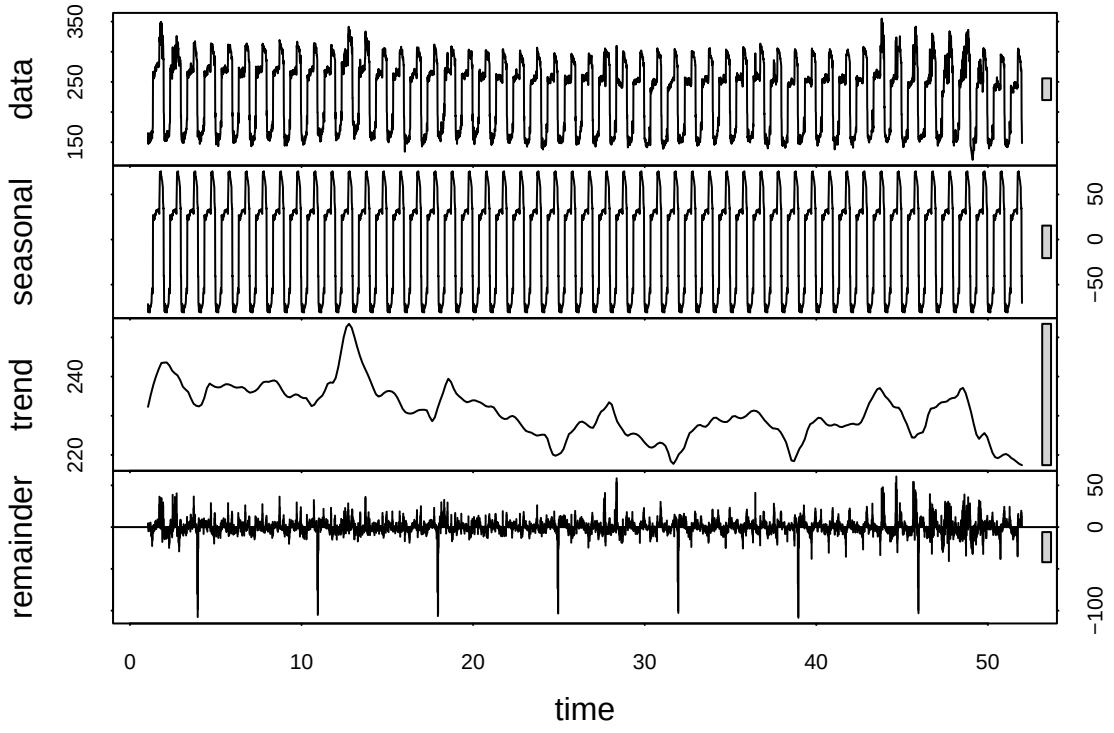
Create Time series object

```r
#complete data until 20 Feb
elec_ts = ts(elec$Power_kW, start = c(1,6), frequency = 96)
```

Time series Investigation

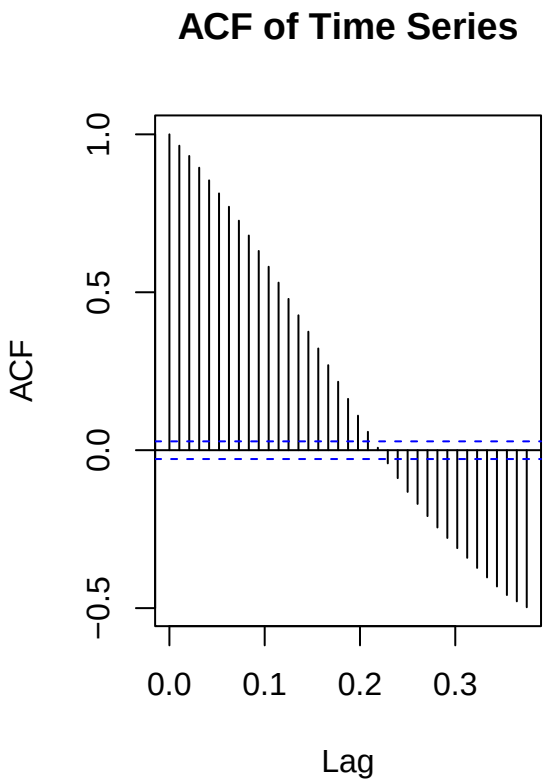
I decompose the time series to investigate seasonality and trends.

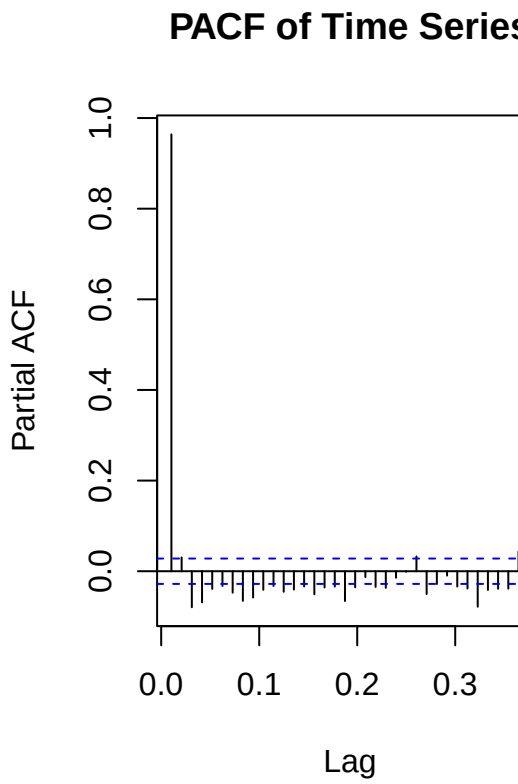
```
decomposed <- stl(elec_ts, s.window = "periodic")

# Plot the decomposed components
plot(decomposed)
```



The seasonal plot shows fixed frequency in the series pattern. The trend does not show any particular pattern or maybe a little decreasing pattern. The residuals shows correlation between them because we can see almost regular low spikes so it is not white noise.

```
par(mfrow = c(1, 2))
# Plot ACF with a large number of lags
#the plot shows cosine wave, this indicates a seasonal component in the time series
acf(elec_ts, main = "ACF of Time Series")

# Plot PACF with a large number of lags
# the spike on the lag of 96 shows daily seasonality (lag = 1 = 96)
pacf(elec_ts, main = "PACF of Time Series")
```



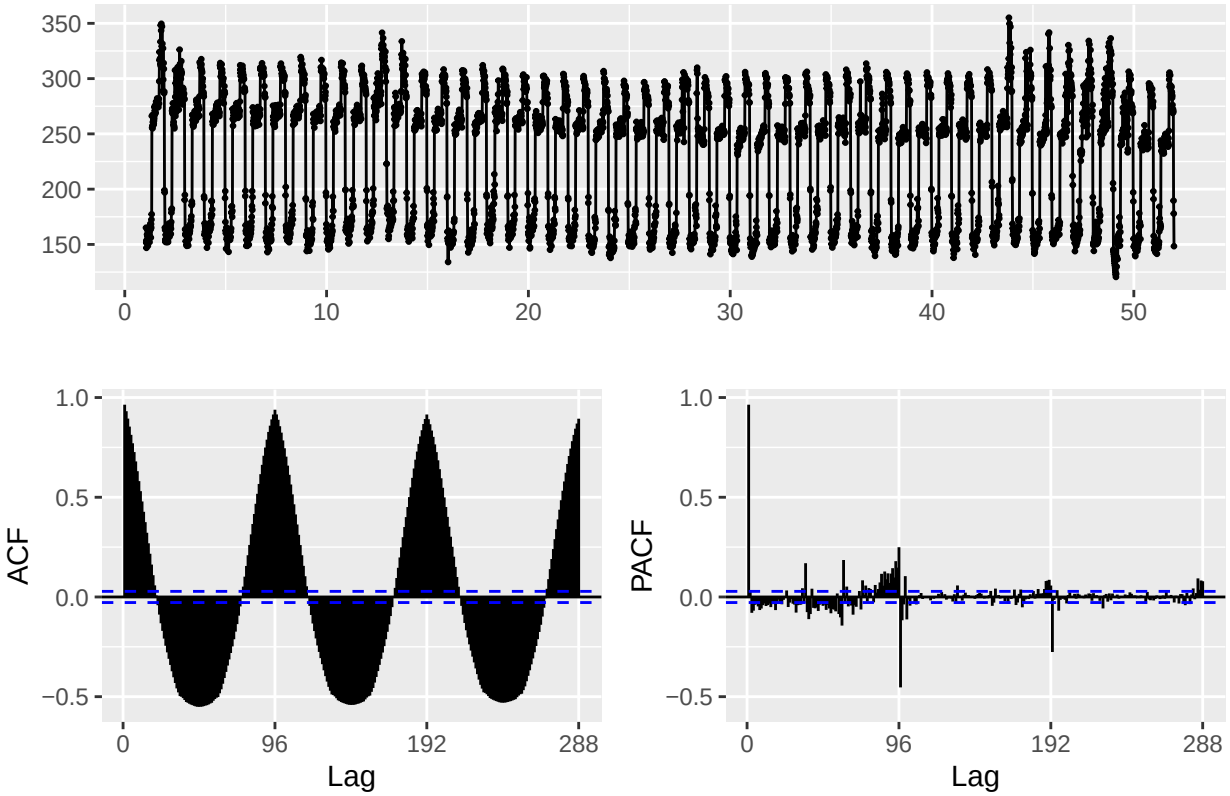


Plot ACF and PACF

The AFC plot shows cosine wave, this indicates a seasonal component in the time series In the PAFC plot the spike on the lag of 96 shows daily seasonality (lag = 1 = 96)

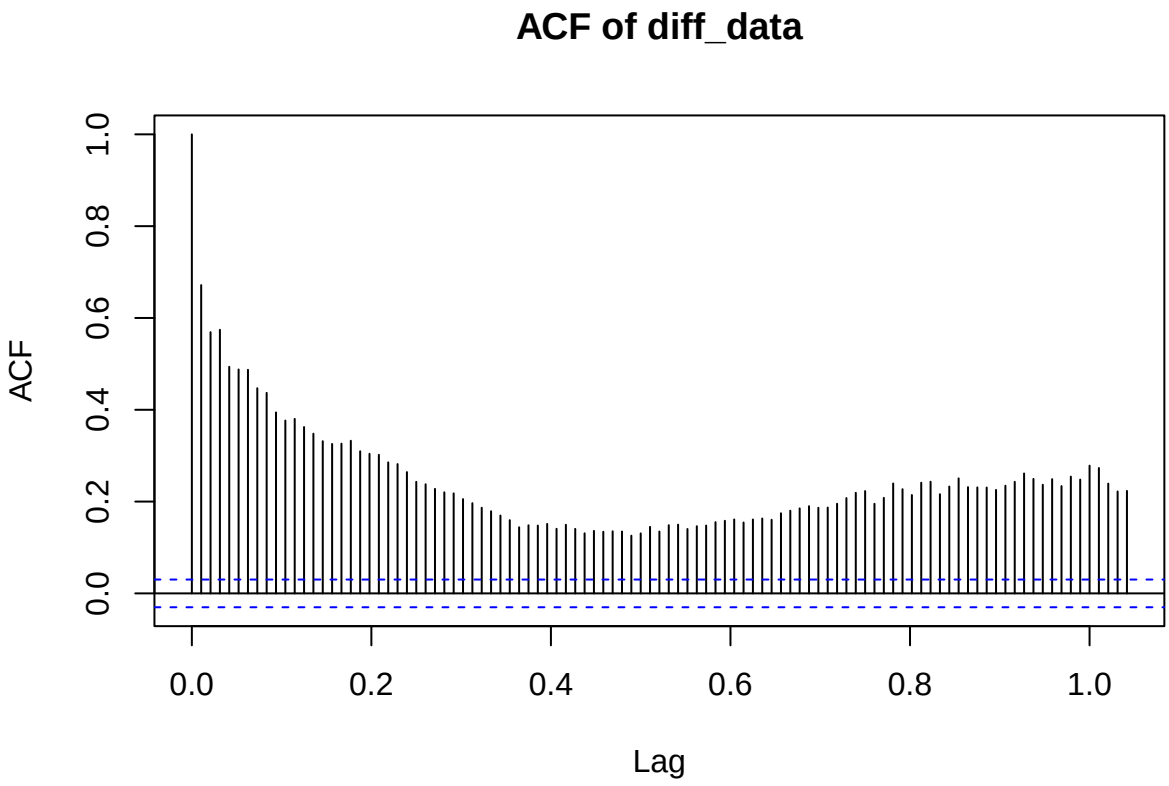
With the ggtsdisplay we can see very cleary the cosine wave, evidence of seasonality.

```
ggtsdisplay(elec_ts)
```

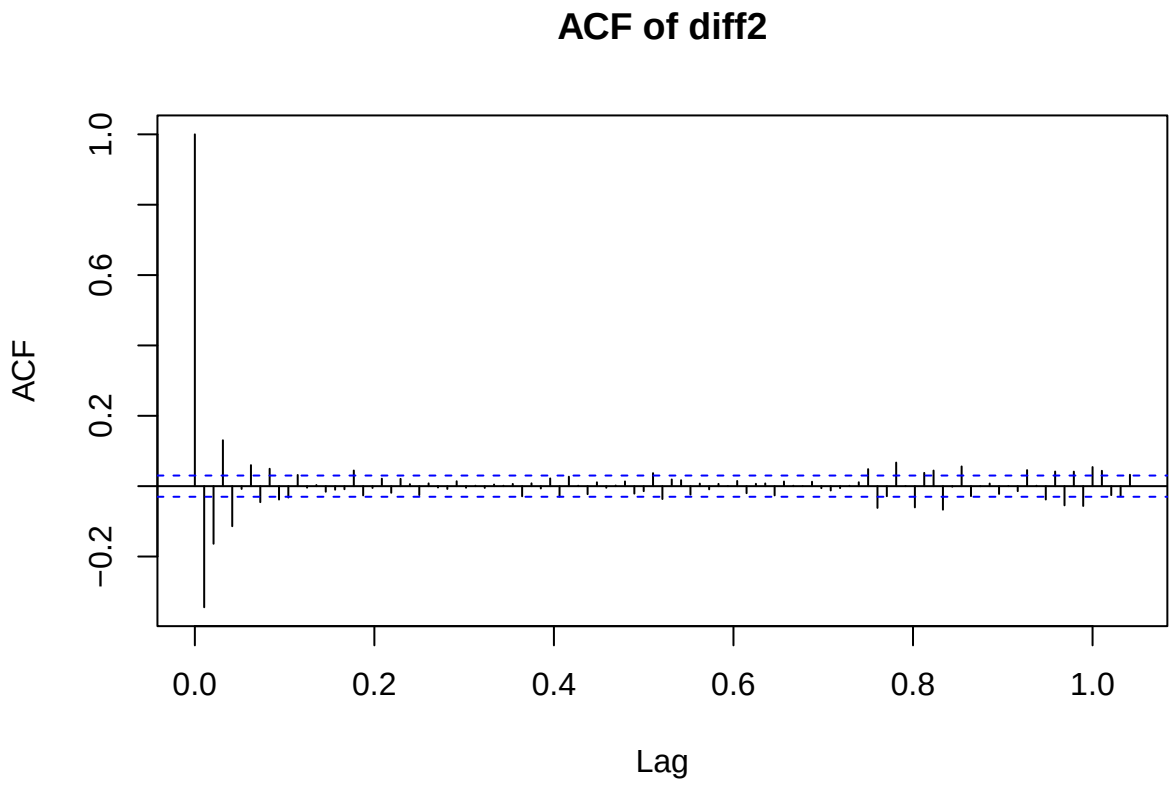


I check hiw many times I need to differentiate the data to remove the non-stationary pattern. The lag applied is 96*7 because I saw that the cycle was of 96 period repeated every 7 day.

