Time-Series Analysis Exam - Report

Carlos Eduardo Tussi Leite

2025-07-15

Electricity Consumption Time-Series Analysis and Forecast

1. Introduction

The goal of this project is to find the best model to forecast the energy consumption during a 15 minutes interval on a single day (21/02/2010). In order to do that, EDA is done to analyze the data set and clean it when needed. After, Feature Engineering is performed to try to get a better insight and find underlying information in the data (such as days of the week's influence and different daily consumption demand levels), including possible outliers.

In the modeling part, different types of models are tested with different hyper parameters (both statistical and machine learning models). The models are trained both with the covariate temperature and without it, in order to analyze its impact in the overall result. The models are compared using the MSE metric.

For the best model, cross-validation is performed with different splits of the train and test set to confirm that no overfitting is present. Finally, the models are retrained with the whole dataset to obtain the final best model: best model without temperature covariate and best model with the temperature covariate.

2. Data

Loading the Data

```
# Loading the data
data = read.csv("2025-06-Elec-train.csv")

summary(data)
```

```
##   Timestamp           Power..kW.      Temp..C..
##  Length:4987        Min.   :  0.0   Min.   : 3.90
##  Class :character   1st Qu.:162.9   1st Qu.: 9.40
##  Mode  :character   Median :253.0   Median :11.10
##                     Mean   :230.8   Mean   :10.95
##                     3rd Qu.:277.3   3rd Qu.:12.80
##                     Max.   :355.1   Max.   :19.40
##                     NA's   :96
```

```
# Changing column names for convenience
colnames(data) = c("time", "consumption", "temperature")
```

Overview

- The dataset contains 4987 observations that correspond to observations measured at every 15 minutes.
- Also, we can see the presence of 96 missing values for consumption that need to be further investigated.

Missing Values

- We can see that the 96 missing values for consumption are the ones to be predicted by our best model at the end, so the missing values do not need to be imputed, since they are not going to be used for training.

```
cat("First NA: ", head(data[is.na(data$consumption), ],1)$time, "\n")
```

```
## First NA:  2/21/2010 0:00
```

```
cat("Last NA:  ", tail(data[is.na(data$consumption), ],1)$time)
```

```
## Last NA:   2/21/2010 23:45
```

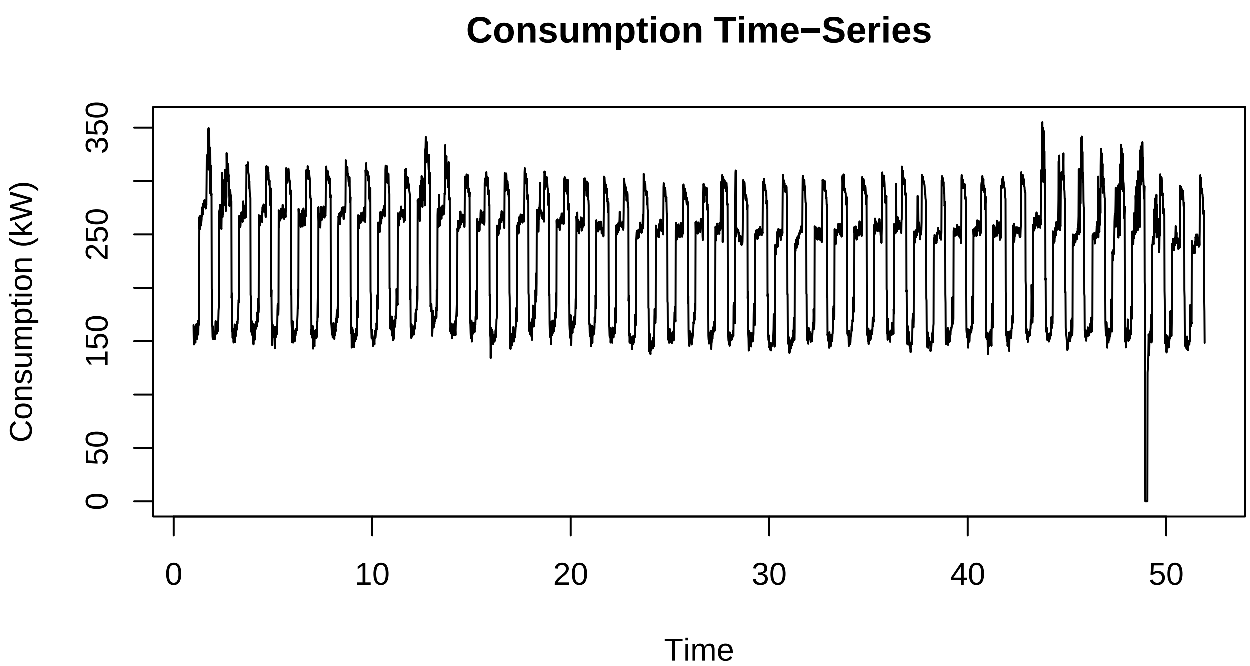
Time-Series Data

Converting the data into time series and separating the future values of temperature.

```
# Ignore the NAs observations
ts = ts(na.omit(data), frequency = 96)

# Preparing future temperature data for prediction with covariate
ts_future_temp = data[is.na(data$consumption),"temperature"]
```

EDA

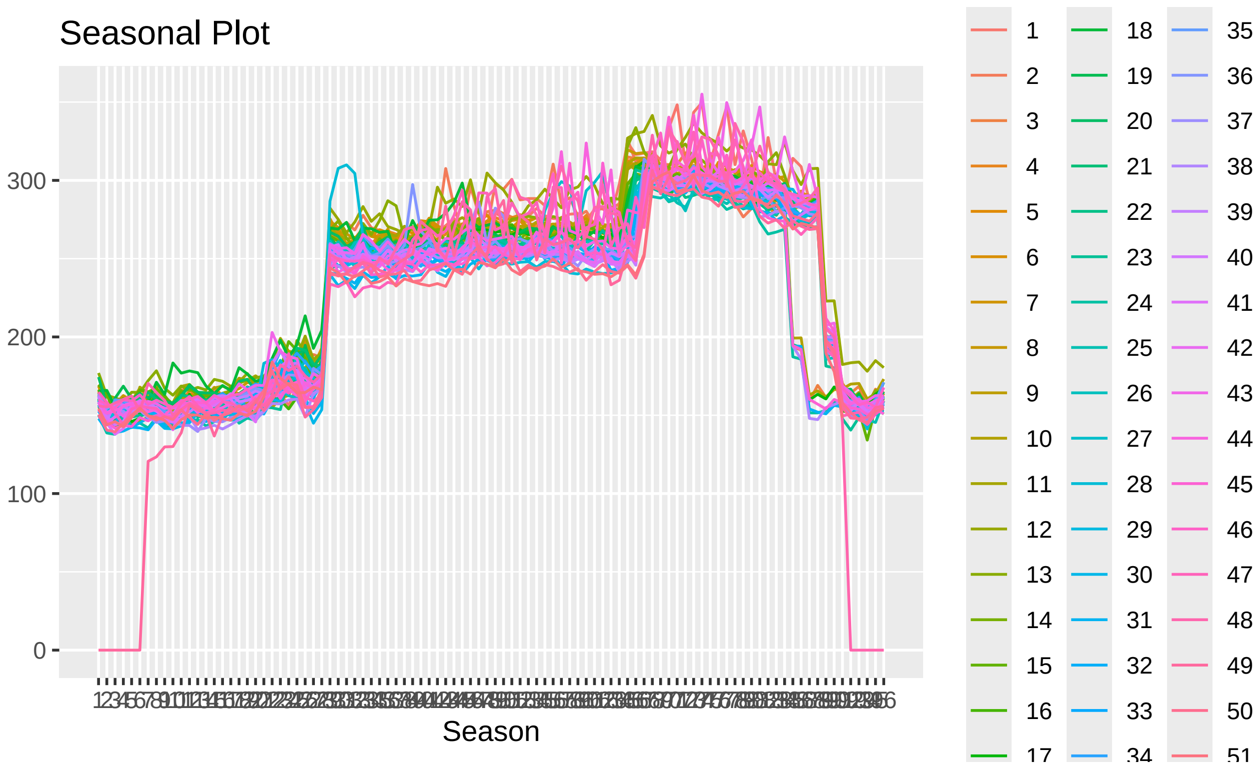
```
plot(ts[,"consumption"], main = "Consumption Time-Series", ylab = "Consumption (kW)")
```



Observations:

- Strong seasonality present.
- There seem to be no apparent trend, except for a slight decrease at the end that does not seem to be very significant.
- Except for a big spike towards the end of the series, which might indicate the presence of outliers, variance seems to be stable, so we assume homoscedasticity.
- The time series is clearly not stationary.

```
ggseasonplot(ts[,"consumption"], main = "Seasonal Plot")
```



Observations:

- Clear seasonal pattern of 1 day
- Different energy consumption levels during the day (lowest in the night, increase in the morning and a peak at the end of the day)

Observations with zero consumption

- We can see we have very few observations that have zero value for consumption which have a time frame of approximately 2 hours. It could indicate a blackout during that period or problem with the data collection. For these points, the lower average bigger than zero will be attributed.

```
ts[ts[,"consumption"] == 0, ]
```

```
##        time consumption temperature
##   [1,] 3836           0        12.8
##   [2,] 3837           0        12.2
##   [3,] 3838           0        12.2
##   [4,] 3839           0        12.2
##   [5,] 3840           0        12.2
##   [6,] 3841           0        11.7
##   [7,] 3842           0        11.7
##   [8,] 3843           0        11.7
##   [9,] 3884           0        11.7
##  [10,] 3885           0        11.1
##  [11,] 3886           0        11.1
```

Imputing initial outliers

- Imputing values with zero energy consumption with the smallest value different from zero.

```r
min_value = summary(ts[ts[,"consumption"] != 0, "consumption"])[1]

# Update the time series
ts[ts[,"consumption"] == 0, "consumption"] = min_value

# Update the initial data set
data[!is.na(data$consumption) &
       data$consumption == 0, "consumption"] = min_value
```

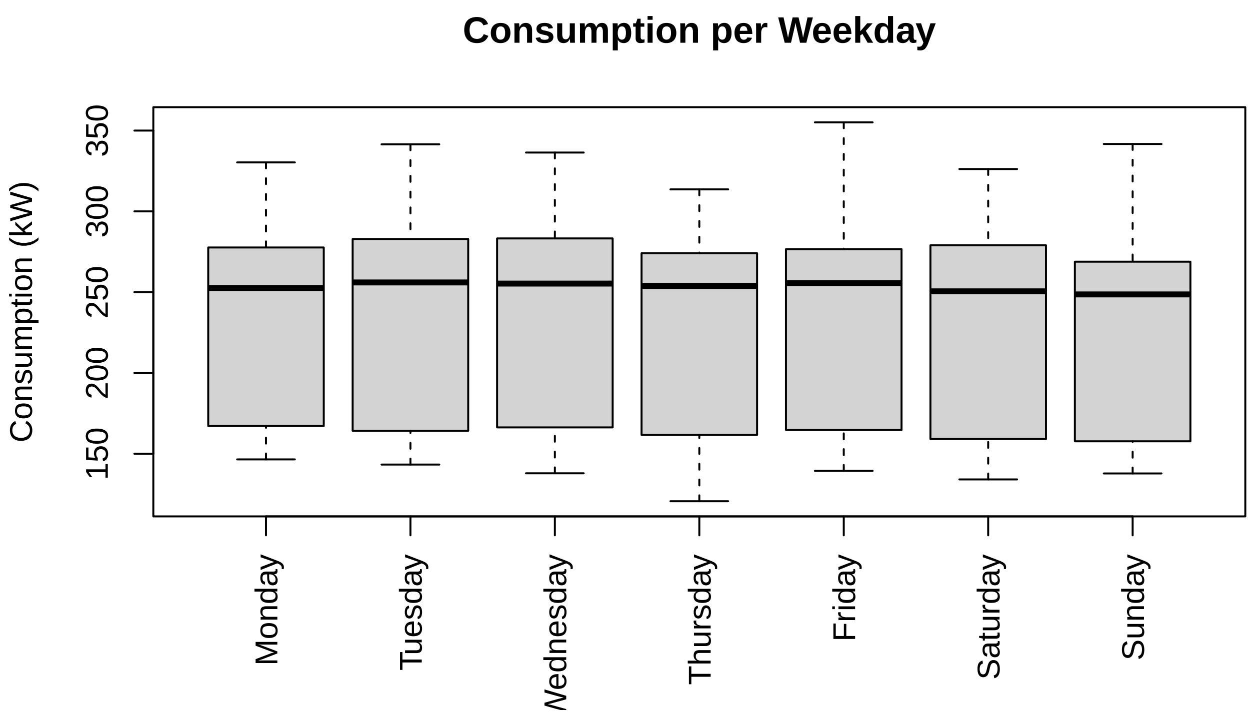
Features Engineering

```r
# Using POSIXct library to better interpret the days of the week and time
data[,"time"] = as.POSIXct(data[,"time"], format="%m/%d/%Y %H:%M")
```

Weekend Effect Analysis

- Analysis of the impact of the day of the week in the energy consumption.

```r
# Create Weekday feature
data["weekday"] = weekdays(data[,"time"])

# Transform into factors
data_cpy = data
data_cpy[,"weekday"] = factor(data[,"weekday"], levels = c("Monday","Tuesday",
        "Wednesday","Thursday","Friday","Saturday","Sunday"), ordered = TRUE)

# Consumption per weekday
boxplot(consumption ~
weekday, data = data_cpy, main = "Consumption per Weekday", las = 3, xlab = "",
ylab = "Consumption (kW)")
```