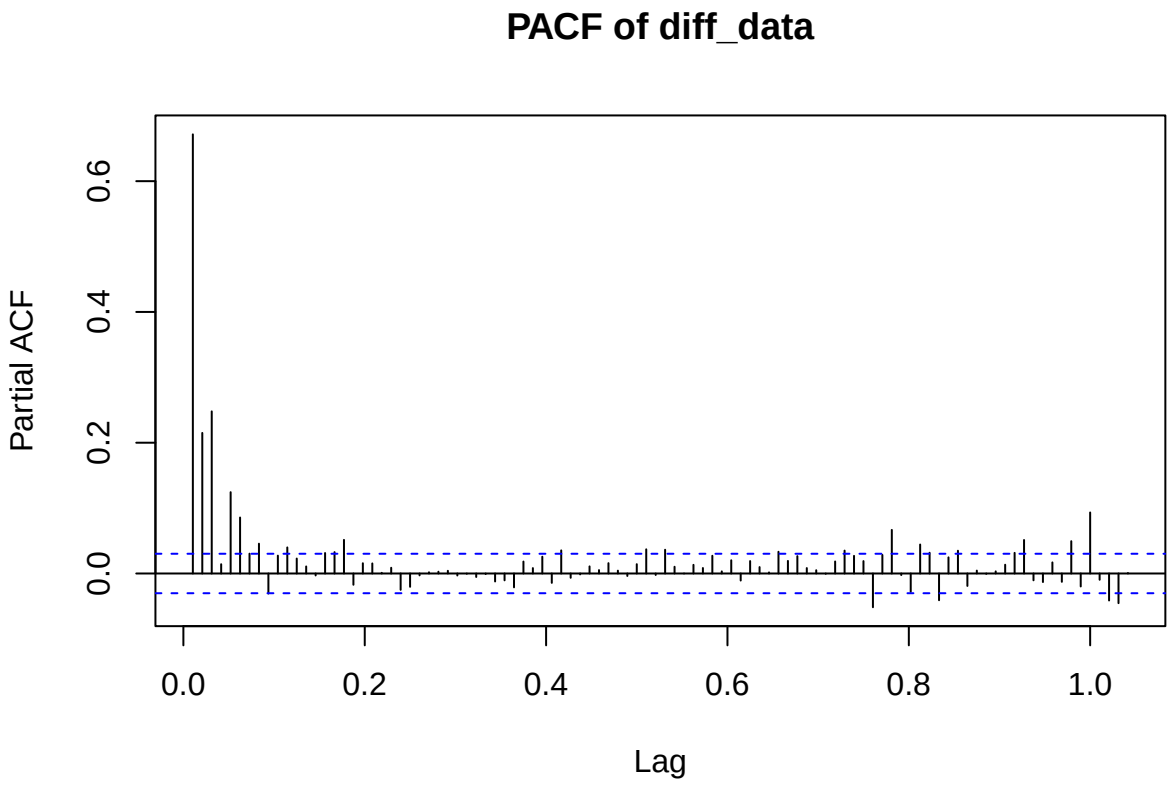
```
lag = 96 * 7
diff_data = diff(elec_ts, lag = lag)

diff2 = diff(diff_data)

#diff3 = diff(diff2)

acf(diff_data, main = "ACF of diff_data", lag.max = 100)
```



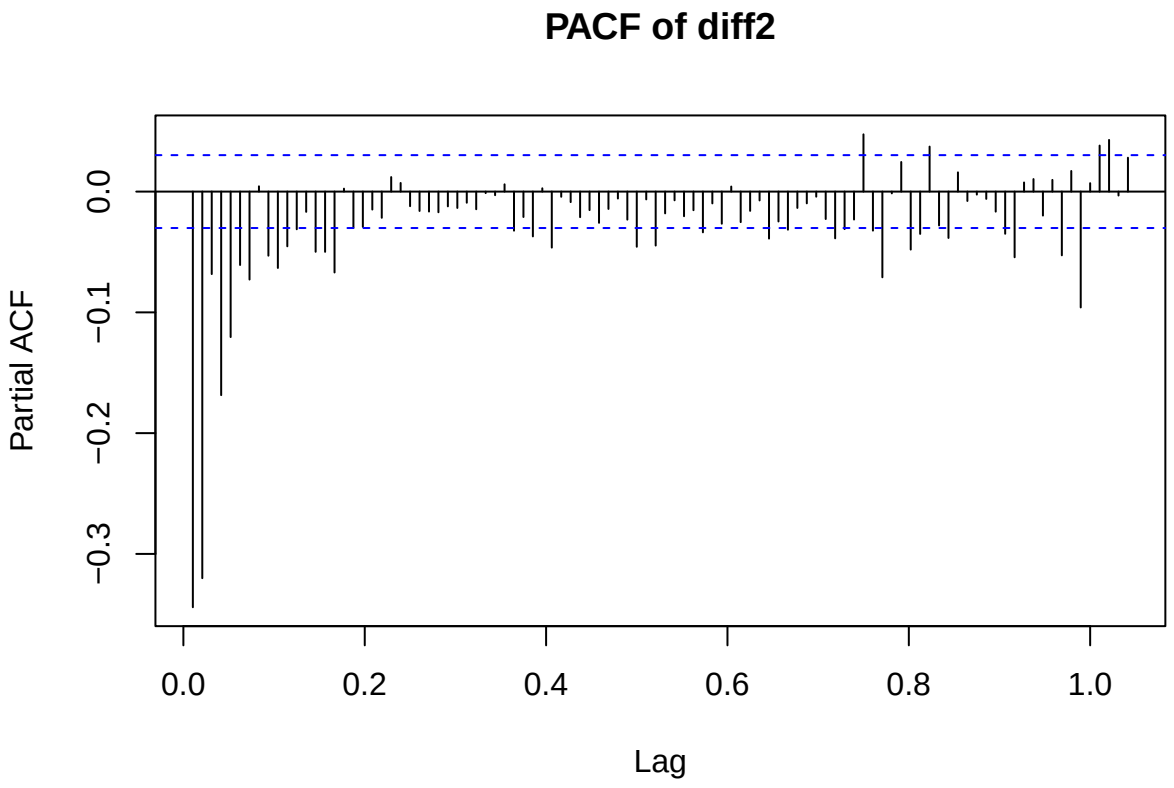

```r
acf(diff2, main = "ACF of diff2", lag.max = 100)
```



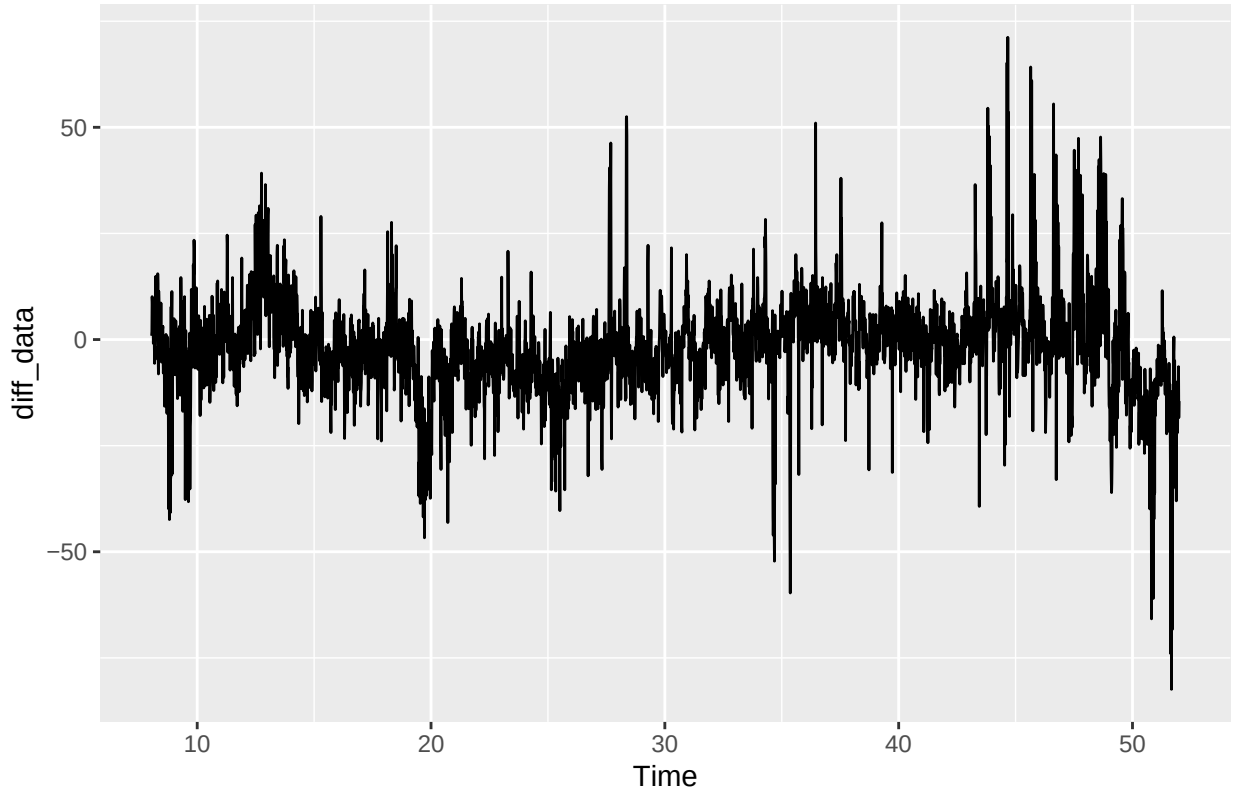

```
#acf(diff3, main = "ACF of diff3", lag.max = 100)

# Plot PACF
pacf(diff_data, main = "PACF of diff_data", lag.max = 100)
```



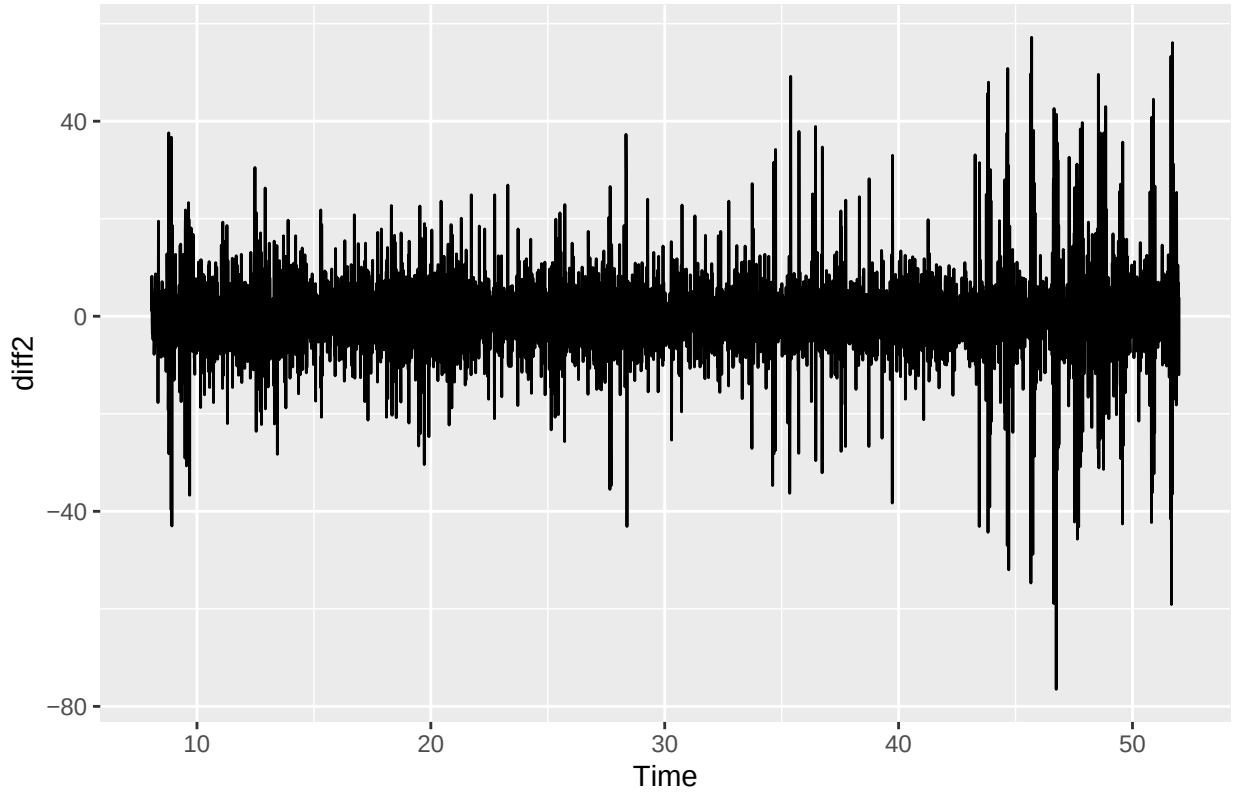

```
pacf(diff2, main = "PACF of diff2", lag.max = 100)
```




```
#pacf(diff3, main = "PACF of diff3", lag.max = 100)

autoplot(diff_data)
```

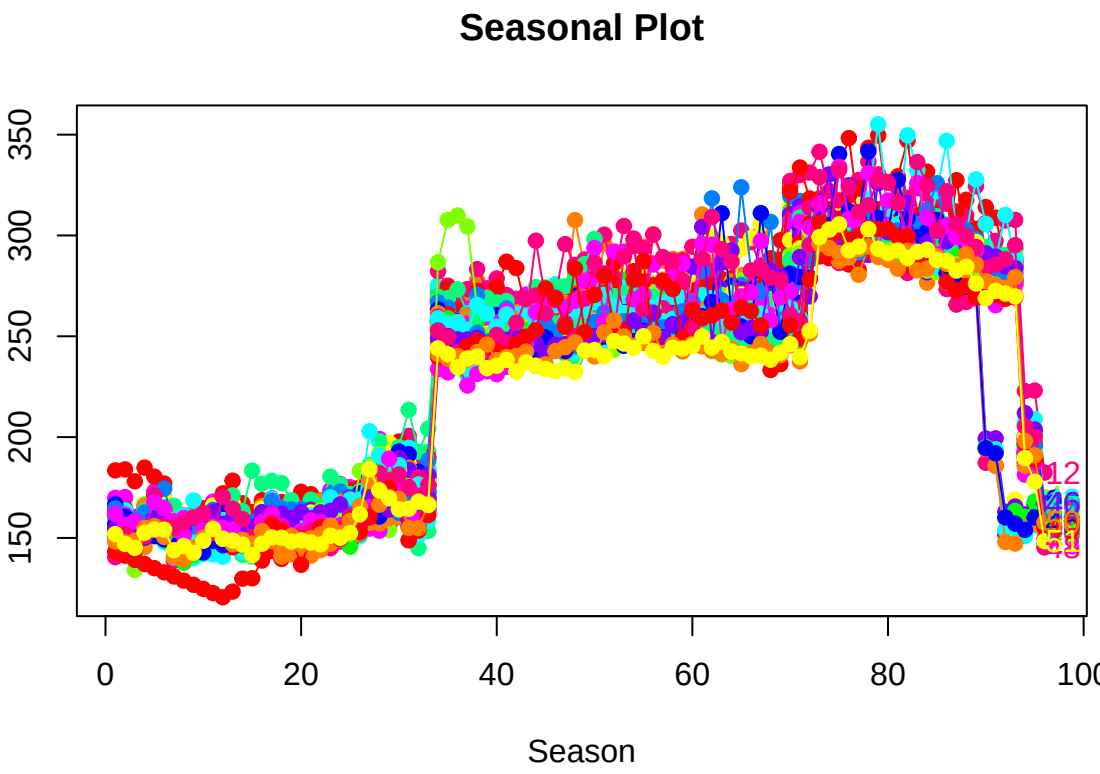



```
autoplot(diff2)
```



From the above I can see that to remove stationary we need to differentiate 1 or 2 times. The 3rd time is not necessary.

```
# Plot seasonal component
seasonplot(elec_ts, main = "Seasonal Plot", col = rainbow(12), year.labels = TRUE, pch = 19)
```



Check seasonality

From the above plot we can clearly see the seasonality. We can see that the most of the power consumption is during the central hours of the day and decrease in the night and the early morning.

Modelization

After all the above investigation we can start modelling. We split the dataset in training and test set.

```
# Set up the start and end dates fro train and test dataset
train_start = as.POSIXct("2010-01-01 01:15:00", tz = "UTC")
train_end = as.POSIXct("2010-02-18 23:45:00", tz = "UTC")
test_start = as.POSIXct("2010-02-19 00:00:00",  tz = "UTC")
test_end = as.POSIXct("2010-02-20 23:45:00",  tz = "UTC")

# Create training and test datasets
train_data = elec %>% filter(Timestamp >= train_start & Timestamp <= train_end)
test_data = elec %>% filter(Timestamp >= test_start & Timestamp <= test_end)


#check the boundaries of the datasets
min(train_data$Timestamp)
```

```
## [1] "2010-01-01 01:15:00 UTC"
```

```r
max(train_data$Timestamp)
```

```
## [1] "2010-02-18 23:45:00 UTC"
```

```r
min(test_data$Timestamp)
```

```
## [1] "2010-02-19 UTC"
```

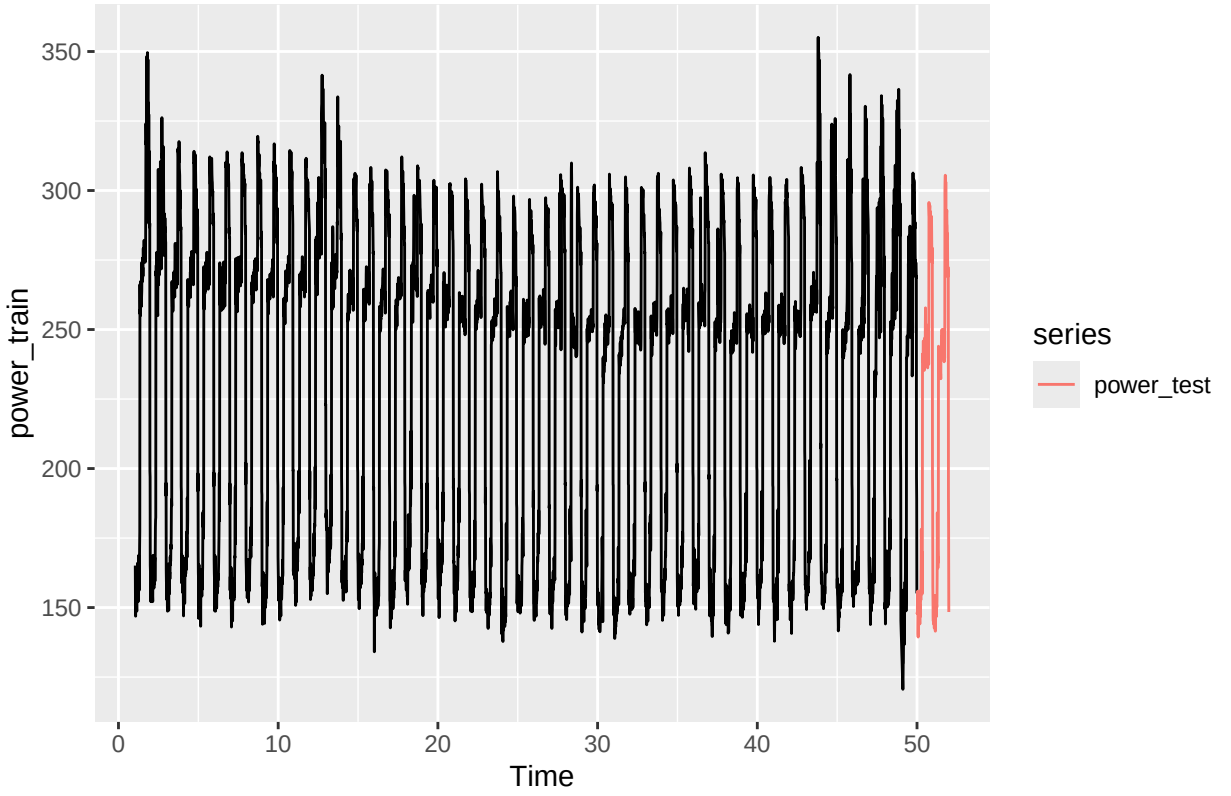
```r
max(test_data$Timestamp)
```

```
## [1] "2010-02-20 23:45:00 UTC"
```

Forecats Power consumption without using Temperature

Create the ts objects to work to forecast the Power without using the Temperature feature

```r
# Create time series objects (without temperature)
frequency = 96  # 96 intervals per day (15-minute intervals)
power_train = ts(train_data$Power_kW, c(1,6), end = c(49, 96), frequency = frequency)
#I do not put the end in the test part because the boundary is already set in the code above
power_test = ts(test_data$Power_kW, start = c(50, 1),  frequency = frequency)

#plot the data, we can see a total of 51 periods. 1 period represent 1 day, with 96 observations per da
autoplot(power_train)+
  autolayer(power_test)
```



Remark: As mentioned above, I tried 3 levels of differentiation and tested the differentiated data several times. The 3rd differentiation is not working properly, the one with best results is one or two differentiation. After 1 differentiation the data looks stationary. I do not use differentiated data in my train and test dataset because when I need to come back to the original values I have a wrong conversion even using "cumsum" and "log" dataset so I removed this part.

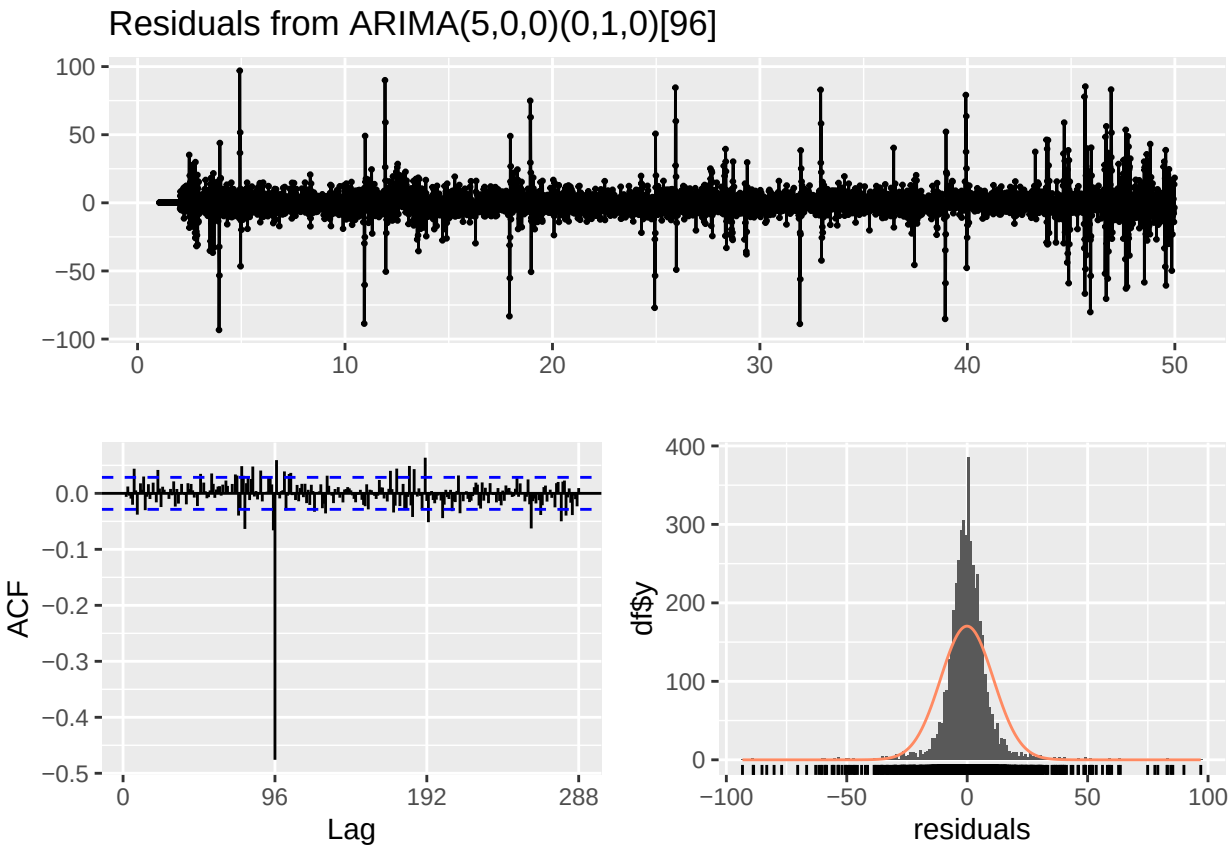
Model: SARIMA

Auto Arima

```
fit_auto_arima = auto.arima(power_train, seasonal = TRUE)
summary(fit_auto_arima)
```

```
## Series: power_train
## ARIMA(5,0,0)(0,1,0)[96]
##
## Coefficients:
##          ar1     ar2     ar3      ar4     ar5
##       0.6722  0.0700  0.1581  -0.2871  0.1360
## s.e.  0.0146  0.0171  0.0170   0.0171  0.0146
##
## sigma^2 = 122.8:  log likelihood = -17601.13
## AIC=35214.25   AICc=35214.27   BIC=35252.86
##
## Training set error measures:
##                       ME     RMSE     MAE        MPE     MAPE      MASE
## Training set -0.09196146 10.96244 6.43308 -0.1532801 2.906827 0.7371052
##                      ACF1
## Training set 0.0006284281
```

```
#check residuals
checkresiduals(fit_auto_arima)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(5,0,0)(0,1,0)[96]
## Q* = 1487.5, df = 187, p-value < 2.2e-16
##
## Model df: 5.   Total lags used: 192
```

With the Auto arima ARIMA(5,0,0)(0,1,0)[96] we can see that from the residuals plot regular spikes showing that is not a white noise and from the ACF plot a big spike at lag 96 showing still auto correlations.

Manually tuned Sarima

In the process of manually tuning the SARIMA model, several configurations were tested to identify the best-performing model. Here are the observations:

SARIMA(2,1,2)(1,1,1): This configuration did not perform well, possibly due to inadequate parameter settings for capturing the data's seasonality and trend.

SARIMA(4,1,2)(1,1,1): This model showed acceptable performance, indicating a better fit compared to the previous configuration.

SARIMA(7,1,2)(1,1,1): The model was too complex, leading to computational issues.

SARIMA(0,1,2)(2,1,1): This model performed adequately but was not as effective as the SARIMA(4,1,2)(1,1,1) configuration.

SARIMA(0,1,5)(1,1,1): This configuration showed reasonable performance, suggesting it could be a viable option.

SARIMA(0,1,6)(1,1,1): This model was the second-best performer, indicating a good balance between complexity and fit.

SARIMA(0,1,7)(1,1,1): This configuration was identified as the best-performing model, providing the most accurate fit for the data.

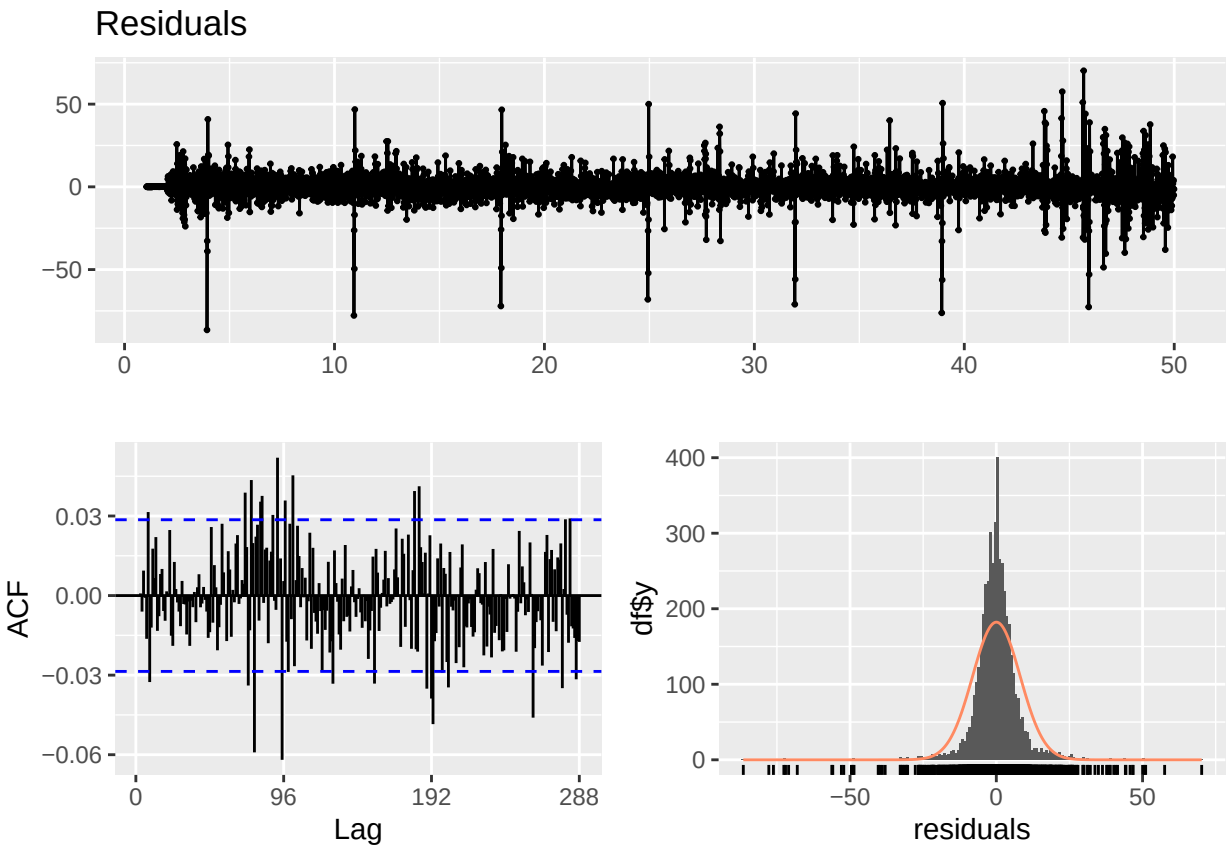
```
fit_manual_sarima <- Arima(
  power_train,
  order = c(1, 1, 7),
  seasonal = list(order = c(1, 1, 1))
)
```

The SARIMA(0,1,7)(1,1,1) model emerged as the best choice, offering the highest accuracy and best fit among the tested configurations. The SARIMA(0,1,6)(1,1,1) model was a close second, while other configurations showed varying degrees of performance, with some being too complex or less effective.

```
# Extract residuals from the SARIMA model
summary(fit_manual_sarima)
```

```
## Series: power_train
## ARIMA(1,1,7)(1,1,1)[96]
##
## Coefficients:
##          ar1      ma1     ma2     ma3      ma4     ma5     ma6      ma7
##       0.8452  -1.1834  0.1383  0.1657  -0.3163  0.1946  0.0153  -0.0090
## s.e.  0.0424   0.0450  0.0273  0.0234   0.0223  0.0253  0.0240   0.0176
##          sar1     sma1
##       -0.0494  -0.8673
## s.e.   0.0172   0.0093
##
## sigma^2 = 67.81:  log likelihood = -16300.26
## AIC=32622.52   AICc=32622.58   BIC=32693.29
##
## Training set error measures:
##                       ME     RMSE      MAE         MPE    MAPE      MASE
## Training set 0.02606993 8.140314 4.984531 -0.08805075 2.25385 0.5711298
##                       ACF1
## Training set -0.0001462164
```

```
residuals_man_sarima <- residuals(fit_manual_sarima)
checkresiduals(residuals_man_sarima)
```




```
##
##  Ljung-Box test
##
## data:  Residuals
## Q* = 296.68, df = 192, p-value = 1.834e-06
##
## Model df: 0.   Total lags used: 192
```

```
AIC(fit_manual_sarima)
```

```
## [1] 32622.52
```