Analysis of the demand

- We look now at the different consumption peaks during one day.

```
mean_consumption = data %>% group_by(format(time, "%H:%M:%S")) %>%
  summarise(mean_consump = mean(consumption, na.rm = TRUE))
colnames(mean_consumption) = c("time","mean")

transitions = list("08:00:00", "08:15:00","17:00:00",
                   "17:15:00", "23:00:00", "23:15:00")

mean_consumption[mean_consumption$time %in% transitions, ]
```

```
## # A tibble: 6 x 2
##   time      mean
##   <chr>    <dbl>
## 1 08:00:00  175.
## 2 08:15:00  258.
## 3 17:00:00  259.
## 4 17:15:00  276.
## 5 23:00:00  266.
## 6 23:15:00  190.
```

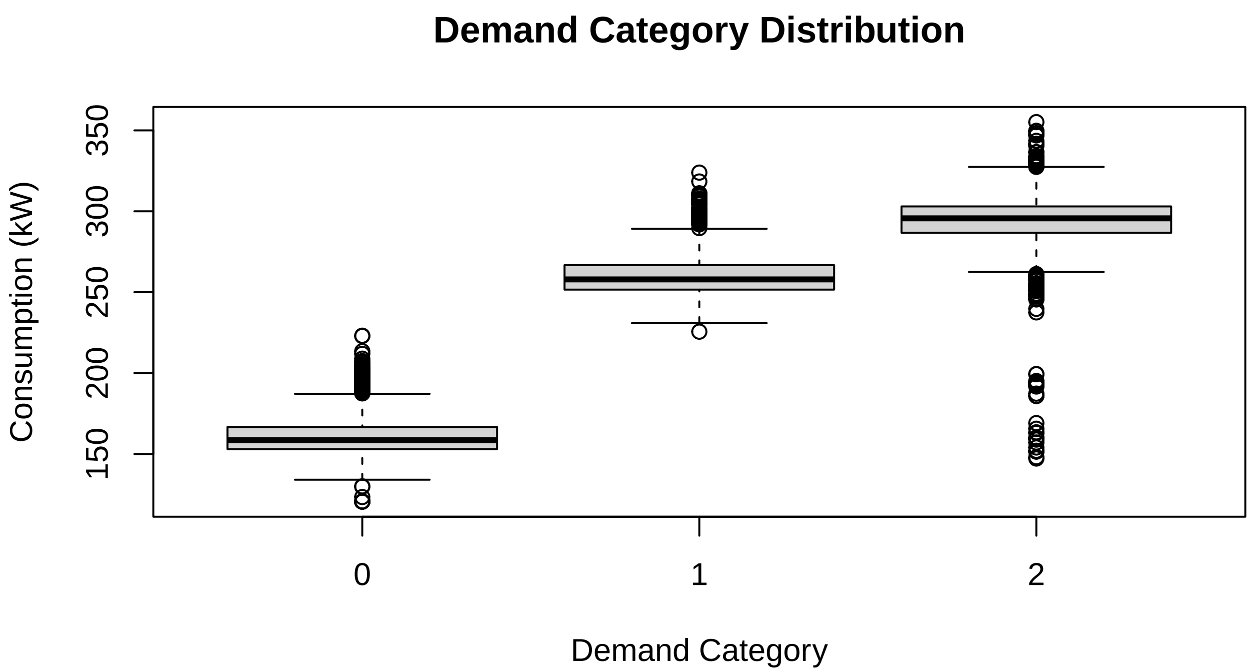
Observations:

- As we can see from the seasonality plot, we have three main different patterns of energy demand that are roughly classified as:
  - Low: 23:15 until 08:00
  - Medium: 08:15 until 17:00
  - High: 17:15 until 23:00

Demand feature

- Creating a new feature "demand" that contains the code for the demand period that we are considering:
  - "1" - 08:15 - 17:00 (Peak 1)
  - "2" - 17:15 - 23:00 (Peak 2)
  - "0" - 23:15 - 08:00 (Low)

```
data[,"demand"] = ifelse((format(data[,"time"], "%H:%M") >= "08:15") &
                           (format(data[,"time"], "%H:%M") <= "17:00"),1,
                         ifelse((format(data[,"time"], "%H:%M") >= "17:15") &
                                  (format(data[,"time"], "%H:%M") <= "23:00"),2,0))

boxplot(consumption ~ demand, data = data,
        main = "Demand Category Distribution",
        xlab = "Demand Category",
        ylab = "Consumption (kW)")
```

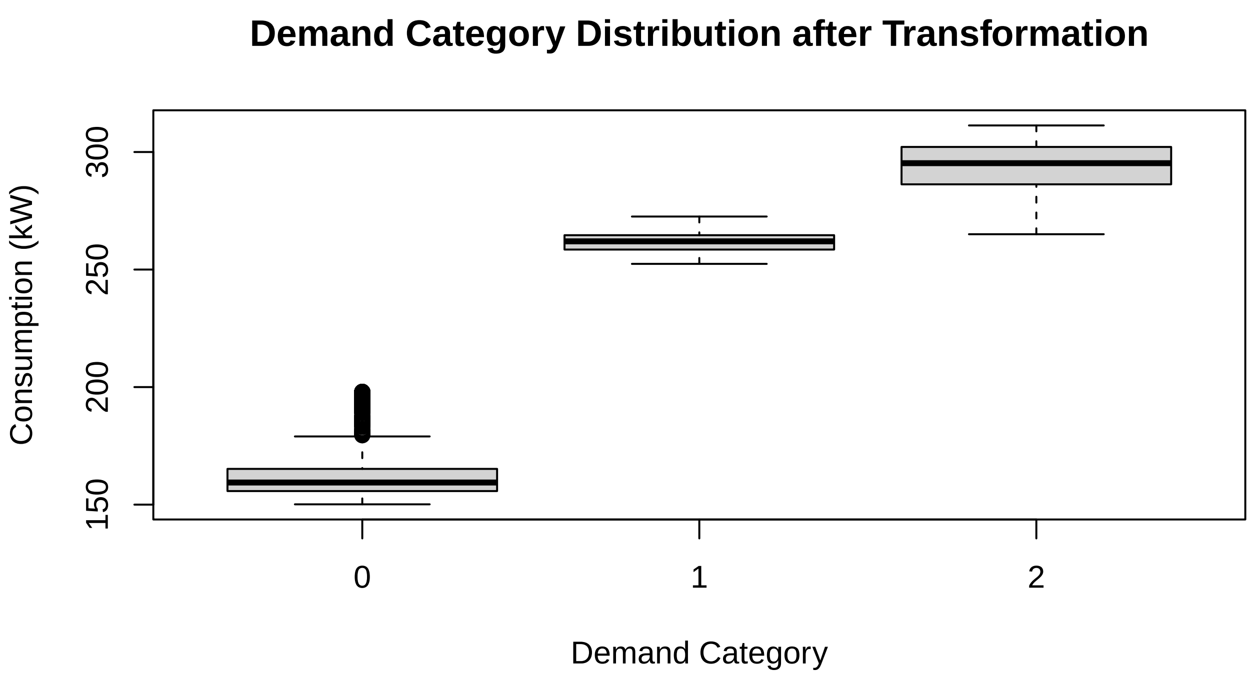


Observations:

- We can see that for the category 2 there are quite a lot of outliers, which can explain some days where the abrupt drop from demand high peak 2 to low demand happened before the pre-defined cutoff time (23:15).
- To address these outliers, we could have two approaches:
- Option 1: Re-label those categories to the correct one. Problem: Impossible to use it as an external feature, since we will not know the consumption to correctly label the demand beforehand.
- Option 2: Replace the outliers with some value:

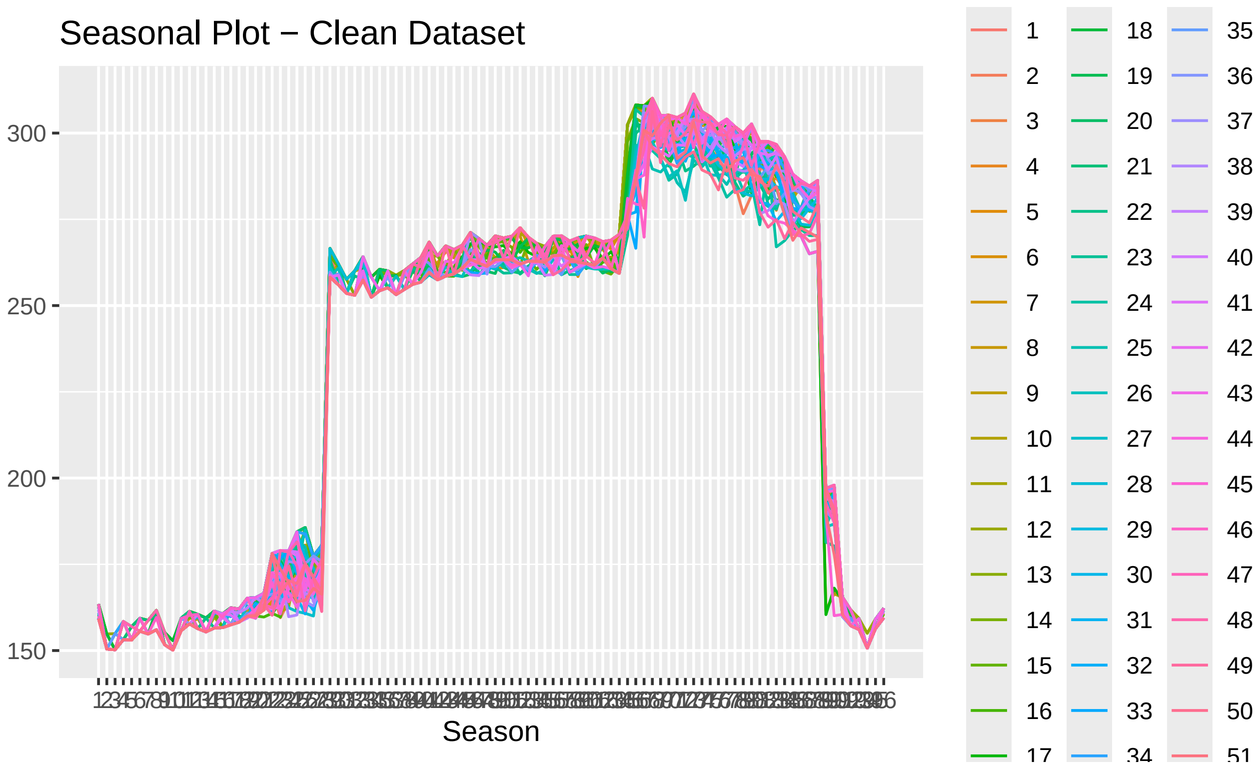
```r
library(dplyr)
# 1) Identify those observations that are mislabeled
# 2) Replace those days with the mean for all the days if lower value outliers
# 3) Replace those days with the third quantile for all the
# days if higher value outliers

# For each time period of the day (regardless of the day of the
# week at this point) calculate the respective mean and the
# respective 3rd quantile.

# Calcualate the mean for
time_means = data %>% group_by(format(time, "%H:%M:%S")) %>%
  summarise(mean_consump = mean(consumption, na.rm = TRUE))
colnames(time_means) = c("time", "mean")

# Calculate third quantiles
thrid_quantiles =data %>% group_by(format(time, "%H:%M:%S")) %>%
  summarise(third_quantile = quantile(consumption, 0.75, na.rm = TRUE))
colnames(thrid_quantiles) = c("time", "third quantile")

# Add a table with the observation mean for an
#observation considering the time only and not the days
data[,"time_HMS"] = format(data[,"time"],"%H:%M:%S")
data = left_join(data, time_means, by = c("time_HMS" = "time"))
data = left_join(data, thrid_quantiles, by = c("time_HMS" = "time"))
#data[,c("time", "time_HMS", "consumption", "mean")]


# Defining mean threshold
high_peak_mean = pull(time_means[time_means$time == "23:00:00" , "mean"])
medium_peak_mean = pull(time_means[time_means$time == "08:15:00" , "mean"])
low_peak_mean = pull(time_means[time_means$time == "01:15:00" , "mean"])


# Replacing values if lower than threshold (mean)
data[,"consumption"] = ifelse((data$consumption < high_peak_mean)
                              & (data$demand == 2),
                              data[,"mean"],
                              data[,"consumption"])

data[,"consumption"] = ifelse((data$consumption < medium_peak_mean)
                              & (data$demand == 1),
                              data[,"mean"],
                              data[,"consumption"])

data[,"consumption"] = ifelse((data$consumption < low_peak_mean)
                              & (data$demand == 0),
                              data[,"mean"],
                              data[,"consumption"])


# Replacing values if higher than threshold (3rd quantile)
data[,"consumption"] = ifelse((data$consumption > data[,"third quantile"])
```

```r
                              & (data$demand == 2),
                              data[,"third quantile"],
                              data[,"consumption"])

data[,"consumption"] = ifelse((data$consumption > data[,"third quantile"])
                              & (data$demand == 1),
                              data[,"third quantile"],
                              data[,"consumption"])

data[,"consumption"] = ifelse((data$consumption > data[,"third quantile"])
                              & (data$demand == 0),
                              data[,"third quantile"],
                              data[,"consumption"])

# Verifying results
boxplot(consumption ~ demand, data = data,
        main = "Demand Category Distribution after Transformation",
        xlab = "Demand Category",
        ylab = "Consumption (kW)")
```



Pre-processed data

```r
# Remove NAs from future temperature
data_clean = na.omit(data)

# Convert imto Time Series
ts = ts(data_clean[,-1], freq = 96) # Removing time columns

ggseasonplot(ts[,"consumption"], main = "Seasonal Plot - Clean Dataset")
```



Observations

- We can see that now the season plot does not have the outliers for the demand level being mislabelled.
- The data set seems to be cleaned at this point, despite some variations in the amplitude on each demand level.

Data Split

- Initially, to select the best model type, the time series will be split into train and test set.
- After hyper tuning and comparison with all models, the best model will be cross-validated to ensure that it did not overfit our data.
- The cross-validation is done at the end of this report.

Performing a 80/20 split:

```
# Roughly 80/20 split
ts_train = window(ts, end = c(40, 96))
ts_test = window(ts, start = c(41,1))
test_horizon = length(ts_test[,"consumption"])
```

3. Modeling

The best model selection was divided into two parts: without using temperature covariate and with the temperature covariate as part of the model training. For each model trained, the error is collected for comparison at the end.

```r
# Data frame that will contain all the errors for comparison
errors_df = data.frame(
  method = character(),
  covariate = numeric(),
  MSE = numeric()
)
```

- Support functions to be used with different models:

```r
# Support function to calculate the error with train and test set
mse_error = function(y_pred, y_test){
  mse = sqrt(mean((y_pred-y_test)^2))
  return (mse)
}
```

```r
# Manually check residuals for some models (like neural networks)
check_residuals = function(ypred, ytrue, freq){
  # Calculate residuals
  residuals = ypred - ytrue
  ts_res= ts(residuals,frequency = freq)
  print(Box.test(ts_res, type = "Ljung-Box"))
}
```

```r
# This function plots the predicted
# forecast and the true forecast from the test set
plot_forecast = function(train, test, pred, xlim = c(40, 55), title)
{
  plot(train, xlim = xlim, main = title, ylab = "Consumption (kW)")
  lines(pred, col = "blue")
  lines(test, col = "red")
  legend("topright",  legend = c("Train", "Forecast", "Test"),
        col = c("black", "blue", "red"),
       lty = 1)
}
```

Without Temperature Covariate

- The time series is now trained with different models without using temperature as covariate.
  - Exponential Smoothing
  - SARIMA
  - TSLM
  - Machine Learning Models

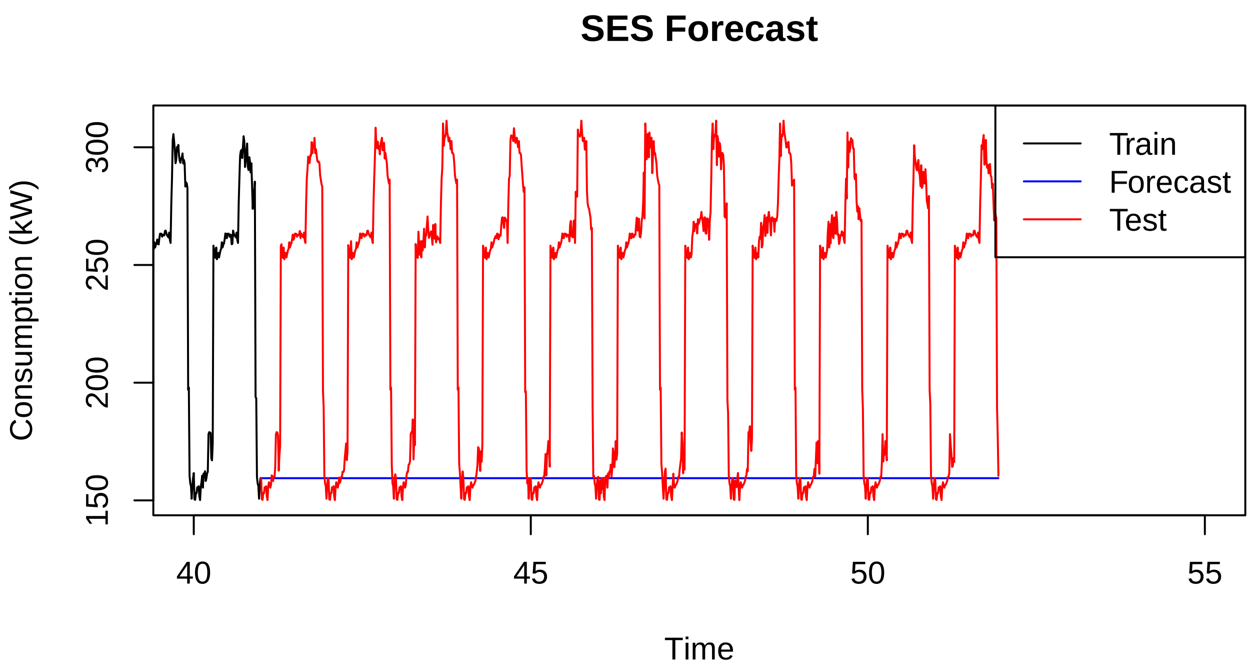
Exponential Smoothing

```r
# Train and Forecast
fit_ses = ses(ts_train[,"consumption"], h = test_horizon)
pred = fit_ses$mean
```

```r
# Save Error
mse = mse_error(fit_ses$mean, ts_test[,"consumption"])
cat("SES MSE: ",  mse, "\n")
```

```
## SES MSE:  91.86519
```

```r
errors_df = rbind(errors_df, data.frame(
                                        method = "SES",
                                        covariate = FALSE,
                                        MSE = mse))

# Plot Forecast
plot_forecast(train = ts_train[,"consumption"], test = ts_test[,"consumption"],
              pred = pred, xlim = c(40, 55), title = "SES Forecast")
```

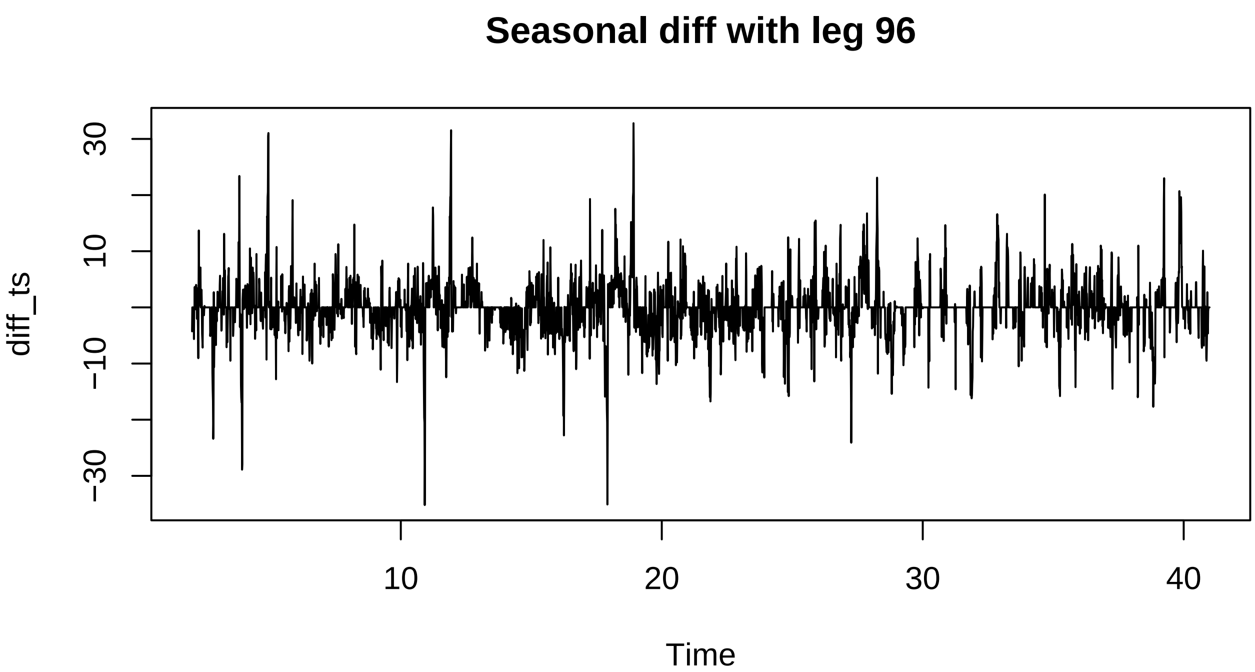


Sarima

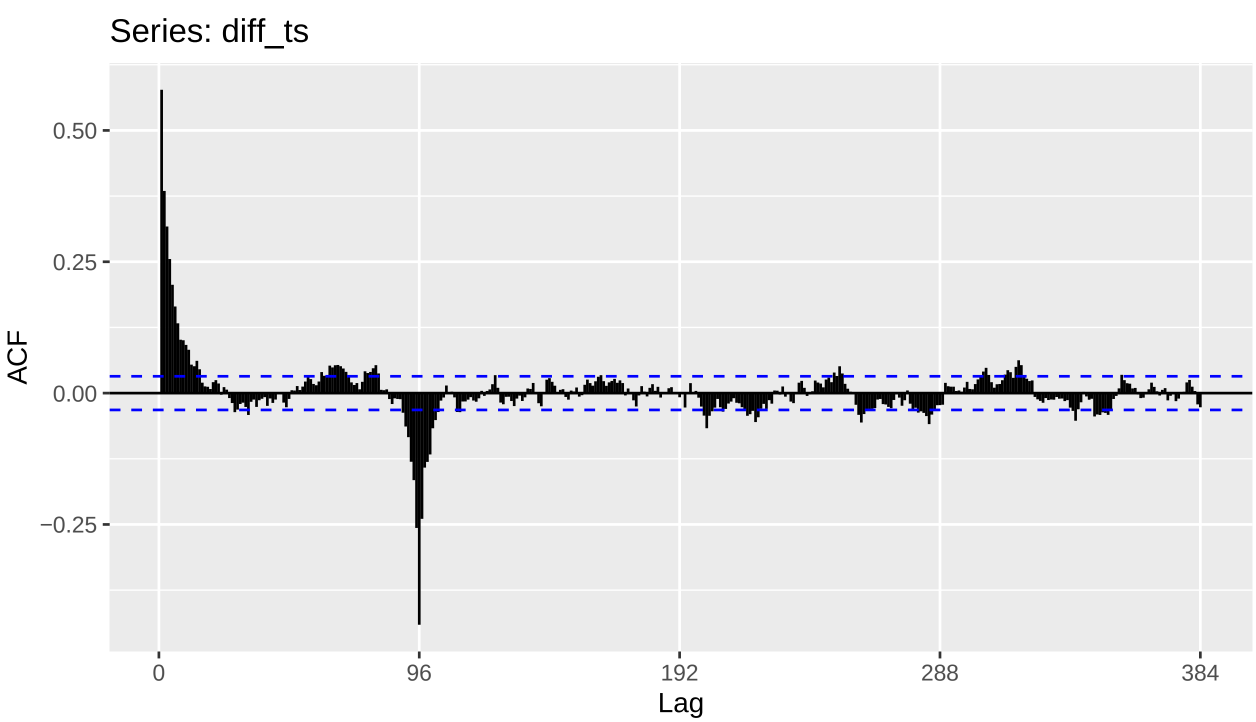
Differenciating

- Given the strong seasonality of our time series, a first seasonal differentiation is performed in order to try to obtain a stationary time series.

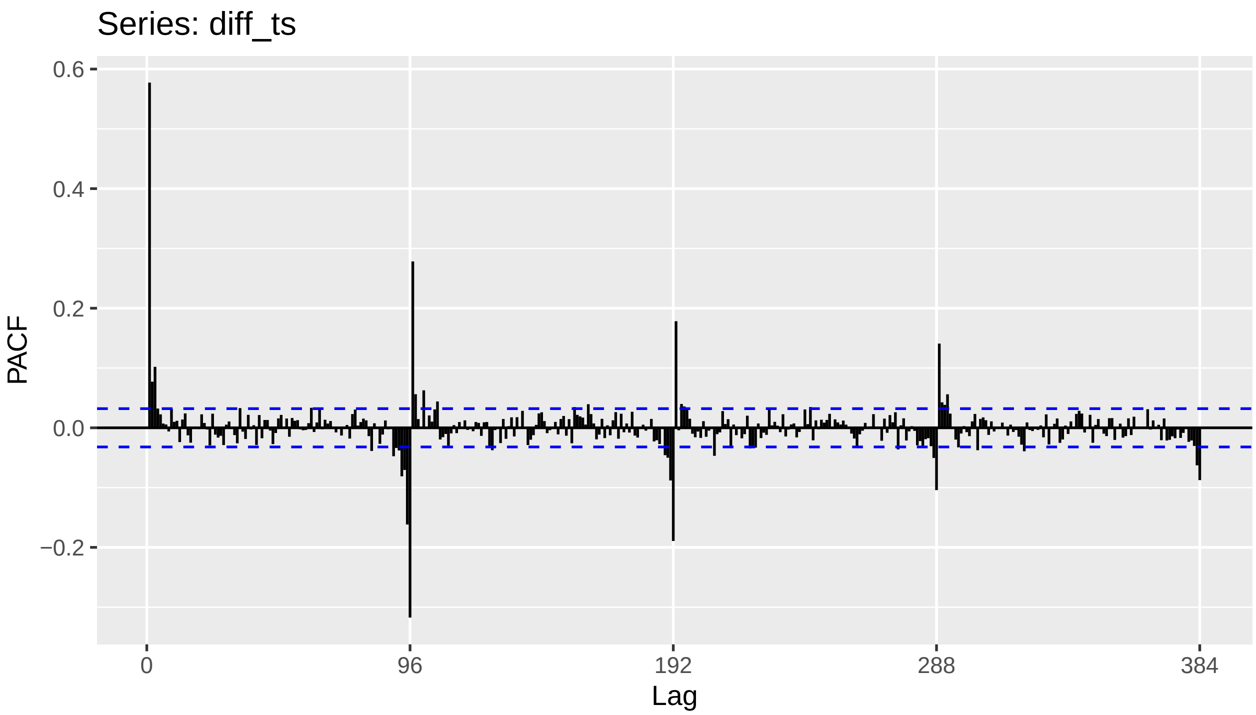
```r
diff_ts = diff(ts_train[,"consumption"], lag = 96)

plot(diff_ts, main = "Seasonal diff with leg 96")
```




```
ggAcf(diff_ts, lag = 96*4)
```




```
ggPacf(diff_ts, lag = 96*4)
```




```r
# Checking with ADF test
adf(diff_ts, criterion = "AIC")
```

```
##
##  Two-step ADF test (with intercept) on a single time series
##
## data: diff_ts
## null hypothesis: Series has a unit root
## alternative hypothesis: Series is stationary
##
##         estimate largest root statistic   p-value
## diff_ts                0.6611    -19.73 1.864e-45
```