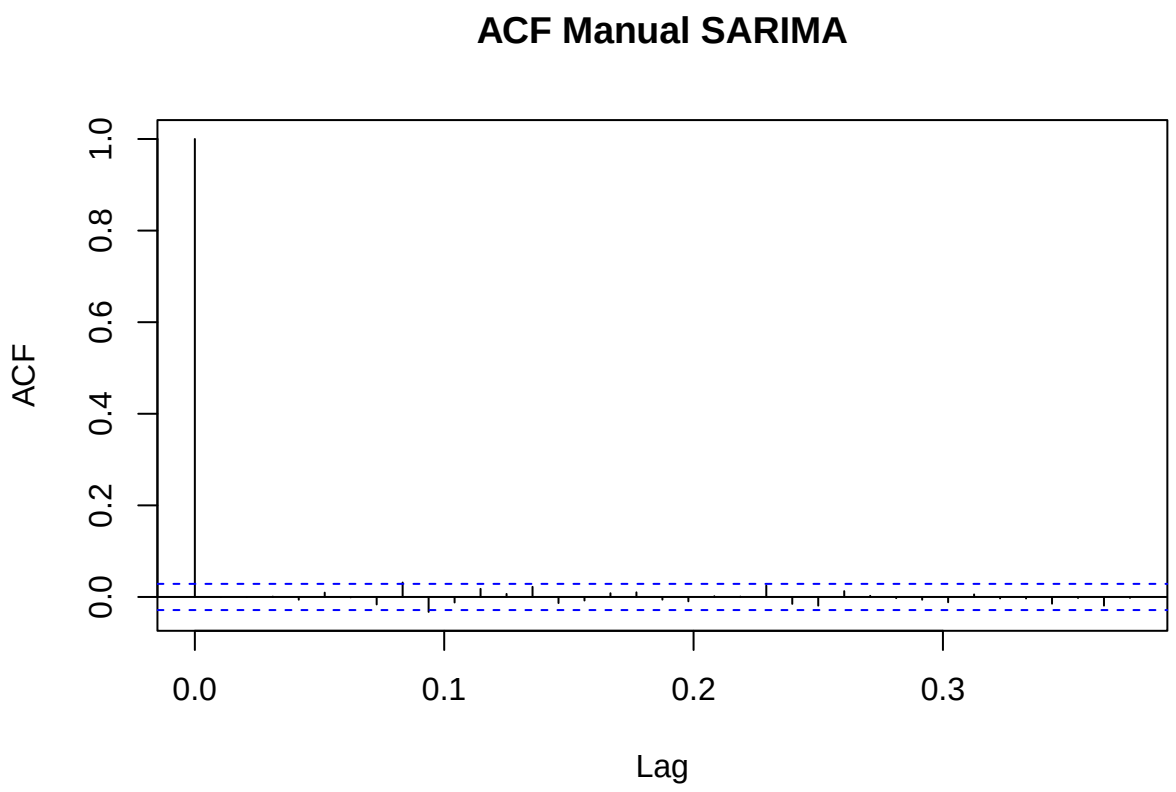
```
BIC(fit_manual_sarima)
```

```
## [1] 32693.29
```

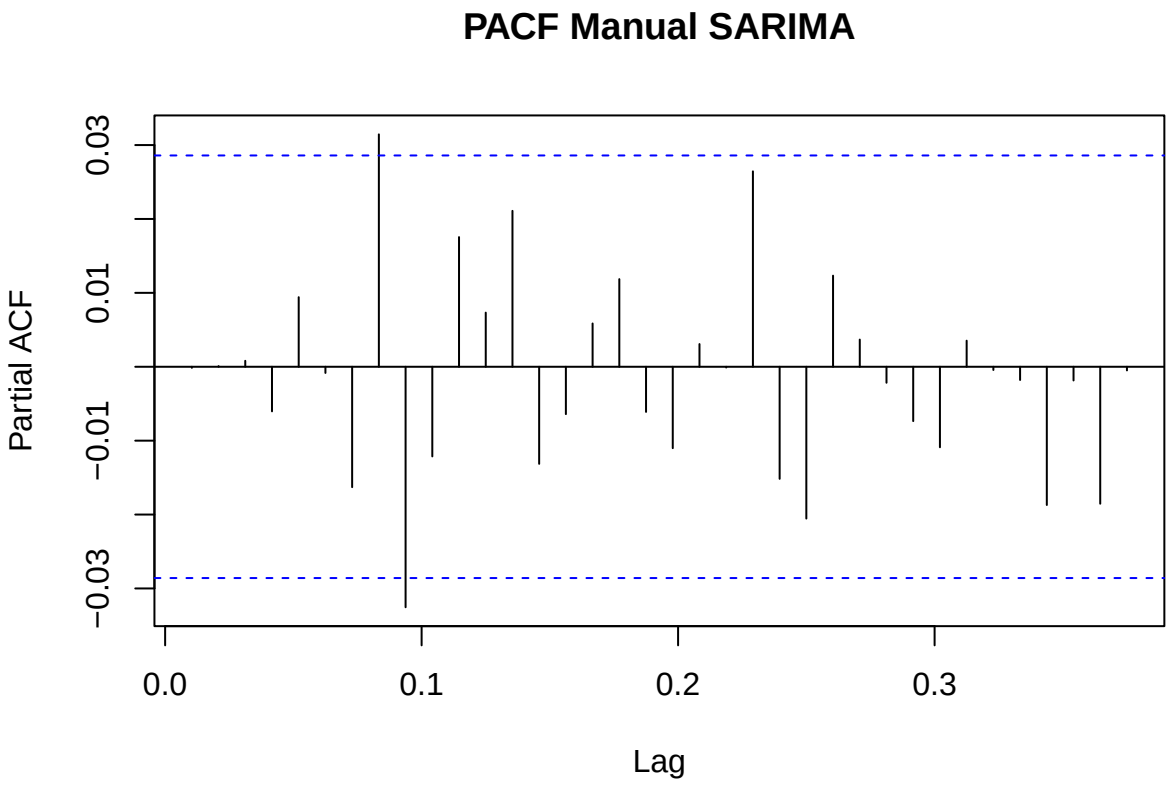
```
rmse_man_sarima <- sqrt(mean(residuals_man_sarima^2))
print(rmse_man_sarima)
```

```
## [1] 8.140314
```

```r
# Plot PACF for the residuals
acf(residuals_man_sarima, main = "ACF Manual SARIMA")
```




```r
pacf(residuals_man_sarima, main = "PACF Manual SARIMA")
```

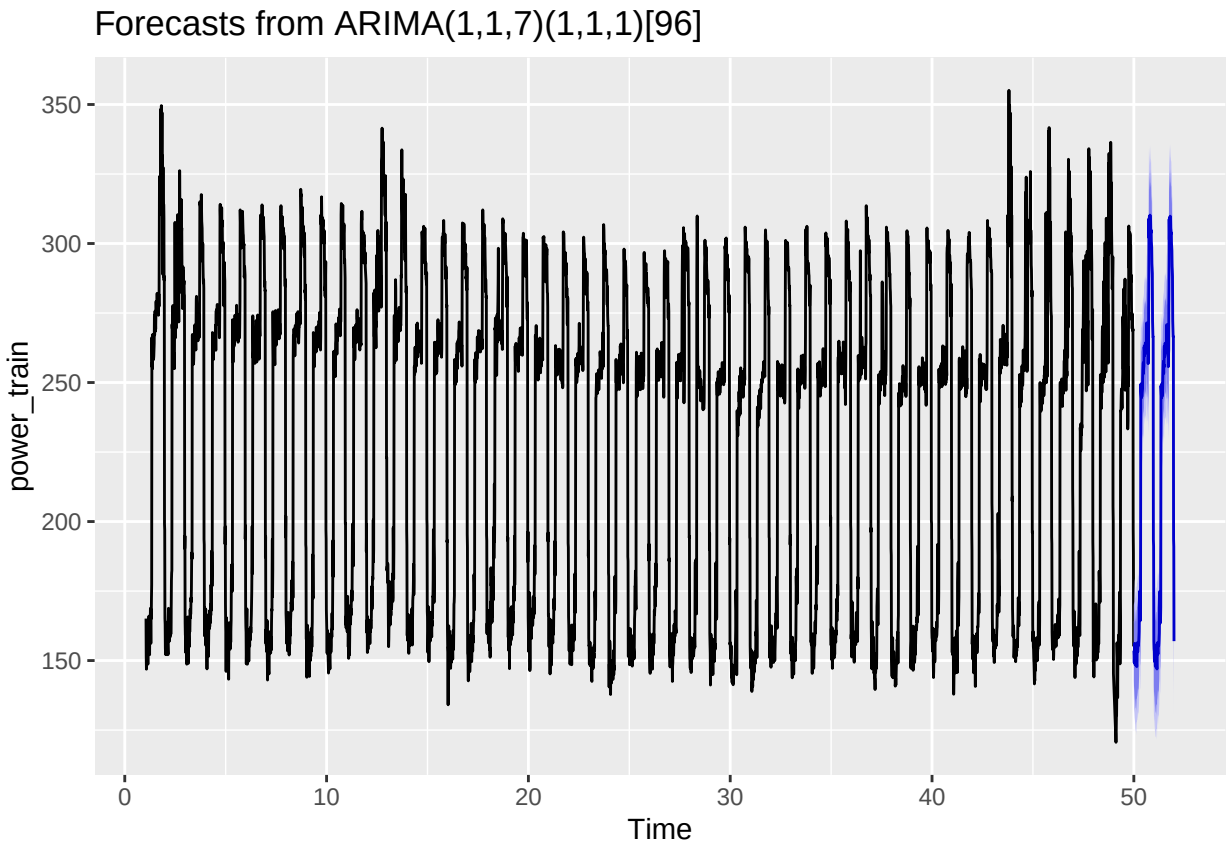


Forecast with Sarima models comparing with test set.

With the fine tuned sarima the RMSE is lower same as the AIC and BIC

```
prev_auto_sarima=forecast(fit_auto_arima,h=length(power_test))
#autoplot(prev_auto_sarima)
```

```
prev_man_sarima=forecast(fit_manual_sarima,h=length(power_test))
autoplot(prev_man_sarima)
```



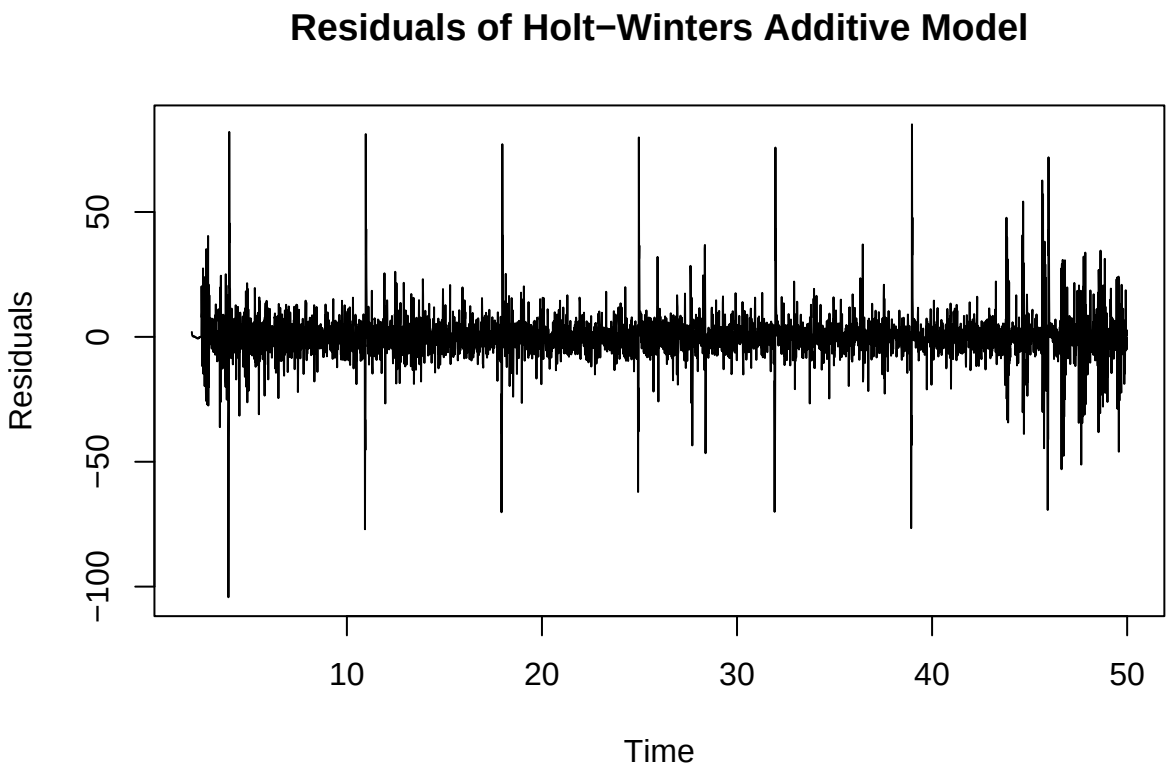
The above forecasts will be compared with the other models I will create below.