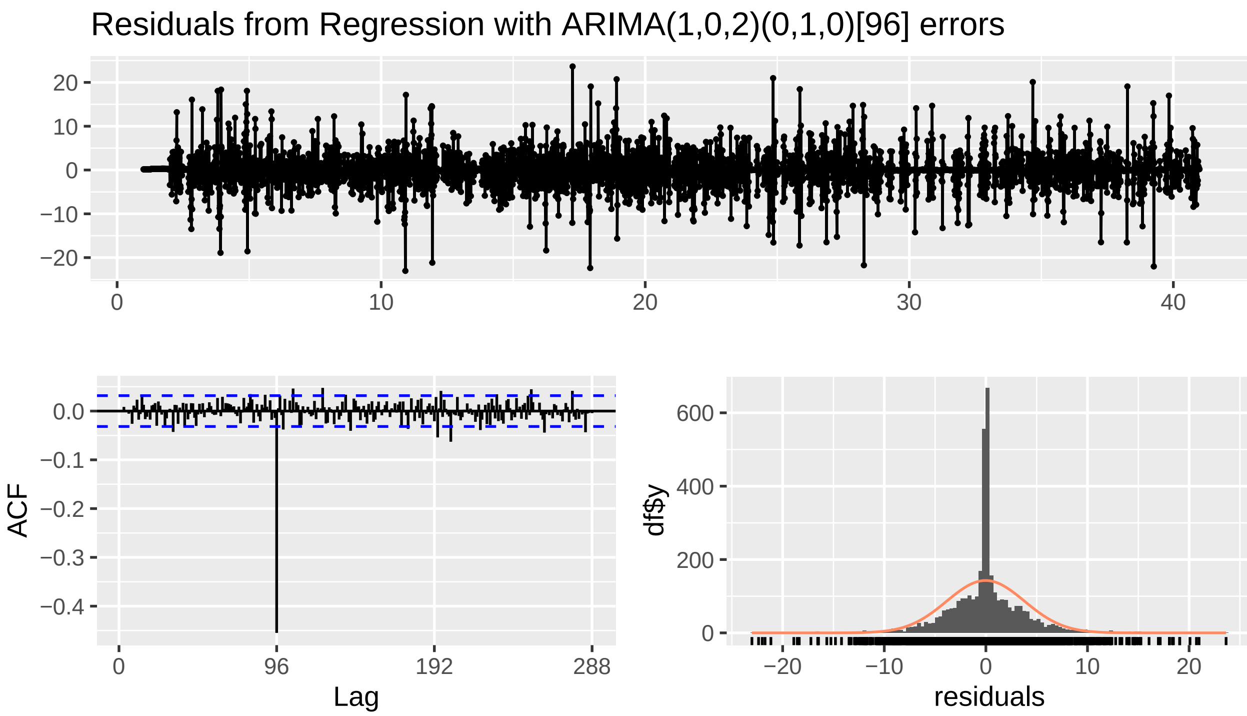
Observations:

- The series seems to be stationary despite some spikes present after applying a seasonal differentiation.
- The ADF test provides us with more statistically significant result to reject the non-stationary hypothesis (p-value < 0.01), agreeing with the visual analysis.
- We can see a quick exponential decrease in the ACF and possibly at the PACF as well, although with a slower decay.
- MA models possible orders:
  - **q:** between 0 and 15: by looking at the most significant spikes at the first few lags
  - **Q:** 1, due to the big spike at the first period 96.
- AR models possible orders:
  - **p:** between 0 and 8: bigger spike in the first few lags
  - **P:** between 1 and 4: given the recurrent decreasing spiked at each lag

Auto arima

- In order to get an insight into a possible model's parameters, auto.arima is first performed.

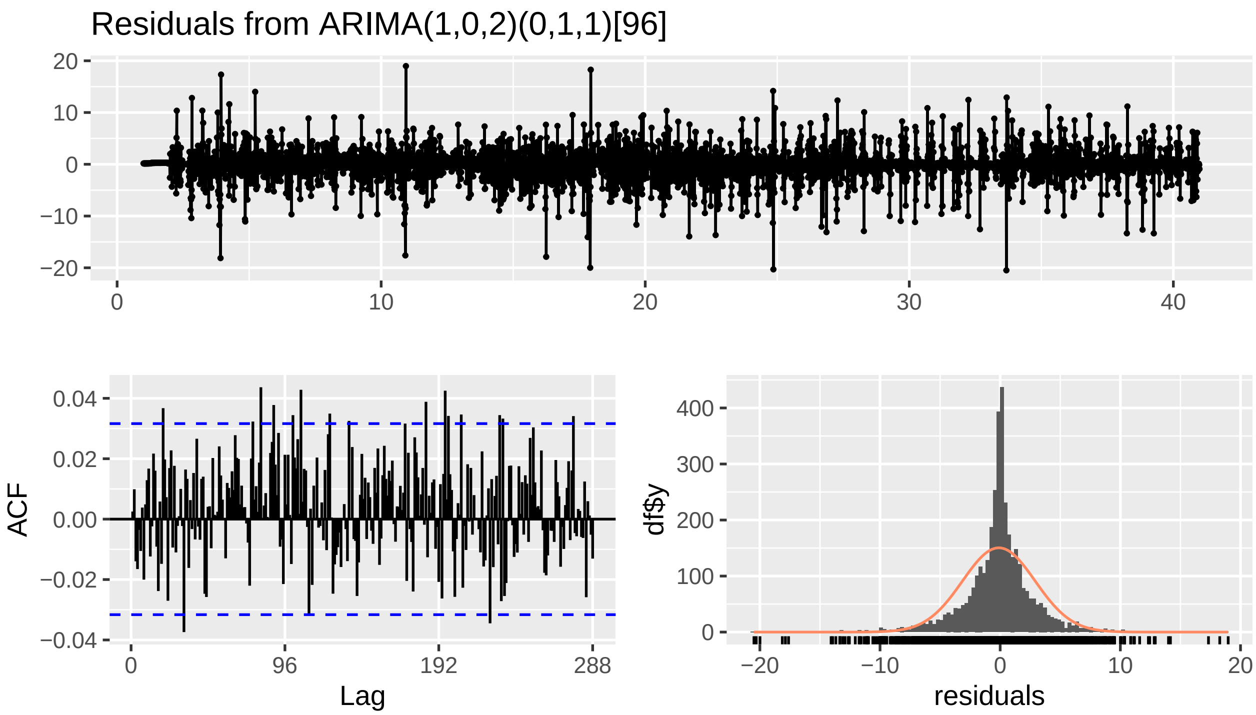
```
fit_auto_xreg=auto.arima(ts_train[,"consumption"],xreg=ts_train[,"temperature"])
checkresiduals(fit_auto_xreg)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(1,0,2)(0,1,0)[96] errors
## Q* = 1036.9, df = 189, p-value < 2.2e-16
##
## Model df: 3.   Total lags used: 192
```

- By looking at the residuals and specifically at the ACF plot, we can see a big spike at lag 96 (one period). For this reason, increasing the Q component seems appropriate to try to account for that spike.

```
improved_fit=Arima(ts_train[,"consumption"],order=c(1,0,2),seasonal=c(0,1,1))
checkresiduals(improved_fit)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,0,2)(0,1,1)[96]
## Q* = 201.38, df = 188, p-value = 0.2393
##
## Model df: 4.   Total lags used: 192
```

Observations:

- Indeed, we can see that the big spike at lag 96 has been successfully removed and we seem to have white noise as a result as indicated by the p-value > 0.05, meaning we **cannot** reject the null hypothesis of the residuals being white noise..

Improving Auto arima

- In order to try to obtain the best model, the non-seasonal hyper parameters of the Arima parameters have been tested.

```
# Data frame containing the results of the tuning parameters for SARIMA
arima_tuning_df = data.frame(p = character(), q = numeric(), error = numeric())

for(p in (1:5)){
  for(q in (2:5)){
    # Train current model
    tuning_arima_fit = Arima(ts_train[,"consumption"], order = c(p,0,q),
                             seasonal = c(0,1,1))

    # Check residuals
```

```r
    residuals = checkresiduals(tuning_arima_fit)

    # Save results if p-value > 0.05
    if(residuals$p.value > 0.05)
    {
      # Forecast
      pred = forecast(tuning_arima_fit, h = test_horizon)$mean

      # Calculate the error
      mse = mse_error(pred, ts_test[,"consumption"])

      # Save results in the data frame
      arima_tuning_df = rbind(arima_tuning_df, data.frame(p = p,
                                                          q = q, error = mse))
    }

  }
}
```

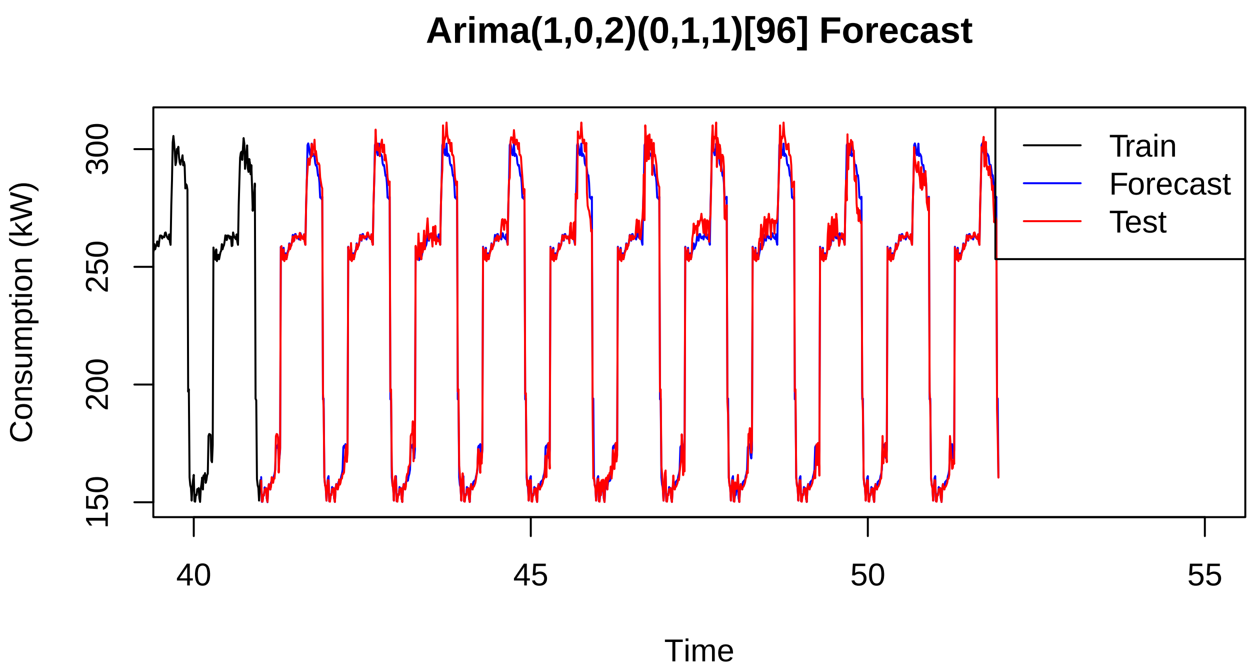
```r
head(arima_tuning_df[order(arima_tuning_df$error), ])
```

```
##   p q    error
## 1 5 4 4.665866
## 2 5 5 4.666776
## 3 1 2 4.666840
## 4 1 3 4.666876
## 5 2 2 4.666892
## 6 1 4 4.666905
```

Observations:

- By looking at the comparison between different orders of the non-seasonal part of the SARIMA, we can see that there is no significant improvement. Therefore, we keep the simplest model that achieved a good result in this experiment: ARIMA(1,0,2)(0,1,1)[96]

```r
# Saving error of best SARIMA model (ARIMA(1,0,2)(0,1,1)[96]).
pred = forecast(improved_fit, h = test_horizon)$mean

# Calculate the error
mse = mse_error(pred, ts_test[,"consumption"])

errors_df = rbind(errors_df, data.frame(method = "Arima(1,0,2)(0,1,1)[96]",
                                        covariate = FALSE,
                                        MSE = mse))
# Plot Forecast
plot_forecast(train = ts_train[,"consumption"], test = ts_test[,"consumption"],
pred=pred,xlim=c(40,55),title="Arima(1,0,2)(0,1,1)[96] Forecast")
```

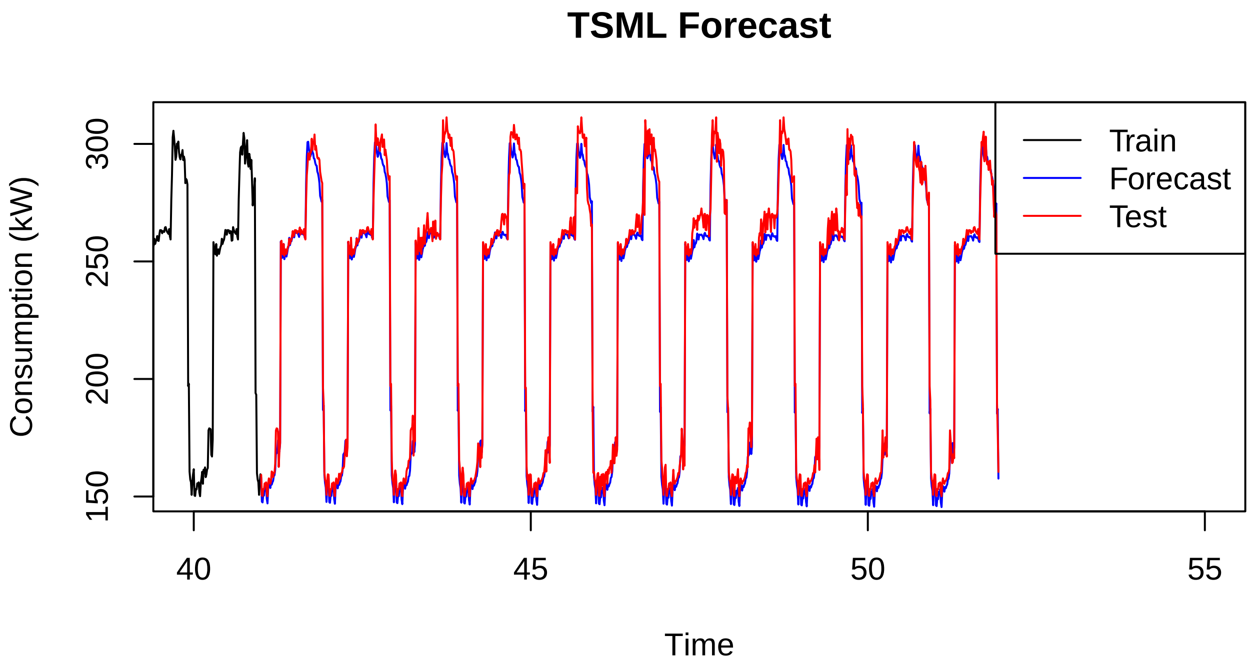


Time Series Linear Model (TSLM)

```
# Fit
tslm_fit = tslm(consumption~trend+season,data= ts_train)

# Forecast
pred = forecast(tslm_fit, h = test_horizon)$mean

#Calculate error
mse =  mse_error(pred, ts_test[,"consumption"])
cat("tsml MSE: ", mse, "\n")
```

```
## tsml MSE:  5.722733
```

```
errors_df = rbind(errors_df, data.frame(method = "TSLM",covariate = FALSE,
                                        MSE = mse))

plot_forecast(train = ts_train[,"consumption"], test = ts_test[,"consumption"],
              pred = pred, title = "TSML Forecast")
```



Machine Learning Models

Support functions

```
# Forecast function for machine learning models
predMl = function(h, freq, pred_start,train, model, matrix = FALSE)
{
  pred = rep(NULL, h)

  # The first prediction will be the last row of  the training data set.
  newdata = t(tail(train, freq))
  for (t in 1:(h)){

    # To deal with the cases where we need to have a matrix for prediction
    if(matrix == TRUE)
    {
      newdata = matrix(newdata,1,freq)
    }
    pred[t] = predict(model, newdata = newdata)

    # The following element forecast will be from the second element until
    #the previously predicted value (and so on)

    newdata = c(newdata[-1],pred[t])

  }

  # Finally, we transform the predictions into a Time-Series again
  pred_ts = ts(pred, start = pred_start, freq = freq)

  return(pred_ts)
```

```
}
```

Transforming into ML compatible dataset

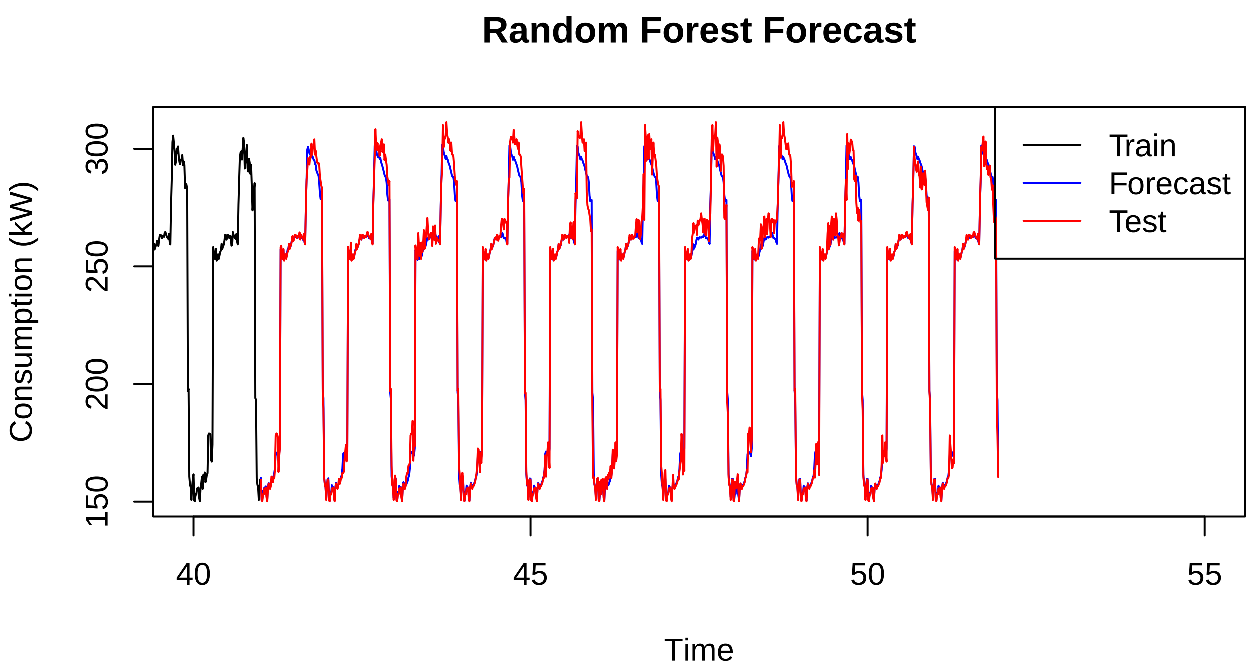
```
# We transform everything in vector
ts_vector = as.vector(ts_train[,"consumption"])

# Get the first row (in vector form) to start the iteration
ts_ml = ts_vector[1:97]

# For each element of t he vector, generate a new
# row with a response columns added
for(i in 1:(length(ts_vector)-97))
{
  ts_ml = rbind(ts_ml, as.vector(ts_vector[(i+1):(i+97)]))
}
```

Random Forest

```
library(randomForest)

# Fit
rf_fit = randomForest(x = ts_ml[,-97], y=ts_ml[,97])

# Forecast
pred = predMl(h = test_horizon, freq = 96, pred_start = c(41,1),
              train = ts_train[,"consumption"], model = rf_fit)

# Calculate and store error
mse = mse_error(pred, ts_test[,"consumption"])
cat("Random Forest MSE: ", mse)
```

```
## Random Forest MSE:  4.939105
```

```
errors_df = rbind(errors_df, data.frame(method = "Random Forest",
                                        covariate = FALSE,
                                        MSE = mse))

# Plot forecast
plot_forecast(train = ts_train[,"consumption"], test = ts_test[,"consumption"],
              pred = pred, title = "Random Forest Forecast")
```




```r
# Check Residuals
check_residuals(ypred = pred, ytrue = ts_test[,"consumption"], freq = 96)
```

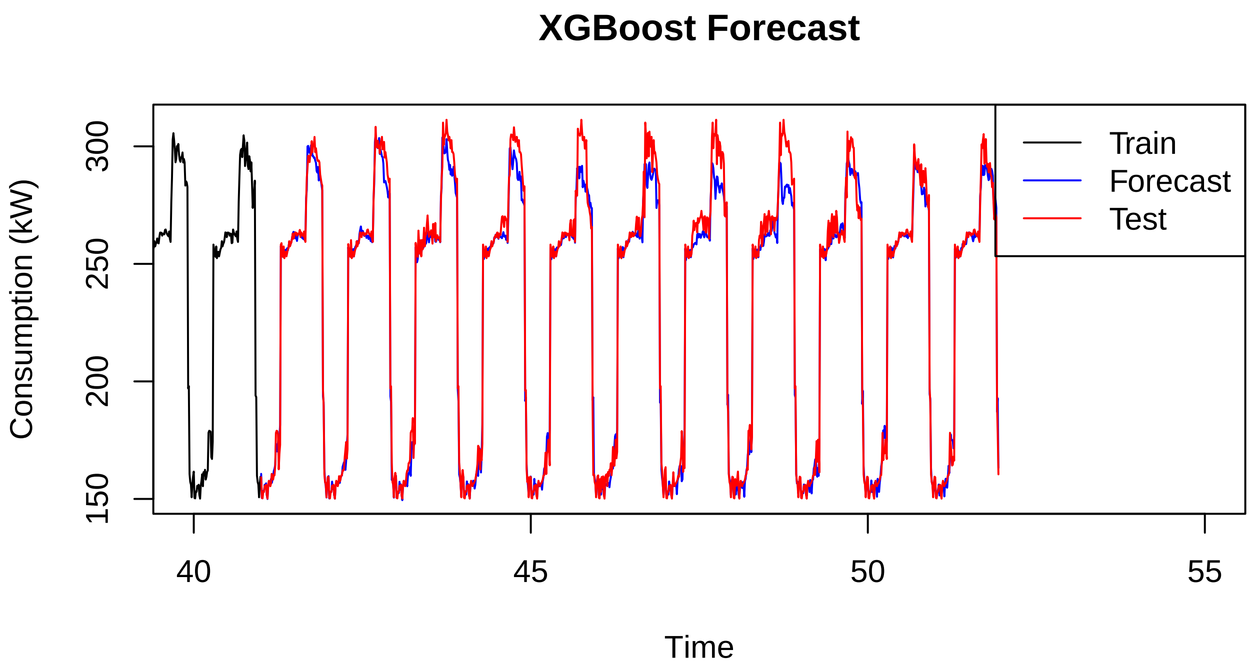
```
##
##  Box-Ljung test
##
## data:  ts_res
## X-squared = 289.94, df = 1, p-value < 2.2e-16
```

XGBoost

```r
library(xgboost)

# Fit
xg_fit = xgboost(data = ts_ml[,-97], label = ts_ml[,97],
                                        max_depth = 10,
                                        eta = .5,
                                        nrounds = 50,
                                        nthread = 2,
                                        objective = "reg:squarederror",
                                        verbose = FALSE)

# Forecast
pred = predMl(h = test_horizon, freq = 96, pred_start = c(41,1),
              train = ts_train[,"consumption"], model = xg_fit, matrix = TRUE)

# Calculate and store error
mse = mse_error(pred, ts_test[,"consumption"])
cat("XGBoost MSE: ", mse)
```

```
## XGBoost MSE:  6.624767
```

```r
errors_df = rbind(errors_df, data.frame(
                                    method = "XGBoost",
                                    covariate = FALSE,
                                    MSE = mse))

# Plotting Forecast
plot_forecast(train = ts_train[,"consumption"], test = ts_test[,"consumption"],
              pred = pred, title = "XGBoost Forecast")
```




```r
# Check residuals
check_residuals(ypred = pred, ytrue = ts_test[,"consumption"], freq = 96)
```

```
##
##  Box-Ljung test
##
## data:  ts_res
## X-squared = 369.69, df = 1, p-value < 2.2e-16
```

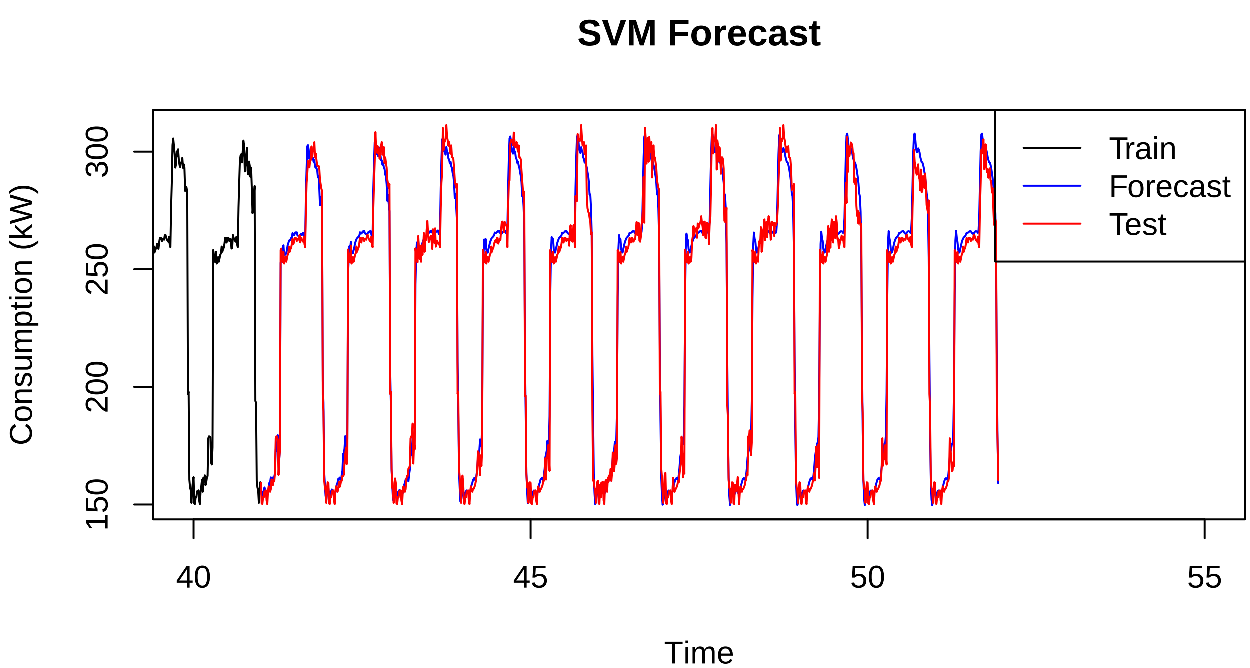
SVM

```r
library(e1071)

# Fit
svm_fit = svm(x = ts_ml[,-97], y = ts_ml[,97])

# Forecast
pred = predMl(h = test_horizon, freq = 96, pred_start = c(41,1),
              train = ts_train[,"consumption"], model = svm_fit, matrix = TRUE)
```

```r
# Calculate and store error
mse = mse_error(pred, ts_test[,"consumption"])
cat("SVM MSE: ", mse)
```

```
## SVM MSE:  6.676689
```

```r
errors_df = rbind(errors_df, data.frame(
                                        method = "SVM",
                                        covariate = FALSE,
                                        MSE = mse))

# Plotting Forecast
plot_forecast(train = ts_train[,"consumption"], test = ts_test[,"consumption"],
              pred = pred, title = "SVM Forecast")
```




```r
# Check residuals
check_residuals(ypred = pred, ytrue = ts_test[,"consumption"], freq = 96)
```

```
##
##  Box-Ljung test
##
## data:  ts_res
## X-squared = 58.344, df = 1, p-value = 2.198e-14
```