Model: Exponential Smoothing

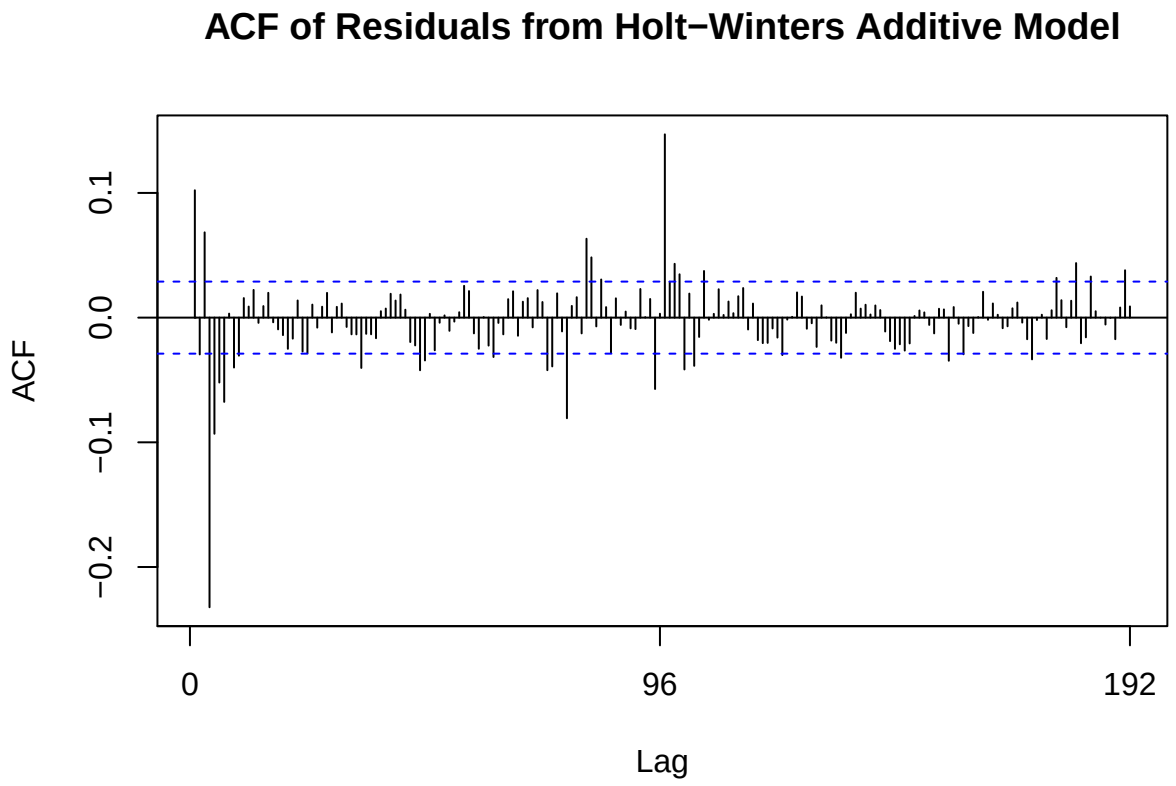
Holt Winters Additive

I create an Holt-Winters additive model since it's a reliable method for forecasting time series data that exhibits seasonality

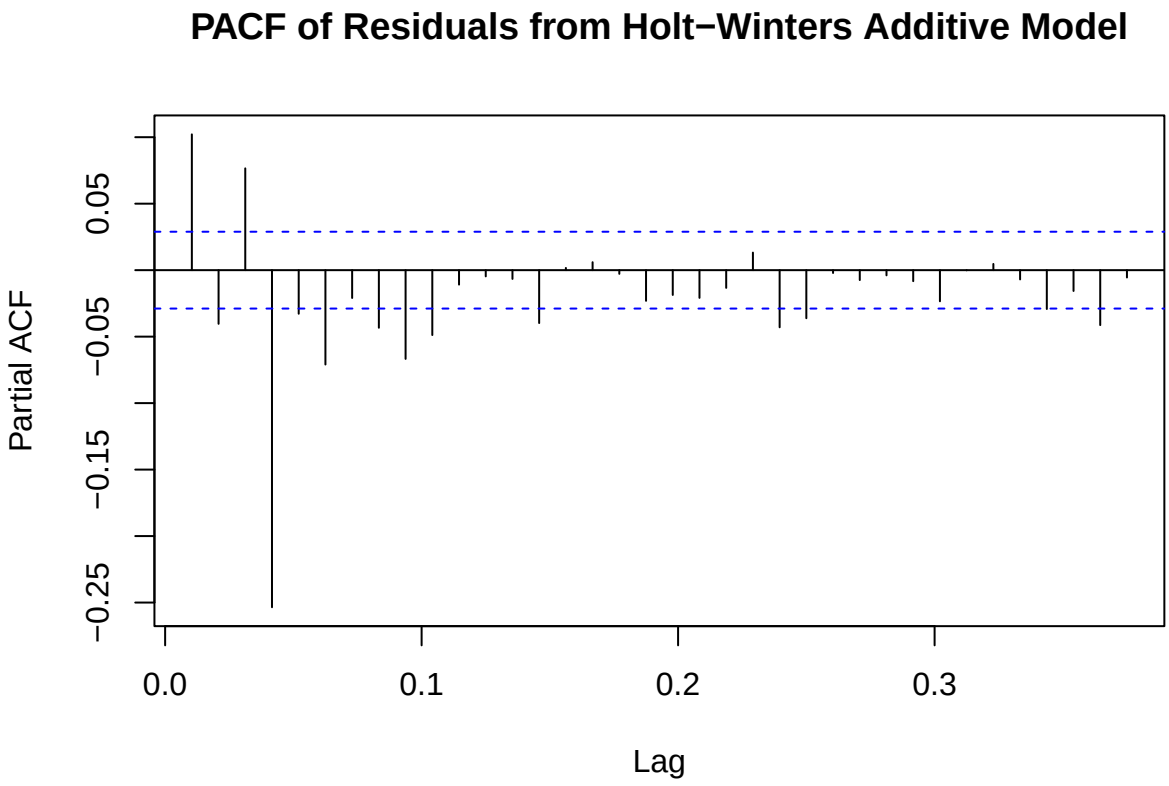
```
hw_Power_add = HoltWinters(power_train, seasonal = "additive")

residuals_hw = residuals(hw_Power_add)
#the residuals still have some spikes
#plot residuals, the plot does not show white noise, there is still some spikes visible
plot(residuals_hw, main = "Residuals of Holt-Winters Additive Model", xlab = "Time", ylab = "Residuals"
```




```r
residuals_hw <- residuals(hw_Power_add)
#summary(residuals_hw)
Acf(residuals_hw, main = "ACF of Residuals from Holt-Winters Additive Model")
```




```
pacf(residuals_hw, main = "PACF of Residuals from Holt-Winters Additive Model")
```



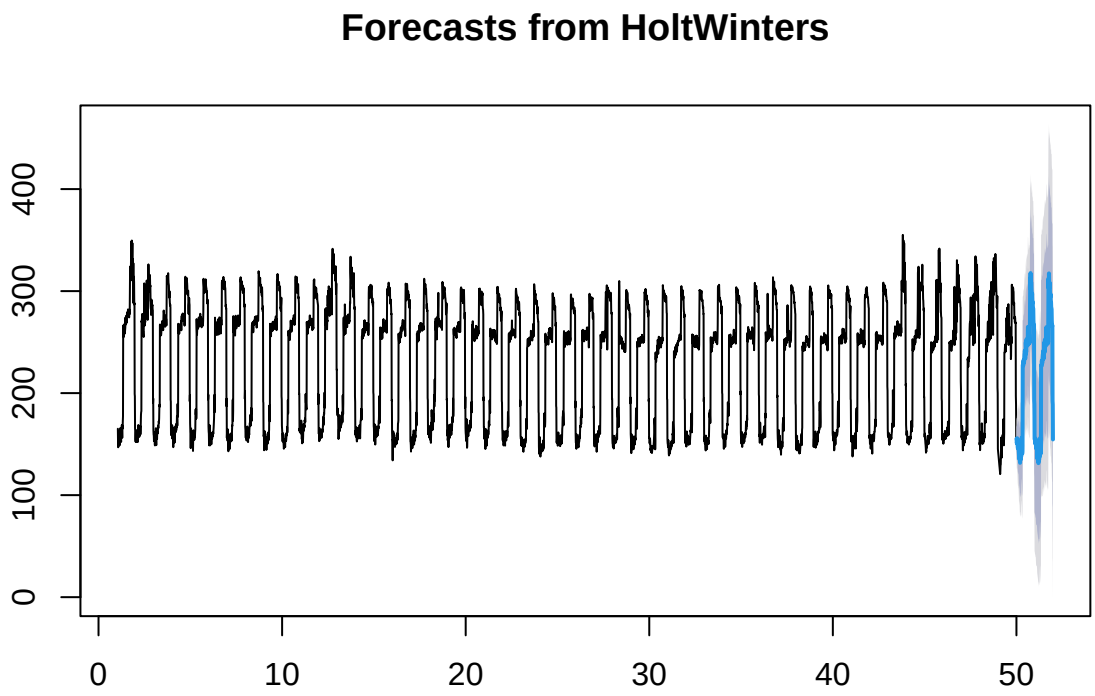
The plot of the residuals does not show white noise, as we can see regular spikes. Same for the plot of ACF and PACF, this suggests that there are remaining patterns or anomalies in the data that the model did not account for.

Holt Winters Additive : Finetuning I will fine tune the model finding playing on parameters to find a better fit:

```
hw_add_manual = HoltWinters(power_train, alpha = 0.6, beta = 0, gamma = 0.5, seasonal = "additive")
```

Forecast HW Additive using best parameters

```
# Generate forecasts using the best model
hw_add_forecasts <- forecast(hw_add_manual, h = 192)

# Print the forecasts
#print(forecasts)

# Plot the forecasts
plot(hw_add_forecasts)
```



Calculated accuracy with RMSE, AIC, BIC

```
#RMSE
hw_add_forecasts_values <- hw_add_forecasts$mean
hw_add_residuals <- power_test - hw_add_forecasts_values
rmse_hw_add <- sqrt(mean(hw_add_residuals^2))
cat("RMSE:", rmse_hw_add, "\n")
```

```
## RMSE: 11.531
```

```
#AIC & BIC
n <- length(power_train)
k <- 3  # Number of parameters (alpha, beta, gamma)
residuals <- residuals(hw_add_manual)
rss <- sum(residuals^2)
hw_add_aic <- n * log(rss/n) + 2 * k
hw_add_bic <- n * log(rss/n) + log(n) * k
cat("AIC:", hw_add_aic, "\n")
```

```
## AIC: 20995.58
```

```
cat("BIC:", hw_add_bic, "\n")
```

```
## BIC: 21014.95
```

Holt Winters multiplicative

I try also the multiplicative model

With the multiplicative is better to use the dataset log because it will not work with zero values

```
constant <- abs(min(power_train)) + 1  # Adjust to make the minimum value positive
power_train_log <- log(power_train + constant)


power_train_log[power_train_log == 0] <- 1e-5


hw_mult = HoltWinters(power_train_log, seasonal = "multiplicative")

#summary(hw_mult)
```

Forecast HW Multiplicative

```
# Predict using the Holt-Winters model
predicted_log <- predict(hw_mult, n.ahead = 192)

# Reverse the log transformation
hw_mult_predicted_orig <- exp(predicted_log) - constant
```

Check RMSE

```
# RMSE calculation
residuals <- power_test - hw_mult_predicted_orig
rmse_hw_mult <- sqrt(mean(residuals^2))
cat("RMSE:", rmse_hw_mult, "\n")
```

```
## RMSE: 11.29504
```

Other models, Simple Exponential Smoothing: Holt's Linear Trend model

I create a Holt model damped and not damped

```
hw_notdamped <- holt(power_train, h=length(power_test))
hw_damped <- holt(power_train, damped = TRUE, phi = 0.9, h=length(power_test))

# Check summary and results
#summary(hw_notdamped)
#summary(hw_damped)

# Print the forecasts
# cat("Non-Damped Holt's Forecast:\n")
# print(hw_notdamped$mean)
#
# cat("Damped Holt's Forecast:\n")
# print(hw_damped$mean)
```

The RMSE for both is very bad, the model is not working properly, I will not pursue with these models

```
# Extract predicted values from the forecast objects
predicted_notdamped <- hw_notdamped$mean
predicted_damped <- hw_damped$mean

# Calculate residuals
residuals_notdamped <- power_test - predicted_notdamped
residuals_damped <- power_test - predicted_damped

# RMSE Calculation
rmse_notdamped <- sqrt(mean(residuals_notdamped^2))
rmse_damped <- sqrt(mean(residuals_damped^2))

# Print RMSE for both models
cat("RMSE (Non-Damped Holt):", rmse_notdamped, "\n")
```

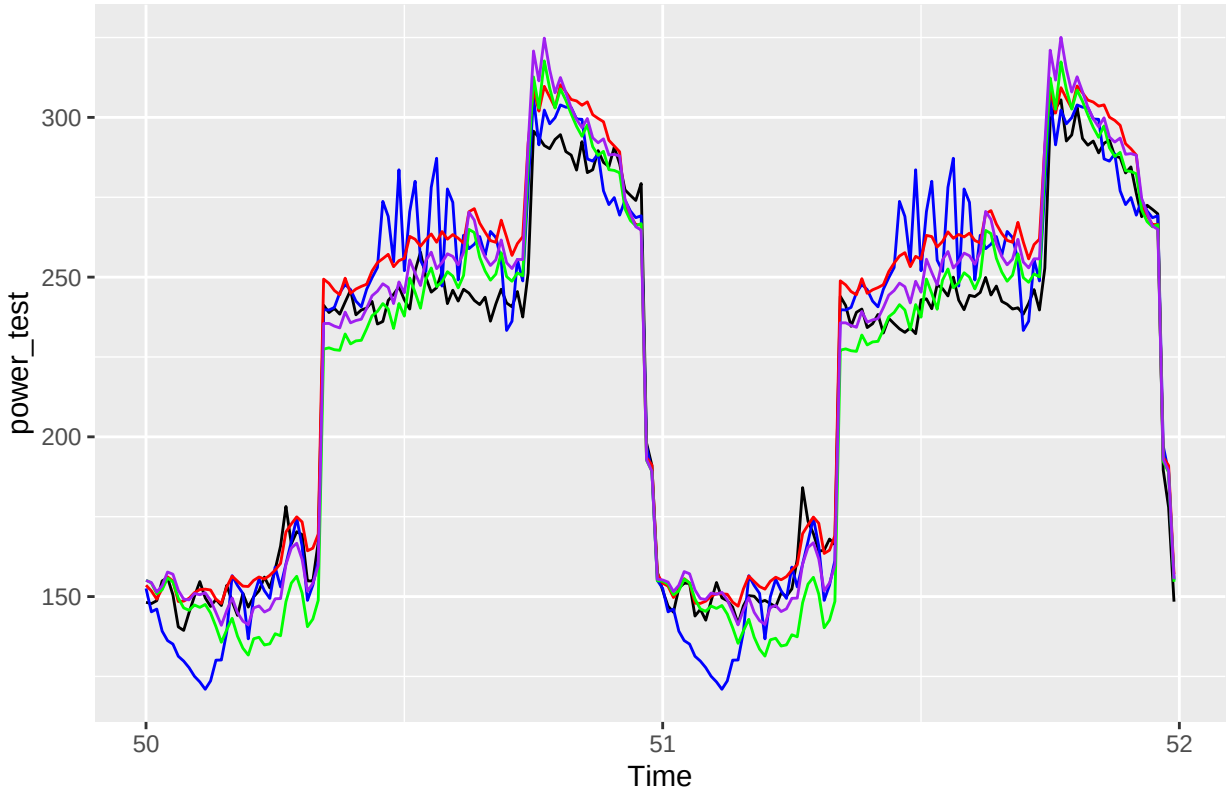
```
## RMSE (Non-Damped Holt): 82.88428
```

```
cat("RMSE (Damped Holt):", rmse_damped, "\n")
```

```
## RMSE (Damped Holt): 122.8583
```

Plot my best models

```
# Plot test data and reintegrated forecast
autoplot(power_test) +
  autolayer(prev_auto_sarima$mean, series = "SARIMA Forecast", color = "blue") +
  autolayer(prev_man_sarima$mean, series = "SARIMA Forecast", color = "red") +
  autolayer(hw_add_forecasts$mean, series = "HW Add Forecast (Best)", color = "green") +
  autolayer(hw_mult_predicted_orig, series = "HW Add Forecast (Best)", color = "purple")
```



Choose the best model based on AIC, BIC, RMSE

Recap Arima models: RMSE, AIC, BIC

```
# AIC and BIC for ARIMA models
aic_auto_arima <- AIC(fit_auto_arima)
bic_auto_arima <- BIC(fit_auto_arima)

aic_manu_sarima <- AIC(fit_manual_sarima)
bic_manu_sarima <- BIC(fit_manual_sarima)

rmse_auto_arima <- sqrt(mean((prev_auto_sarima$mean - power_test)^2))
rmse_manu_sarima <- sqrt(mean((prev_man_sarima$mean - power_test)^2))


# Summary for ARIMA models
cat("Auto ARIMA - AIC:", aic_auto_arima, "BIC:", bic_auto_arima, "RMSE:", rmse_auto_arima, "\n")
```

```
## Auto ARIMA - AIC: 35214.25 BIC: 35252.86 RMSE: 15.81268
```

```
cat("Manual SARIMA - AIC:", aic_manu_sarima, "BIC:", bic_manu_sarima, "RMSE:", rmse_manu_sarima, "\n")
```

```
## Manual SARIMA - AIC: 32622.52 BIC: 32693.29 RMSE: 13.17098
```