Neural Netwrok

```r
# Fit
nn_fit = nnetar(ts_train[,"consumption"], size = 35, p = 10, P = 2)
```

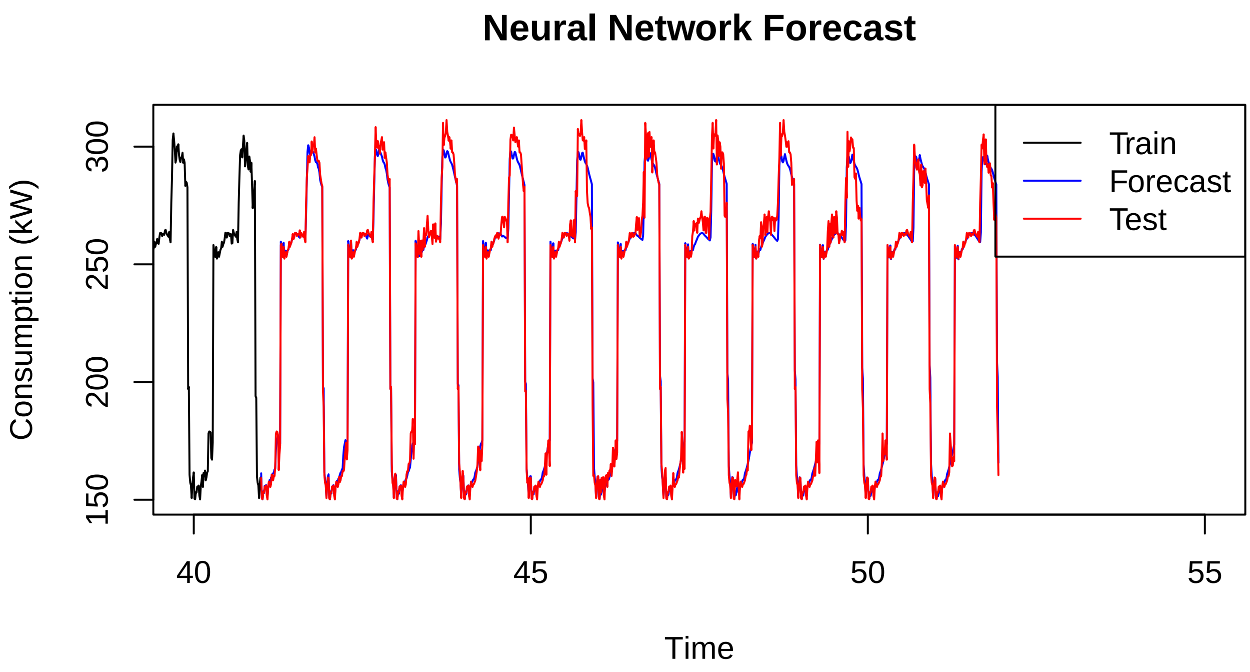
```r
# Forecast
pred = forecast(nn_fit, h = test_horizon)$mean

# Calculate and store error
mse = mse_error(pred, ts_test[,"consumption"])
cat("MSE Neural Networks: ", mse)
```

```
## MSE Neural Networks:  5.656255
```

```r
errors_df = rbind(errors_df, data.frame(
                                        method = "Neural Networks",
                                        covariate = FALSE,
                                        MSE = mse))

# Check residuals
check_residuals(pred, ts_test[,"consumption"],freq = 96)
```

```
##
##  Box-Ljung test
##
## data:  ts_res
## X-squared = 374.56, df = 1, p-value < 2.2e-16
```

```r
# Plot Forecast
plot_forecast(train = ts_train[,"consumption"], test = ts_test[,"consumption"],
              pred = pred, xlim = c(40, 55), title = "Neural Network Forecast")
```



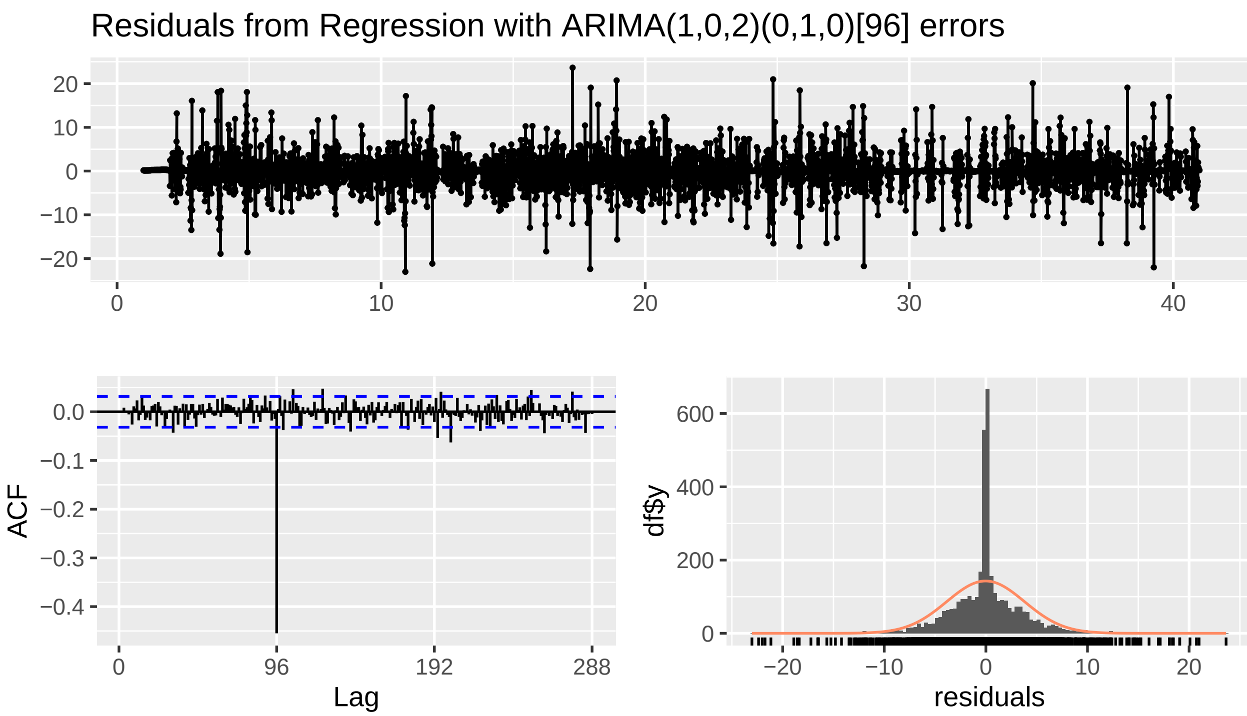
With Temperature Covariate

Sarima

Auto.arima

- Once again, we first test with auto.arima.

```
fit_auto_xreg = auto.arima(ts_train[,"consumption"],
                           xreg = ts_train[,"temperature"])
checkresiduals(fit_auto_xreg)
```



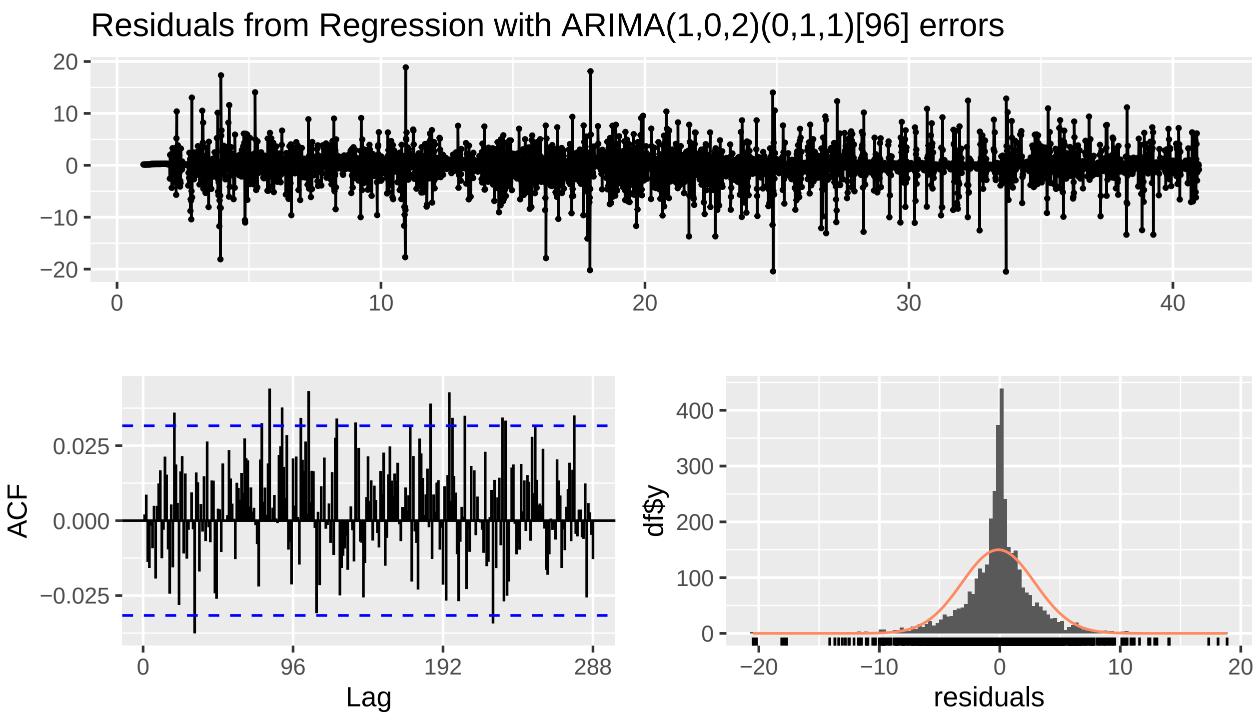

```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(1,0,2)(0,1,0)[96] errors
## Q* = 1036.9, df = 189, p-value < 2.2e-16
##
## Model df: 3.   Total lags used: 192
```

```
# Fit
arima_fit_xreg = Arima(ts_train[,"consumption"], order = c(1,0,2),
                       seasonal = c(0,1,1), xreg = ts_train[,"temperature"])

# Forecast
pred = forecast(arima_fit_xreg, h = test_horizon,
                xreg = ts_test[,"temperature"])$mean

# Calculate and store error
mse = mse_error(pred, ts_test[,"consumption"])
cat("MSE Arima(1,0,2)(0,1,1) XREG: ", mse)
```

Improved Auto.arima

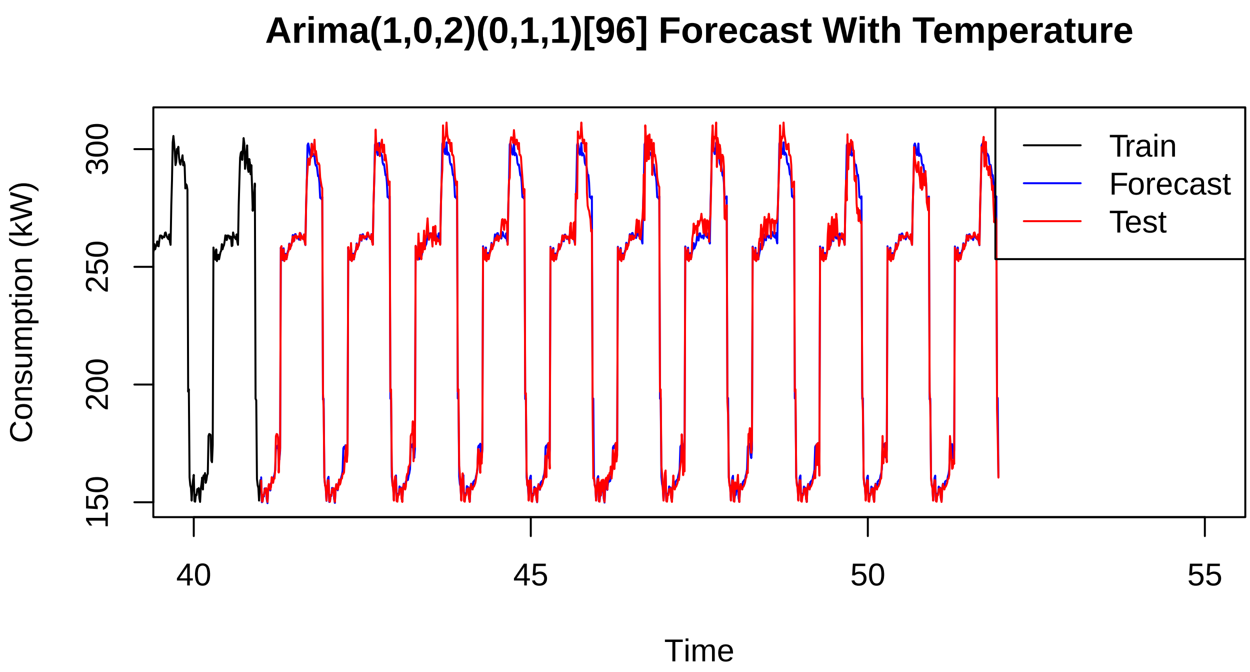
```
## MSE Arima(1,0,2)(0,1,1) XREG:  4.605829
```

```
errors_df = rbind(errors_df, data.frame(method = "Arima(1,0,2)(0,1,1)[96]",
                                        covariate = TRUE,
                                        MSE = mse))

# Check residuals
checkresiduals(arima_fit_xreg)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(1,0,2)(0,1,1)[96] errors
## Q* = 200.56, df = 188, p-value = 0.252
##
## Model df: 4.   Total lags used: 192
```

```
# Plot Forecast
plot_forecast(train = ts_train[,"consumption"], test = ts_test[,"consumption"],
pred=pred,xlim=c(40,55),title="Arima(1,0,2)(0,1,1)[96] Forecast With Temperature")
```



Machine Learning Models

- Support function to predict with covariates

```
predMl_xreg = function(h, freq, pred_start,train,
                       xreg_data, model, matrix = FALSE)
{
  pred = rep(NULL, h)
  # The first prediction will be the last row of  the training data set.
  newdata = c(tail(train, freq), xreg_data[1])

  for (t in 1:(h)){

    # To deal with the cases where we need to have a matrix for prediction
    if(matrix == TRUE)
    {
      newdata = matrix(newdata,1,freq+1) #+1 to account for the xreg
    }
    pred[t] = predict(model, newdata = newdata)

    # The following element forecast will be from the second element until
    # the previously predicted value (and so on)
    if(t+1 <= length(xreg_data)){
      # Remove the first and last (temperature) element
      newdata = newdata[-c(1, length(newdata))]
      # Add new prediction and new future temperature value
      newdata = c(newdata, pred[t], xreg_data[t+1])
    }
  }

  # Finally, we transform the predictions into a Time-Series again
  pred_ts = ts(pred, start = pred_start, freq = freq)
```

```
  return(pred_ts)

}
```

Data Transformation into ML data set.

- This time, we add the temperature as the 97th variable.
- The 98th variable is the X_T+1.

```
vec_train = as.vector(ts_train[,"consumption"])
vec_xreg = as.vector(ts_train[,"temperature"])

ts_ml_xreg = c(vec_train[1:97], vec_xreg[96])

# Swapping columns, placing the response variable as the last column]
swap = vec_xreg[96]
ts_ml_xreg[98] =  ts_ml_xreg[97]
ts_ml_xreg[97] = swap

for(i in 1:(length(vec_train)-97))
{
    newdata = c(vec_train[(i+1):(i+97)],vec_xreg[i+96])
    swap = vec_xreg[i+96]
    newdata[98] = newdata[97]
    newdata[97] = swap

    ts_ml_xreg = rbind(ts_ml_xreg, newdata)
}
```

Random Forest

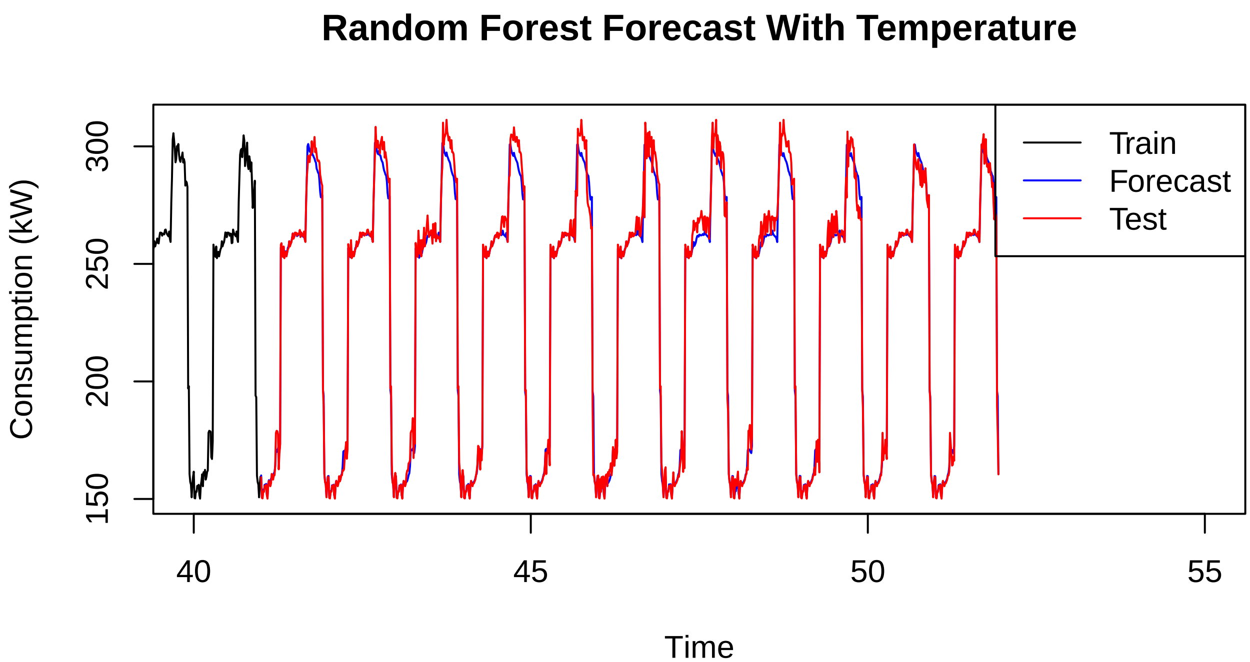
```
# Fit
rf_fit_xreg = randomForest(x = ts_ml_xreg[,-98],
                           y=ts_ml_xreg[,98], ntree = 1000)

# Forecast
pred = predMl_xreg(h = test_horizon, freq = 96, pred_start = c(41,1),
                    train = ts_train[,"consumption"],
                   xreg_data = ts_test[,"temperature"], model = rf_fit_xreg)

# Calculate and store error
mse = mse_error(pred, ts_test[,"consumption"])
cat("Random Forest with XREG MSE: ", mse)
```

```
## Random Forest with XREG MSE:  4.928153
```

```
errors_df = rbind(errors_df, data.frame(method = "Random Forest XREG",
                                        covariate = TRUE,
                                        MSE = mse))
# Check residuals
check_residuals(pred, ts_test[,"consumption"],freq = 96)
```

```
##
##  Box-Ljung test
##
## data:  ts_res
## X-squared = 282.67, df = 1, p-value < 2.2e-16
```

```r
# Plot Forecast
plot_forecast(train = ts_train[,"consumption"], test = ts_test[,"consumption"],
              pred = pred, title = "Random Forest Forecast With Temperature")
```



XGBoost

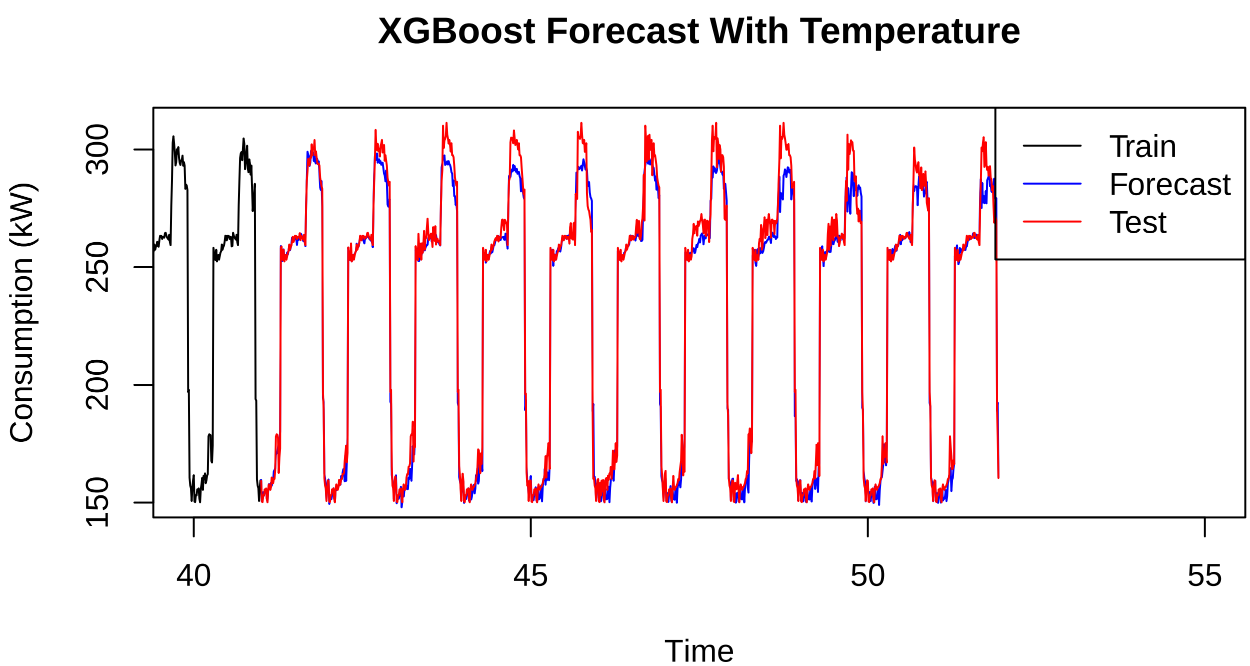
```r
# Fit
xg_fit_xreg = xgboost(data = ts_ml_xreg[,-98], label = ts_ml_xreg[,98],
                                          max_depth = 10,
                                          eta = .5, nrounds = 50,
                                          nthread = 2,
                                          objective = "reg:squarederror",
                                          verbose = FALSE)
# Forecast
pred = predMl_xreg(h = test_horizon, freq = 96, pred_start = c(41,1),
          train = ts_train[,"consumption"], xreg_data = ts_test[,"temperature"],
          model = xg_fit_xreg, matrix = TRUE)

# Error
mse = mse_error(pred, ts_test[,"consumption"])
cat("XGBoost XREG MSE: ", mse)
```

```
## XGBoost XREG MSE:  6.664749
```

```
errors_df = rbind(errors_df, data.frame(
                                        method = "XGBoost XREG",
                                        covariate = TRUE,
                                        MSE = mse))

# Check Residuals
check_residuals(ypred = pred, ytrue = ts_test[,"consumption"], freq = 96)
```

```
##
##  Box-Ljung test
##
## data:  ts_res
## X-squared = 332.16, df = 1, p-value < 2.2e-16
```

```
# Plot Forecast
plot_forecast(train = ts_train[,"consumption"], test = ts_test[,"consumption"],
              pred = pred, title = "XGBoost Forecast With Temperature")
```



SVM

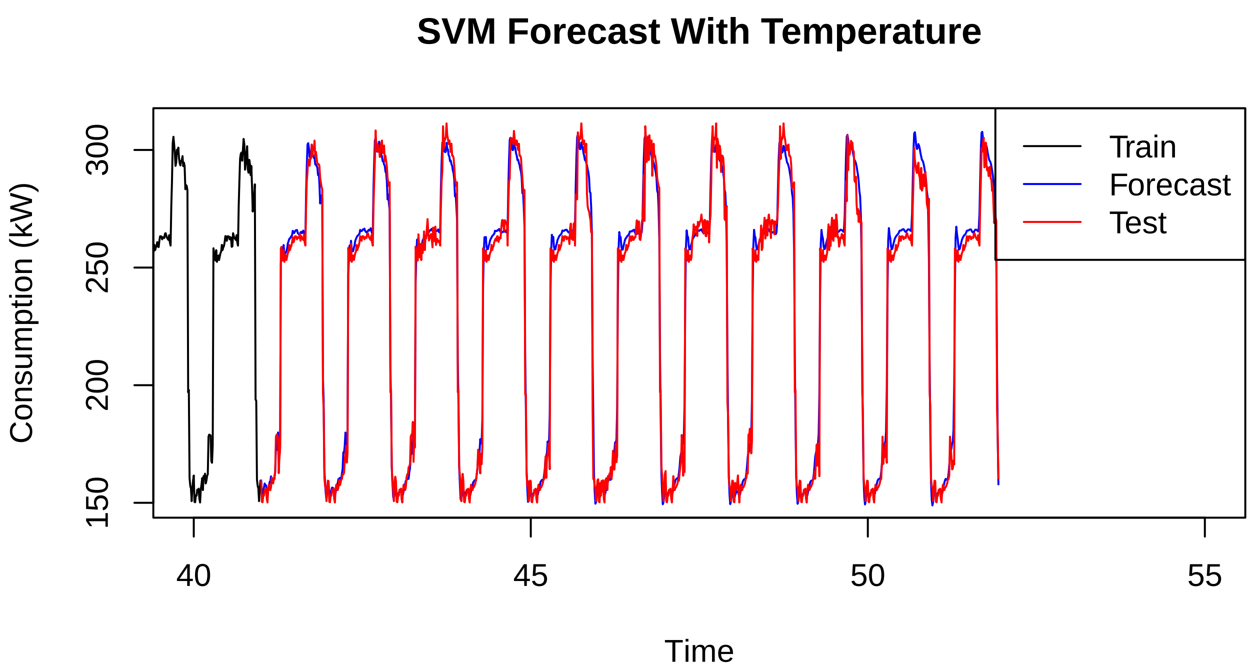
```
# Fit
svm_fit_xreg = svm(x = ts_ml_xreg[,-98], y = ts_ml_xreg[,98])
# Forecast
pred = predMl_xreg(h = test_horizon, freq = 96, pred_start = c(41,1),
          train = ts_train[,"consumption"], xreg_data = ts_test[,"temperature"],
          model = svm_fit_xreg, matrix = TRUE)

# Error
mse = mse_error(pred, ts_test[,"consumption"])
cat("SVM XREG MSE: ", mse)
```

```
## SVM XREG MSE:  6.693663
```

```r
errors_df = rbind(errors_df, data.frame(
                                        method = "SVM XREG",
                                        covariate = TRUE,
                                        MSE = mse))

# Check Residuals
check_residuals(ypred = pred, ytrue = ts_test[,"consumption"], freq = 96)
```

```
##
##  Box-Ljung test
##
## data:  ts_res
## X-squared = 62.497, df = 1, p-value = 2.665e-15
```

```r
# Plot Forecast
plot_forecast(train = ts_train[,"consumption"], test = ts_test[,"consumption"],
              pred = pred, title = "SVM Forecast With Temperature")
```