Recap HW models

Holt Winter Additive: RMSE, AIC, BIC

```
# 1. Calculate RMSE
rmse_hw_add <- sqrt(mean((hw_add_forecasts$mean - power_test)^2))
cat("RMSE for Holt-Winters model:", rmse_hw_add, "\n")
```

```
## RMSE for Holt-Winters model: 11.531
```

```
# 2. Calculate residuals
residuals_hw_add <- power_test - hw_add_forecasts$mean

# 3. Calculate AIC and BIC manually
# RSS (Residual Sum of Squares)
rss_hw_add <- sum(residuals_hw_add^2)


# number of parameters (3 parameters: alpha, beta, gamma)
k_best_model <- 3

# aic calculation
aic_best_model <- length(power_test) * log(rss_hw_add / length(power_test)) + 2 * k_best_model
cat("aic for holt-winters model:", aic_best_model, "\n")
```

```
## aic for holt-winters model: 944.8948
```

```
# bic calculation
bic_best_model <- length(power_test) * log(rss_hw_add / length(power_test)) + log(length(power_test)) *
cat("bic for holt-winters model:", bic_best_model, "\n")
```

```
## bic for holt-winters model: 954.6673
```

Holt Winter Multiplicative: RMSE, AIC, BIC

```
# 1. RMSE calculation
residuals <- power_test - hw_mult_predicted_orig
rmse <- sqrt(mean(residuals^2))
cat("RMSE:", rmse, "\n")
```

```
## RMSE: 11.29504
```

```
# 2. AIC and BIC calculation for Holt-Winters model (multiplicative)
# Residual sum of squares (RSS)
rss_hw_mult <- sum(residuals^2)

# Number of observations (length of test data)
n_hw_mult <- length(power_test)
```

```r
# Number of parameters (for Holt-Winters model, 3 parameters: alpha, beta, gamma)
k_hw_mult <- 3

# AIC calculation
aic_hw_mult <- n_hw_mult * log(rss_hw_mult / n_hw_mult) + 2 * k_hw_mult
cat("AIC for Holt-Winters model:", aic_hw_mult, "\n")
```

```
## AIC for Holt-Winters model: 936.9557
```

```r
# BIC calculation
bic_hw_mult <- n_hw_mult * log(rss_hw_mult / n_hw_mult) + log(n_hw_mult) * k_hw_mult
cat("BIC for Holt-Winters model:", bic_hw_mult, "\n")
```

```
## BIC for Holt-Winters model: 946.7282
```

Conclusion:

Even if the HW Multiplicative shows a slightly better RMSE I choose to go ahead with the HW Additive because I think it is the most appropriate model for this case since the seasonal variations are roughly constant over time. In regard to the AIC and BIC the both HW model are highly better than the SARIMA ones.

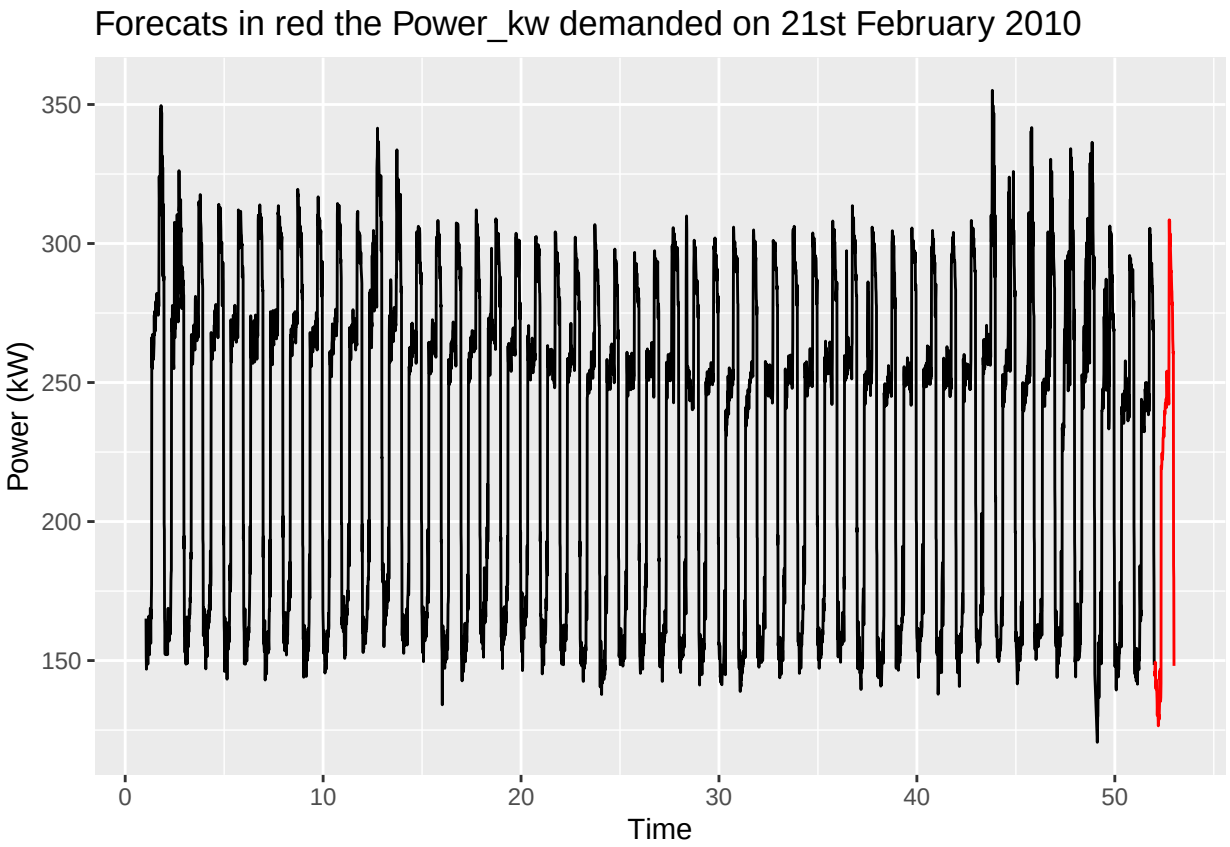
Forecast 21st February of Power kw consumption without using Outdoor Temperature

I train the model using the complete ts object created at the beginning for data exploring

```r
# Fit the model with the best parameters
HW_best_model <- HoltWinters(elec_ts, alpha = 0.6,
                             beta = 0,
                             gamma = 0.5)

#forecast using the best model
hw_forecast_power = forecast(HW_best_model, h = 96)
```

Plot the forecast for the 21st February

```r
# Plot the complete data and forecast
autoplot(elec_ts) +
  autolayer(hw_forecast_power$mean, series = "HW Add Forecast", color = "red") +
  ggtitle("Forecats in red the Power_kw demanded on 21st February 2010") +
  ylab("Power (kW)") +
  xlab("Time") +
  theme(legend.position = "right")
```

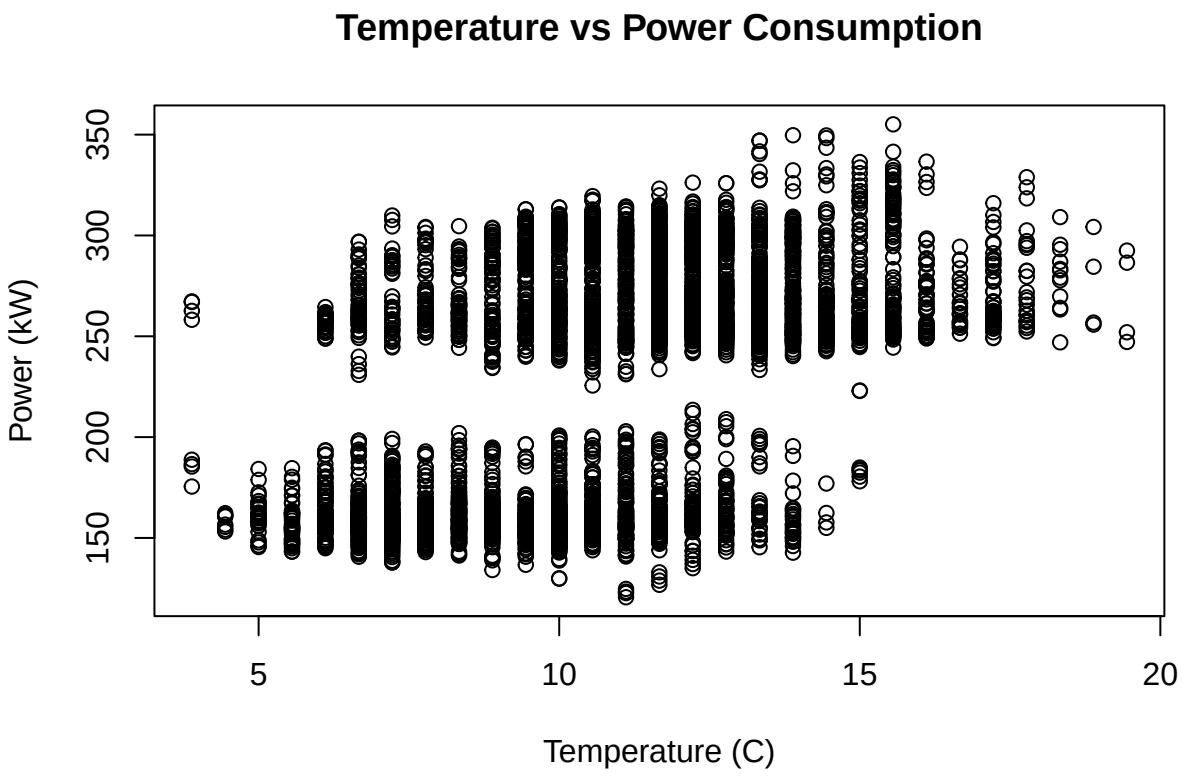


Create the model to forecast the Power adding the Temperature feature

I will now prepare the data for the forecast using the Temperature

I prepare train and test set with temperature

```
# Create time series objects (without temperature)
frequency = 96  # 96 intervals per day (15-minute intervals)
temp_train = ts(train_data$Temp_C, c(1,6), end = c(49, 96), frequency = frequency)
#I do not put the end in the test part because the boundary is already set in the code above
temp_test = ts(test_data$Temp_C, start = c(50, 1),  frequency = frequency)

#plot Temperature vs Power
plot(temp_train, power_train,
     xlab = "Temperature (C)", ylab = "Power (kW)",
     main = "Temperature vs Power Consumption")
```



From the plot no real trend is visible between the two variables so I will not create a new column with the Temp squared.

Modelling with Arima

Auto Arima model

```
auto_arima_temp = auto.arima(power_train, xreg = temp_train )
summary(auto_arima_temp) #ARIMA(5,0,0)(0,1,0)[96]
```

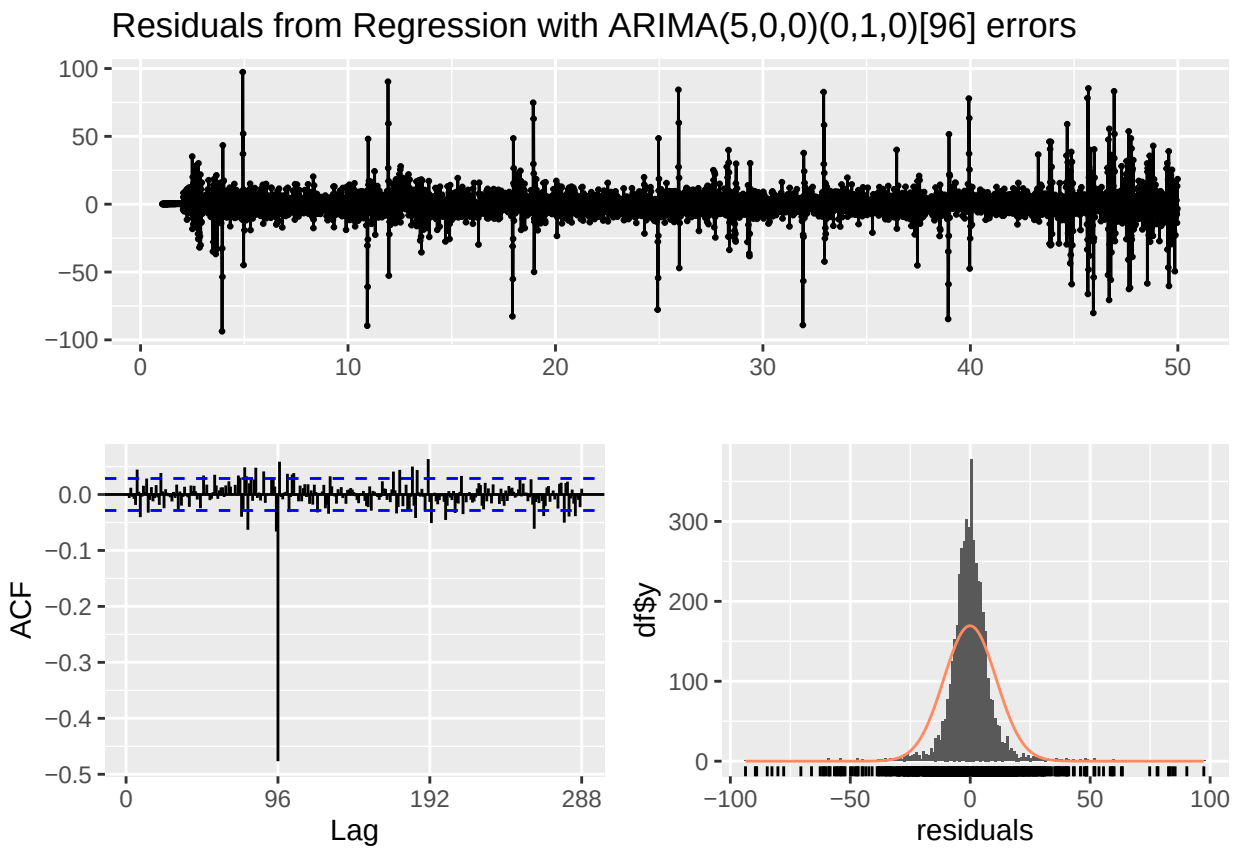
```
## Series: power_train
## Regression with ARIMA(5,0,0)(0,1,0)[96] errors
##
## Coefficients:
##          ar1     ar2     ar3      ar4     ar5    xreg
##       0.6690  0.0709  0.1564  -0.2876  0.1334  0.6405
## s.e.  0.0146  0.0171  0.0170   0.0171  0.0146  0.2279
##
## sigma^2 = 122.6:  log likelihood = -17597.26
## AIC=35208.52   AICc=35208.54   BIC=35253.56
##
## Training set error measures:
##                      ME     RMSE      MAE        MPE     MAPE      MASE
```

```
## Training set -0.0923762 10.95325 6.430485 -0.1546131 2.905205 0.7368078
##                       ACF1
## Training set 0.0009581283
```

```
summary(auto_arima_temp)
```

```
## Series: power_train
## Regression with ARIMA(5,0,0)(0,1,0)[96] errors
##
## Coefficients:
##          ar1     ar2     ar3      ar4     ar5    xreg
##       0.6690  0.0709  0.1564  -0.2876  0.1334  0.6405
## s.e.  0.0146  0.0171  0.0170   0.0171  0.0146  0.2279
##
## sigma^2 = 122.6:  log likelihood = -17597.26
## AIC=35208.52   AICc=35208.54   BIC=35253.56
##
## Training set error measures:
##                       ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -0.0923762 10.95325 6.430485 -0.1546131 2.905205 0.7368078
##                       ACF1
## Training set 0.0009581283
```

```
checkresiduals(auto_arima_temp)
```



```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(5,0,0)(0,1,0)[96] errors
## Q* = 1488, df = 187, p-value < 2.2e-16
##
## Model df: 5.   Total lags used: 192
```

Fine tuned Sarima

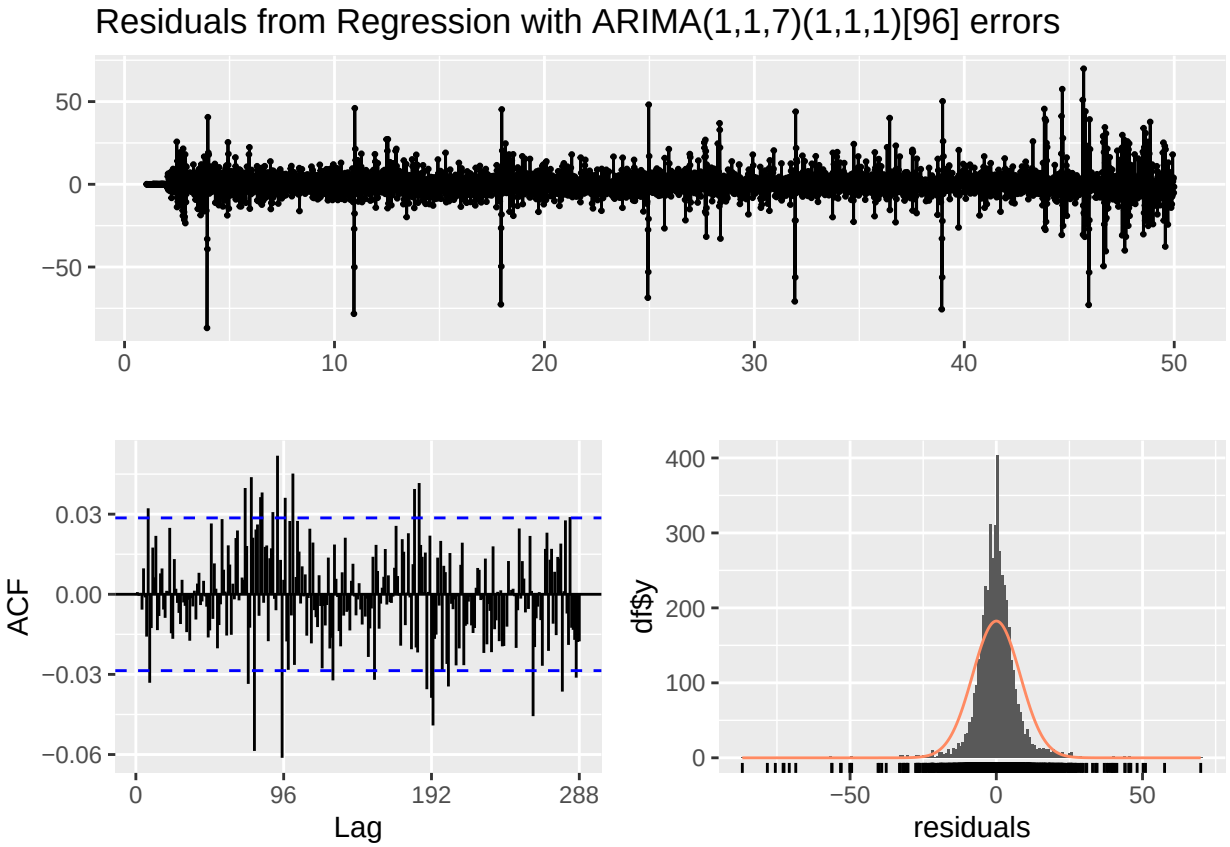
I try my fine tuned sarima with same parameters as for the model without temperature

```
fit_temp_sarima <- Arima(
  power_train, xreg = temp_train,
  order = c(1, 1, 7),
  seasonal = list(order = c(1, 1, 1))
)
```

```
summary(fit_temp_sarima)
```

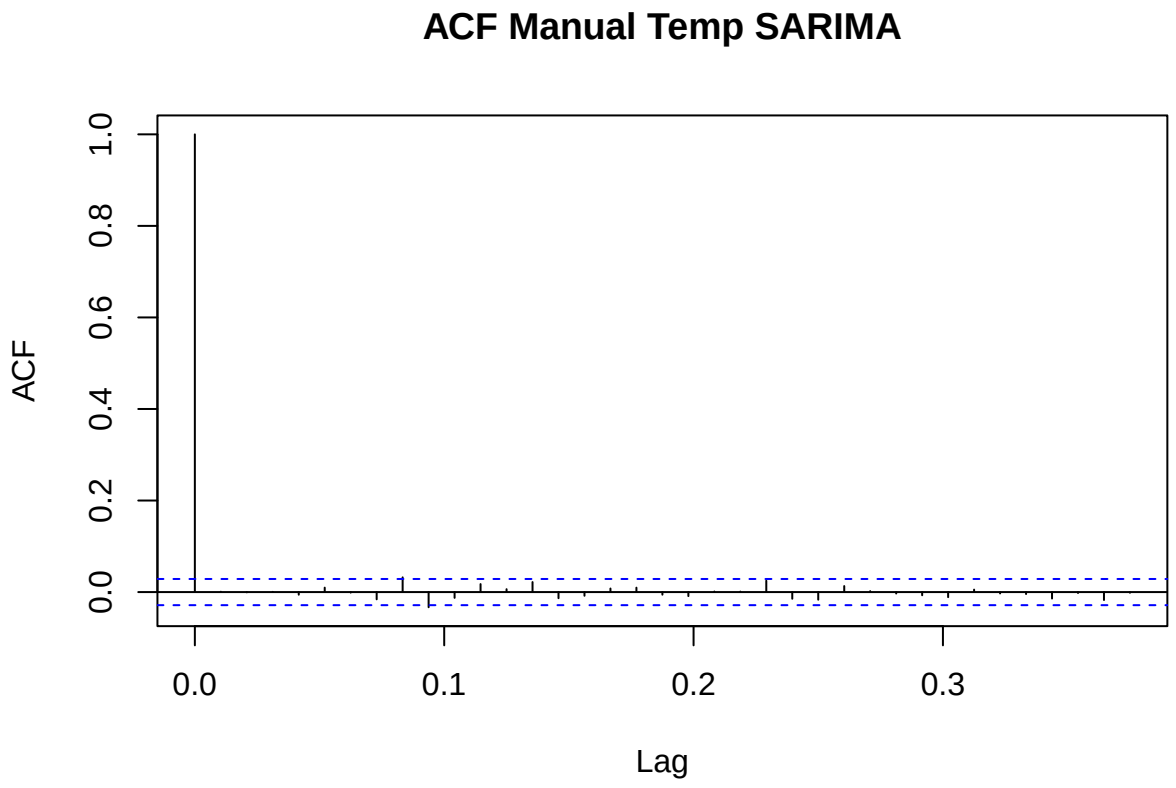
```
## Series: power_train
## Regression with ARIMA(1,1,7)(1,1,1)[96] errors
##
## Coefficients:
##          ar1      ma1     ma2     ma3      ma4     ma5     ma6      ma7
##       0.8449  -1.1869  0.1410  0.1645  -0.3172  0.1961  0.0157  -0.0097
## s.e.  0.0348   0.0378  0.0262  0.0233   0.0224  0.0244  0.0240   0.0173
##          sar1     sma1    xreg
##       -0.0487  -0.8674  0.6252
## s.e.   0.0171   0.0092  0.2256
##
## sigma^2 = 67.7:  log likelihood = -16296.66
## AIC=32617.32   AICc=32617.39   BIC=32694.53
##
## Training set error measures:
##                       ME     RMSE      MAE         MPE     MAPE      MASE
## Training set 0.02118881 8.133086 4.976658 -0.09063196 2.248655 0.5702277
##                       ACF1
## Training set 0.0008049914
```

```
checkresiduals(fit_temp_sarima)
```

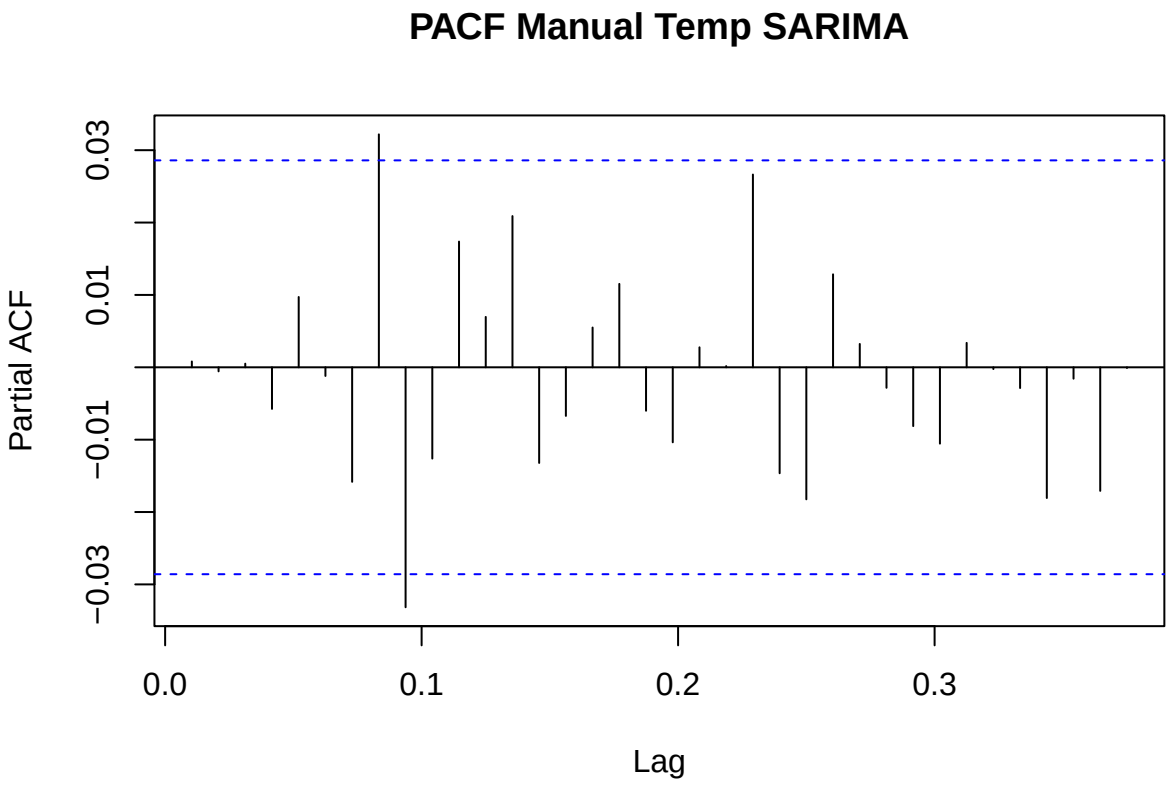



```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(1,1,7)(1,1,1)[96] errors
## Q* = 296.92, df = 182, p-value = 1.554e-07
##
## Model df: 10.   Total lags used: 192
```

```r
# Plot PACF for the residuals
residuals_temp_sarima <- residuals(fit_temp_sarima)
acf(residuals_temp_sarima, main = "ACF Manual Temp SARIMA")
```




```r
pacf(residuals_temp_sarima, main = "PACF Manual Temp SARIMA")
```



Calculate RMSE, AIC, BIC for my fine tuned Sarima model

```
# Calculate RMSE
sarima_temp_fitted_values <- fitted(fit_temp_sarima)
rmse_value <- rmse(power_train, sarima_temp_fitted_values)
print(paste("RMSE:", rmse_value))
```

```
## [1] "RMSE: 8.13308593121154"
```

```
# Extract AIC and BIC
aic_value <- AIC(fit_temp_sarima)
bic_value <- BIC(fit_temp_sarima)
print(paste("AIC:", aic_value))
```

```
## [1] "AIC: 32617.3185520753"
```

```
print(paste("BIC:", bic_value))
```

```
## [1] "BIC: 32694.5295073224"
```

Forecast with Sarima using test set

```
Prev_sarima_temp = forecast(fit_temp_sarima, xreg = temp_test, h = length(temp_test))
```

I convert the forecast into dataframe to be able to plot it with the other models

```
#convert SARIMA predictions to a data frame
sarima_pred_df = data.frame(Time = time(power_test),
                            SARIMA = as.numeric(Prev_sarima_temp$mean))
```

Modelling with Linear Regression model

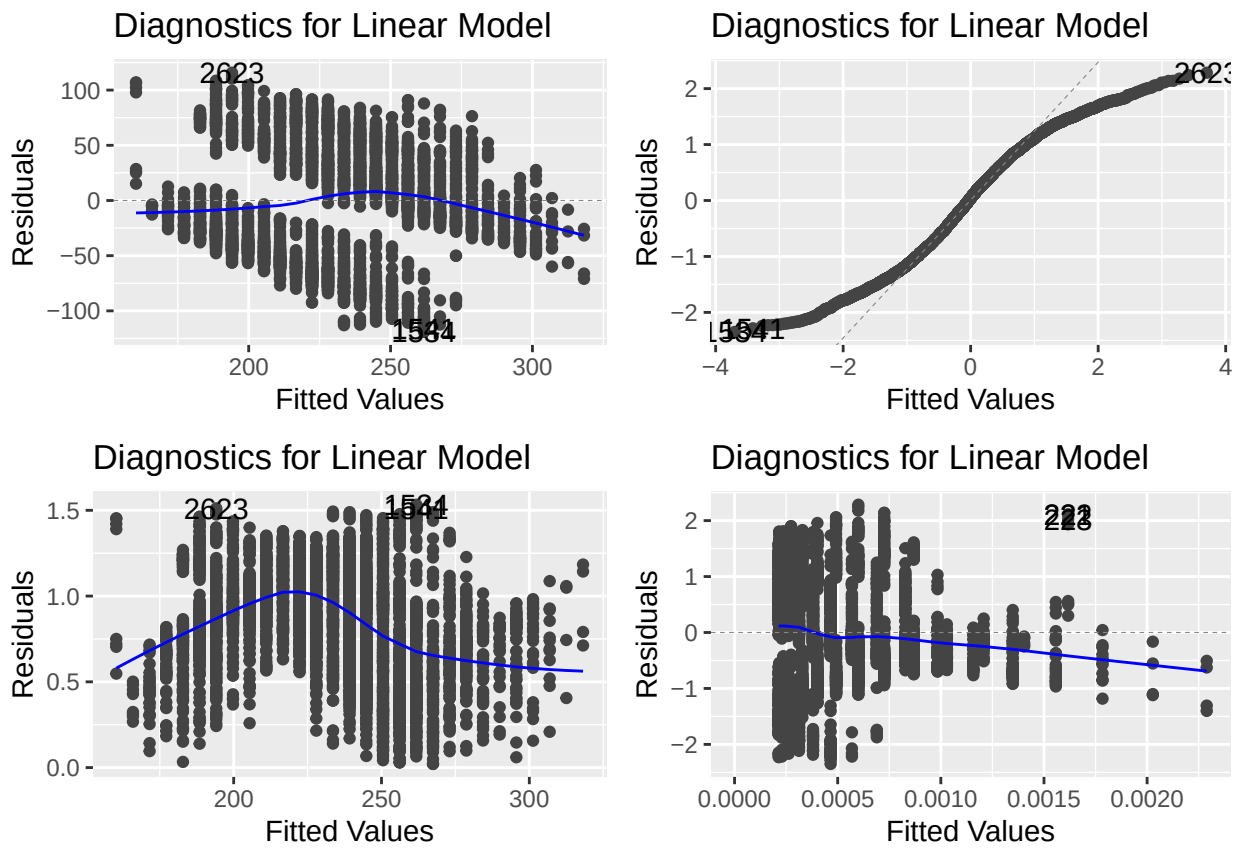
I wanted to try a Linear regression model

```
fit_lm <- lm(power_train ~ temp_train )

#residuals(fit_lm)
library(ggfortify)
```

```
## Warning: package 'ggfortify' was built under R version 4.3.3
```

```
## Registered S3 methods overwritten by 'ggfortify':
##   method                 from
##   autoplot.Arima         forecast
##   autoplot.acf           forecast
##   autoplot.ar            forecast
##   autoplot.bats          forecast
##   autoplot.decomposed.ts forecast
##   autoplot.ets           forecast
##   autoplot.forecast      forecast
##   autoplot.stl           forecast
##   autoplot.ts            forecast
##   fitted.ar              forecast
##   fortify.ts             forecast
##   residuals.ar           forecast
```

```
# Use ggfortify's autoplot for lm objects
autoplot(fit_lm) +
  labs(title = "Diagnostics for Linear Model",
       x = "Fitted Values",
       y = "Residuals")
```



The residuals appear to show some patterns, which could indicate non-linearity or heteroscedasticity. This suggests that the linear model might not be the appropriate model. Therefore I will not go ahead with this model.