Neural Netwrok

```r
# Fit
nn_fit_xreg = nnetar(ts_train[,"consumption"], size = 35, p = 10, P = 2,
                     xreg = ts_train[,"temperature"])


# Forecast
pred = forecast(nn_fit_xreg, h = test_horizon,
```

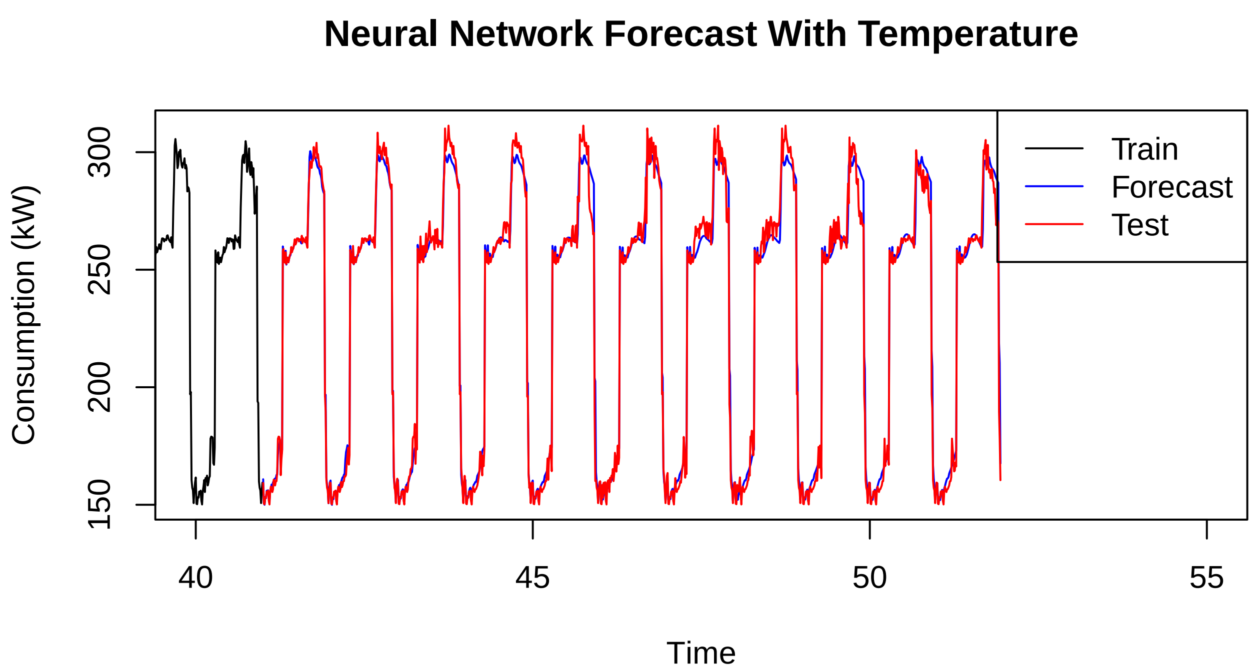
```r
                 xreg = ts_test[,"temperature"])$mean

# Calculate and store error
mse = mse_error(pred, ts_test[,"consumption"])
cat("MSE Neural Networks: ", mse)
```

```
## MSE Neural Networks:  5.897596
```

```r
errors_df = rbind(errors_df, data.frame(
                                        method = "Neural Networks",
                                        covariate = TRUE,
                                        MSE = mse))

# Check residuals
check_residuals(pred, ts_test[,"consumption"],freq = 96)
```

```
##
##  Box-Ljung test
##
## data:  ts_res
## X-squared = 412.91, df = 1, p-value < 2.2e-16
```

```r
# Plot Forecast
plot_forecast(train = ts_train[,"consumption"], test = ts_test[,"consumption"],
pred = pred, xlim = c(40, 55), title="Neural Network Forecast With Temperature")
```



4. Model Selection

- By looking at the MSE, we can conclude that the best models are the SARIMA model for both with and without covariate, with 4.60 using covariate and 4.66 without covariate approximately.

- In addition, the SARIMA models were the only ones, as shown in the Box-Ljung test performed above for each model, that managed to better capture the time series characteristics leaving only white residuals in the end.

```
sorted_error = errors_df[order(errors_df$MSE), ]
sorted_error
```

```
##                      method covariate       MSE
## 8  Arima(1,0,2)(0,1,1)[96]      TRUE  4.605829
## 2  Arima(1,0,2)(0,1,1)[96]     FALSE  4.666840
## 9        Random Forest XREG      TRUE  4.928153
## 4             Random Forest     FALSE  4.939105
## 7           Neural Networks     FALSE  5.656255
## 3                      TSLM     FALSE  5.722733
## 12          Neural Networks      TRUE  5.897596
## 5                   XGBoost     FALSE  6.624767
## 10             XGBoost XREG      TRUE  6.664749
## 6                       SVM     FALSE  6.676689
## 11                 SVM XREG      TRUE  6.693663
## 1                       SES     FALSE 91.865189
```

5. Cross-Validation

- To detect possible over fitting in our models, we try now cross validation with the very best model (Arima(1,0,2)(0,1,1)[96] with temperature)

```
train_splits = list(c(35,96),c(37,96),c(39,96),c(42,96),c(45,96),c(47,96))
test_splits = list(c(36,1),c(38,1),c(40,1),c(43,1),c(46,1),c(48,1))
splits = 6

error = c()

for (i in (1 : splits)){

    # New Split
    ts_cv_train = window(ts, end = train_splits[[i]])
    ts_cv_test = window(ts, start = test_splits[[i]])

    # Train
    fit_cv = Arima(ts_cv_train[,"consumption"], order = c(1,0,2),
                   seasonal = c(0,1,1), xreg = ts_cv_train[,"temperature"])

    # Forecast
    pred = forecast(fit_cv, h = length(ts_cv_test[,"consumption"]),
                    xreg = ts_cv_test[,"temperature"])

    # Calculate error
    mse = mse_error(pred$mean, ts_cv_test[,"consumption"])
    error = rbind(error, mse)
}
```

```r
cat("CV Average:", mean(error))
```

```
## CV Average: 4.625495
```

- By looking at the results of the cross validation being very close to the one of our model in our initial train/test split, we have strong indications that our model did not overfit and seems to be an appropriate model for the proposed problem.

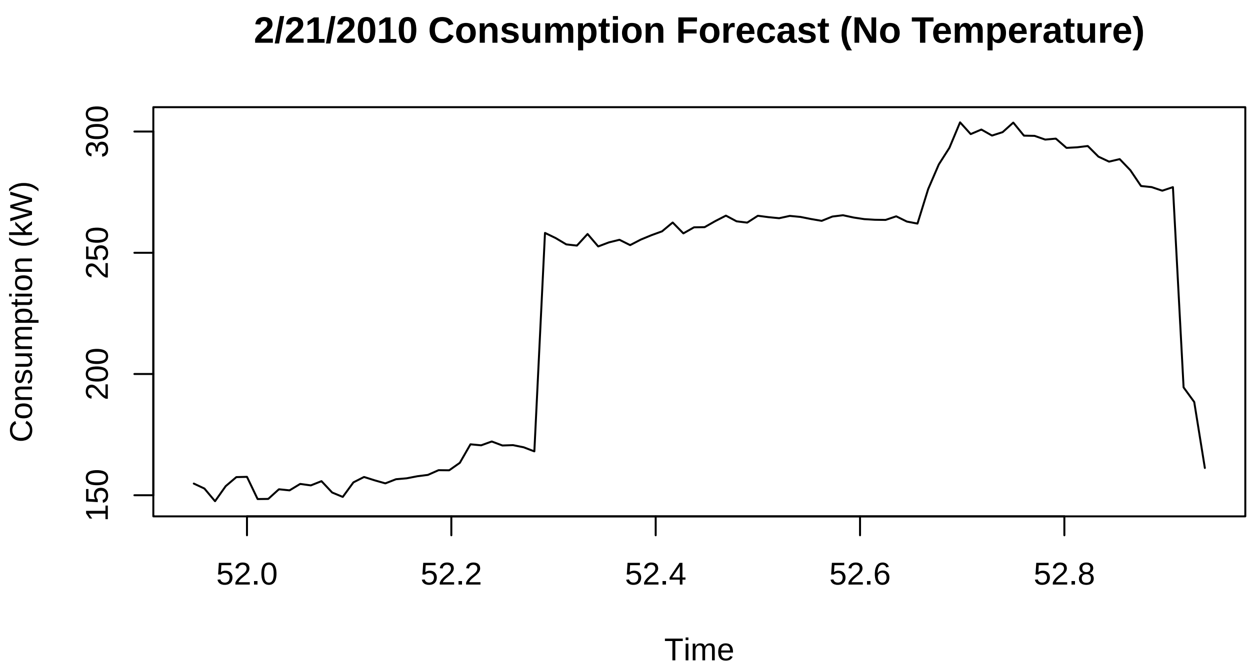
6. Best Model Training

- Training the best models in the whole dataset

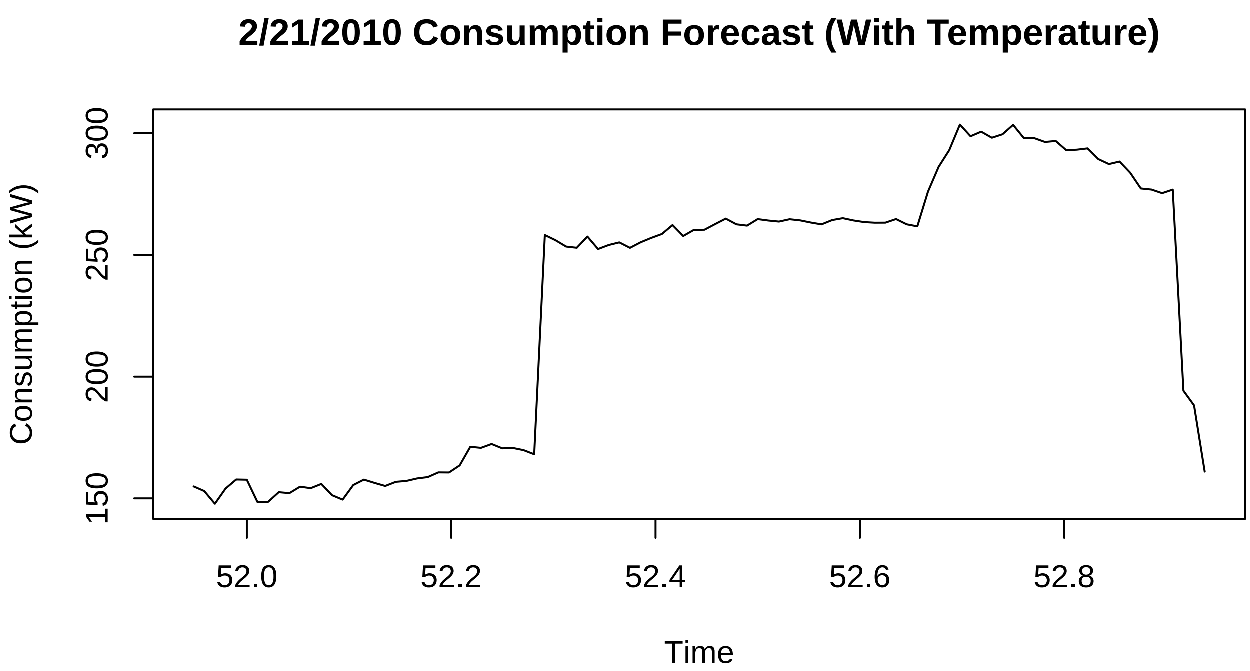
```r
# Avoid warning for different column names
xreg = as.matrix(ts[,"temperature"])
colnames(xreg) = c("temperature")

# With temperature
  best_model_xreg = Arima(ts[,"consumption"], order = c(1,0,2),
                          seasonal = c(0,1,1), xreg = xreg)

# Without temperature
best_model = Arima(ts[,"consumption"], order = c(1,0,2), seasonal = c(0,1,1))
```

7. Forecasting

```r
xreg = as.matrix(ts_future_temp)

colnames(xreg) = c("temperature")

final_pred = forecast(best_model, h = 96)$mean
final_pred_xreg = forecast(best_model_xreg, xreg = xreg)$mean

forescasts = data.frame(
    withoutTemp = final_pred,
    withTemp = final_pred_xreg
  )


plot(final_pred, main = "2/21/2010 Consumption Forecast (No Temperature)",
     ylab ="Consumption (kW)")
```




```r
plot(final_pred_xreg, main="2/21/2010 Consumption Forecast (With Temperature)",
     ylab = "Consumption (kW)")
```



8. Conclusion

- The best model among all of the models trained was a SARIMA model with covariate temperature.
- For all the models that were not SARIMA, the p-value was always very small (< 0.05), indicating that they could not really capture very well the time-series. For that reason, even though they MSE were sometimes quite close to the SARIMA, the later was the one that best modeled and generalized our time-series.