Modelling using Machine learning

Random forest model

I prepare the data for training and testing a Random Forest (RF) model. Since RF models cannot directly use time series (ts) objects, I need to
convert the data into a suitable format.

```
#convert the timestamp in numeric, necessary to work better with rf model
train_data$Timestamp_num = as.numeric(train_data$Timestamp)
test_data$Timestamp_num = as.numeric(test_data$Timestamp)
#elec_rf_model$Timestamp_num = as.numeric(elec_rf_model$Timestamp) to remove

#prepare features and target for training and testing sets
train_features = train_data[, c("Timestamp_num", "Temp_C")]
train_target = train_data$Power_kW
test_features = test_data[, c("Timestamp_num", "Temp_C")]
test_target = test_data$Power_kW
```

Create the Random Forest model

```r
#create RF model
temp_rf = randomForest(x = train_features, y = train_target, ntree = 500)
```

Create forecast

```r
#predictions on test data
rf_pred = predict(temp_rf, newdata = test_data)
```

Check on quality of the model

```r
#actual values from the test set
actual_values = test_data$Power_kW

#calculate evaluation metrics
rf_mse = mean((actual_values - rf_pred)^2)
rf_rmse = sqrt(rf_mse) # (RMSE):  46.32554
cat("Root Mean Squared Error (RMSE): ", rf_rmse, "\n") #the RMSE is worse than the sarima model
```

```
## Root Mean Squared Error (RMSE):  45.83433
```

```r
# Calculate log-likelihood
rf_log_likelihood <- sum(dnorm(actual_values, mean = rf_pred, sd = sd(rf_pred), log = TRUE))

# Number of parameters (including the intercept)
rf_k <- length(temp_rf$forest$ndbigtree) + 1

# Calculate AIC
rf_aic_value <- -2 * rf_log_likelihood + 2 * rf_k
cat("AIC:", rf_aic_value, "\n")
```

```
## AIC: 3031.418
```

```r
# Calculate BIC
rf_n <- length(actual_values)
rf_bic_value <- -2 * rf_log_likelihood + log(rf_n) * rf_k
cat("BIC:", rf_bic_value, "\n")
```

```
## BIC: 4663.424
```

XGBOOST Model

```r
#1st I reuse the data preparation that I created for the RF model so not necessary
#to rewrite the code that is already created

#then prepare DMatrix for training
xg_train = xgb.DMatrix(data = as.matrix(train_features), label = train_target)

#prepare DMatrix for testing
xg_test = xgb.DMatrix(data = as.matrix(test_features), label = test_target)
```

```r
#set parameters for xgboost
params = list(
  objective = "reg:squarederror",  # for regression tasks
  booster = "gbtree",
  eta = 0.1,                          # learning rate
  max_depth = 6,                # maximum depth of trees
  nthread = 2                       # number of threads
)
```

Create XGBOOST model

```r
#train the model
xg_model = xgb.train(params = params, data = xg_train, nrounds = 300)
```

Create forecast

```r
# now I generate predictions using the XGBoost model
xg_pred = predict(xg_model, xg_test)
```

```r
#also ensure that xgboost predictions are in a data frame
xg_pred_df = data.frame(
  Time = time(power_test),
  XGBoost_Predictions = xg_pred
)
```

Check on quality of the model

```r
xgboost_rmse_value <- rmse(test_target, xg_pred)
print(paste("RMSE:", xgboost_rmse_value))
```

```
## [1] "RMSE: 54.1857235603522"
```

```r
# Calculate log-likelihood
xg_log_likelihood <- sum(dnorm(test_target, mean = xg_pred, sd = sd(xg_pred), log = TRUE))

# Number of parameters (including the intercept)
xg_k <- length(xg_model$feature_names) + 1

# Calculate AIC
xg_aic_value <- -2 * xg_log_likelihood + 2 * xg_k
print(paste("AIC:", xg_aic_value))
```

```
## [1] "AIC: 2168.42629529163"
```

```r
# Calculate BIC
xg_n <- length(test_target)
xg_bic_value <- -2 * xg_log_likelihood + log(xg_n) * xg_k
print(paste("BIC:", xg_bic_value))
```

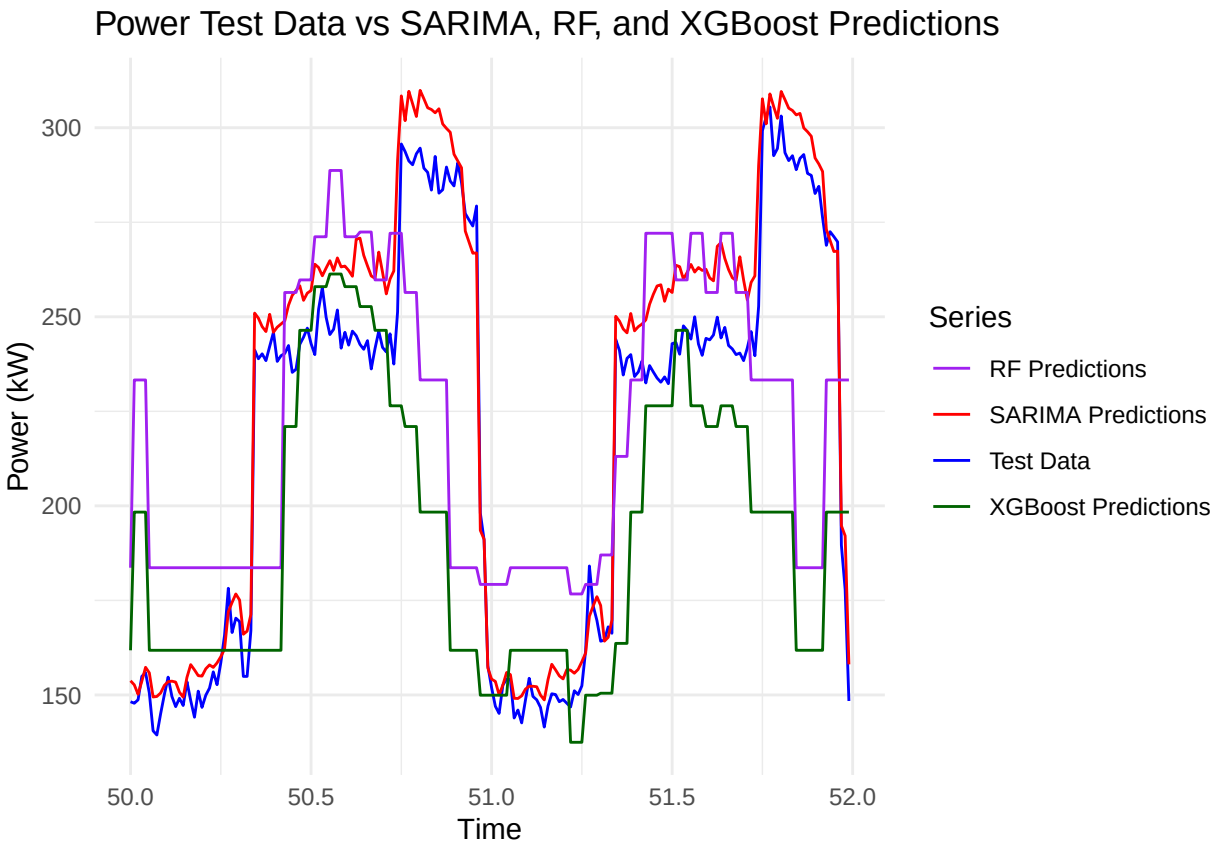
```
## [1] "BIC: 2178.19878140771"
```

Plot the create models and compare with the test set

```
#combine the test data and the predictions
plot_data = data.frame(
  Time = time(power_test),
  Test_Data = as.numeric(power_test),
  SARIMA_Predictions = sarima_pred_df$SARIMA,
  RF_Predictions = rf_pred,
  XGBoost_Predictions = xg_pred
)
```

```
ggplot(plot_data, aes(x = Time)) +
  geom_line(aes(y = Test_Data, color = "Test Data")) +
  geom_line(aes(y = SARIMA_Predictions, color = "SARIMA Predictions")) +
  geom_line(aes(y = RF_Predictions, color = "RF Predictions")) +
  geom_line(aes(y = XGBoost_Predictions, color = "XGBoost Predictions")) +
  ggtitle("Power Test Data vs SARIMA, RF, and XGBoost Predictions") +
  ylab("Power (kW)") +
  xlab("Time") +
  scale_color_manual(name = "Series",
                     values = c("Test Data" = "blue",
                                "SARIMA Predictions" = "red",
                                "RF Predictions" = "purple",
                                "XGBoost Predictions" = "darkgreen")) +
  theme_minimal() +
  theme(legend.position = "right")
```

```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```



Conclusion

The Prediction Accuracy of the Sarima Temp model is superior to the RF and XGboost due to its much lower RMSE. The Model Fit and Complexity are better in terms of AIC and BIC, for the rf and xgboost suggesting it might be a simpler model with a good fit.

Since in this case, prediction accuracy is paramount, I will o with the Sarima Temp model.

Forecast 21st February using Temperature feature

```
#data to be used for forecast, will use only the tail, the last 96 rows for forecast
#with temperature
elec_complete <- data_row %>%
  rename(
    Power_kW = `Power (kW)`,
    Temp_C = `Temp (C°)`
  )

# tail(elec_complete)
```

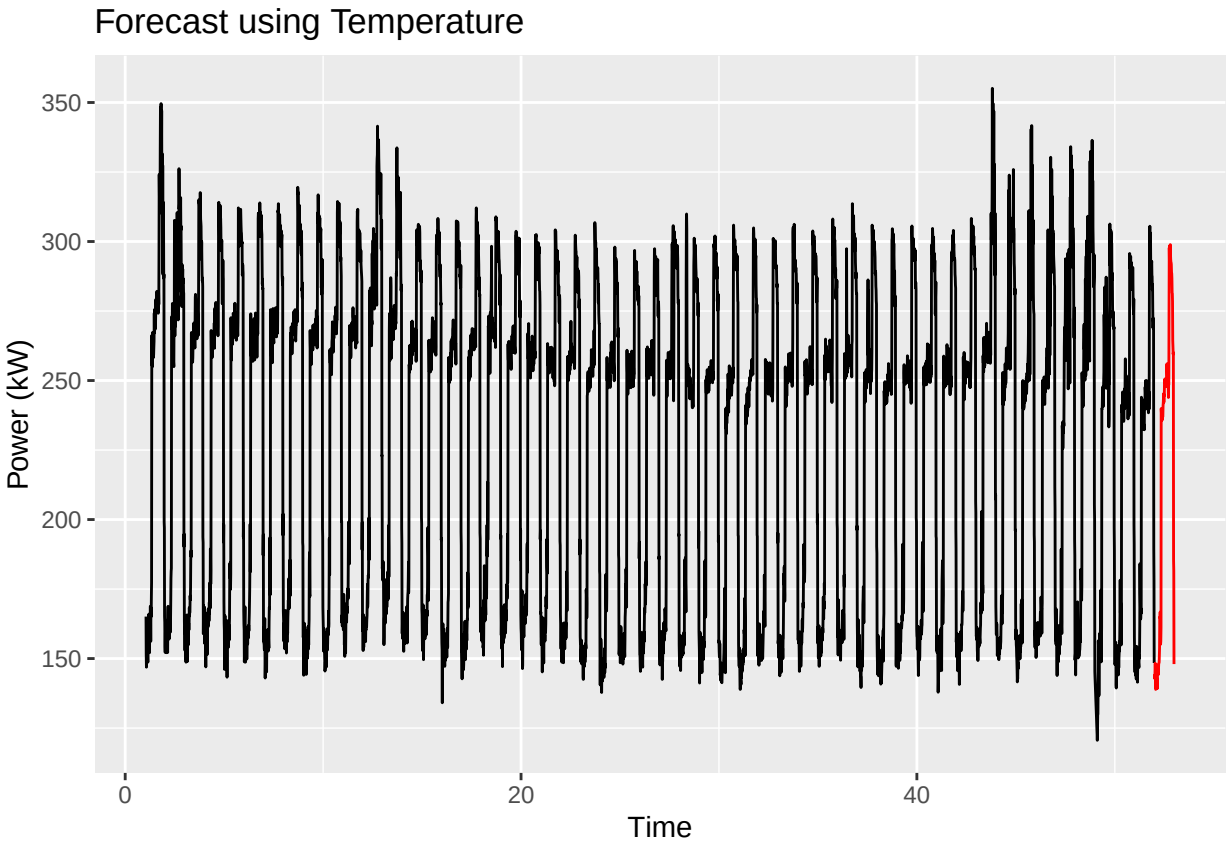
```r
#I now use the complete dataset of power that I already created called elec_ts


#then I create a ts for the complete temperature
temp_next = ts(elec$Temp_C, start = c(1, 6), frequency = frequency)

#I train the model using complete power and temperature data (until 20 Feb 2010)
sarima_next = Arima(
  elec_ts, xreg = temp_next,
  order = c(1, 1, 7),
  seasonal = list(order = c(1, 1, 1))
)
```

Forecast of 21st February using Temperature feature

```r
#extract the relevant temperature series for the next 96 time points
temp_forecast = tail(elec_complete$Temp_C, 96)  # Extract last 96 temperature observations

#I create a ts object for the temperature of the 21 Feb
temp_forecast_ts = ts(temp_forecast, frequency = 96)

#forecast the next 96 time points
sarima_temp_forecast = forecast(sarima_next, h = 96, xreg = temp_forecast_ts)
```

Plot the forecast of 21st February using Temperature feature

```r
autoplot(elec_ts) +
  autolayer(sarima_temp_forecast$mean, series = "Forecast using Temperature", color = "red") +
  ggtitle("Forecast using Temperature") +
  ylab("Power (kW)") +
  xlab("Time") +
  theme(legend.position = "right")
```



Export the two forecast in a excel file

```
#Put the 2 forecast results in an excel file
#correctly convert forecast power to a data frame
forecast_pow = as.data.frame(hw_forecast_power)
#extracting just the forecast points
point_forecast_pow = forecast_pow[["Point Forecast"]]
#create columns that I will use as 1st column in excel file
hw_power_notemp = as.data.frame(point_forecast_pow)


#correctly convert forecast with temp to a data frame
forecast_sar = as.data.frame(sarima_temp_forecast)
#extracting just the forecast points
point_forecast = forecast_sar[["Point Forecast"]]
#create columns that I will use as 2nd column in excel file
sarima_power_withtemp = as.data.frame(point_forecast)

#combine the two data frames into one, side by side
combined_df = data.frame(
  HW_Power_notemp = hw_power_notemp,
  SARIMA_Power_WithTemp = sarima_power_withtemp
)
```

```r
#rename the columns to more descriptive names
colnames(combined_df) = c("HoltWinters_Power_Only", "SARIMA_Power_With_Temperature")
#save the data frames to an Excel file
write_xlsx(combined_df, path = "C:/Users/Utilisateur/OneDrive/Desktop/Romeo_Lorena_secondsub.xlsx")
```