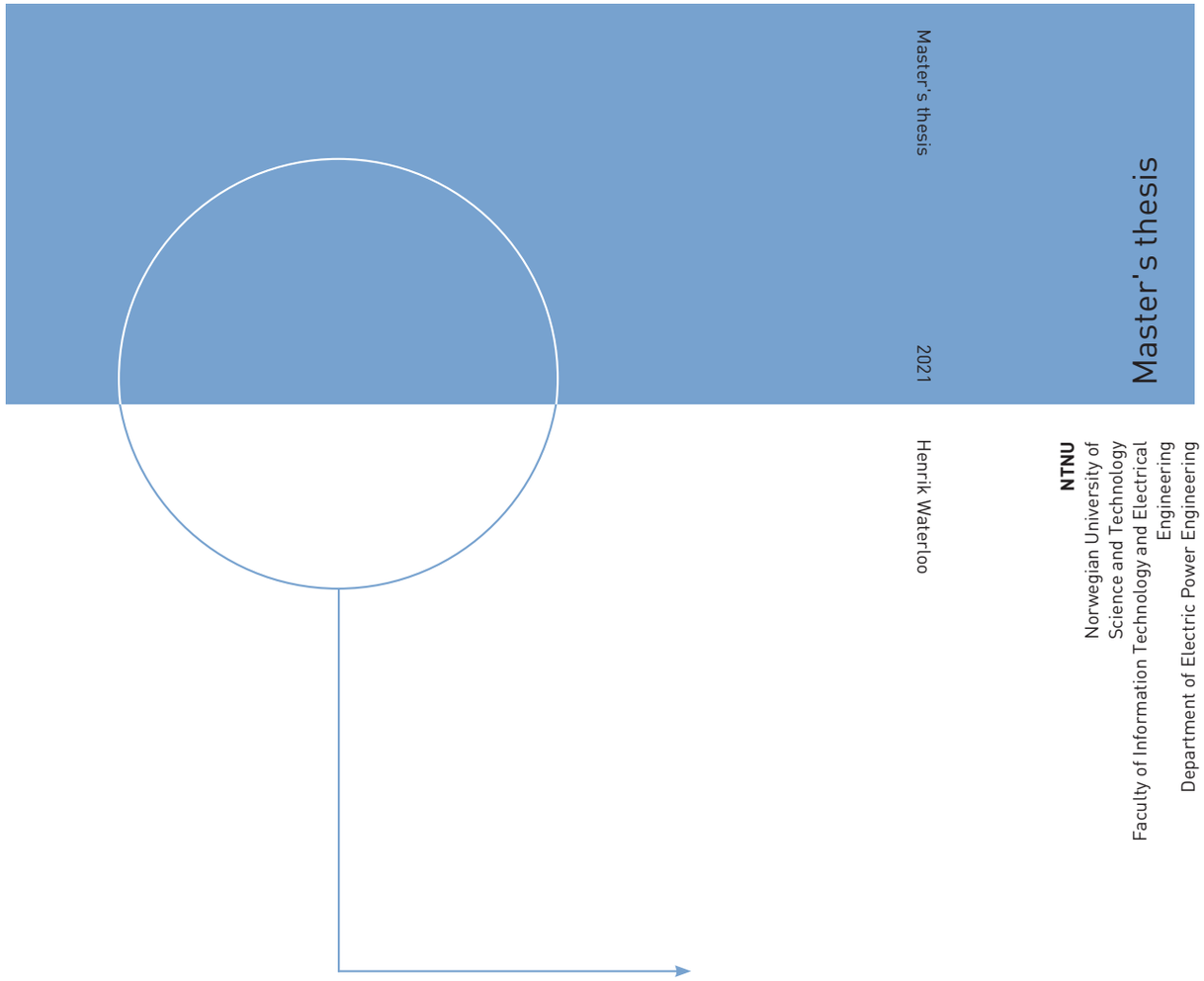


Master's thesis

2021

Henrik Waterloo

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electric Power Engineering

Henrik Waterloo

Predicting Domestic Hot Water Consumption in Buildings in Norway Using Machine Learning

June 2021

NTNU
Norwegian University of Science and Technology

NTNU
Norwegian University of Science and Technology

NTNU
Norwegian University of
Science and Technology

Predicting Domestic Hot Water Consumption in Buildings in Norway Using Machine Learning

Henrik Waterloo

Energy and Environmental Engineering
Submission date: June 2021
Supervisor: Karen Byskov Lindberg

Norwegian University of Science and Technology
Department of Electric Power Engineering

Preface

This thesis is the result of a master's project given in the 5th year of a master's degree in Energy and Environment, civil engineering. The specific task was designed by the Department of Electric Power Engineering at NTNU in Trondheim. The thesis is written in collaboration with FME ZEN, [1], and SINTEF Community, [2], in the spring of 2021 with Karen Byskov Lindberg as main supervisor. The thesis is a natural continuum of the work done in my specialization project in the fall of 2020 which can be found in the reference list [3], however, it is not published so it is not publicly accessible.

Thank you

- Harald Taxt Walnum at SINTEF for providing data and excellent help in Python data handling.

- Jayaprakash Rajasekharan, Associate Professor at the Department of Electric Power Engineering (IEL) at NTNU, for guidance in machine learning theory and especially Microsoft Azure Machine Learning Studios

Summary

This thesis explores how machine learning techniques can be used for medium-term domestic hot water load prediction. A total of six models have been trained, fitted and validated with DHW consumption data from the Varmtvann2030 data set. The data set contains consumption data for 4 apartment buildings, 4 hotels and 4 nursing homes. The six models are the sum of Prophet and XGBoost models for each building type. The XGBoost model is a mathematical optimization process that uses regression tree gradient boosting to minimize the prediction error. The Prophet model is an additive model consisting of a trend component, a Fourier series to fit seasonality in the data, and a holiday component to adjust for holidays. The theory behind the two models is thoroughly explained in this thesis. Predictions on unseen test data are performed for all six models, and the results are displayed and compared for the three building categories.

The correlation between the DHW load, the area of the building and the number of units in the building are discussed and investigated through decomposition of the XGBoost- and Prophet models. A set of hyperparameters were tuned for the Prophet models both manually and through cross validation. These hyperparameters regulates the fitting of the trend- and seasonal components in the Prophet model.

The apartment building DHW load was predicted by a Prophet model and a XGBoost model with a Mean Absolute Percentage Error(MAPE) of $\approx 32\%$ and $\approx 30\%$, respectively. None of the models made to predict the hotel DHW load performed MAPE values under 100%, but the Normalized Root Mean Squared Error(NRMSE) values of 0.49 from the Prophet prediction and 0.47 for the XGBOOST prediction shows that the prediction is not as far off as the MAPE values imply. The MAPE is heavily affected by low true consumption values, as this error metric is calculated by dividing the prediction error in each timestep by the true value. However, no good predictions were made for hotel DHW load. The nursing home DHW load was most accurately predicted by the XGBoost model, with a MAPE of 37% and NRMSE value of 0.27.

Abbriviations

Table 1: Abbreviations and clarifications

Abbreviations and clarifications	Description
AB	Apartment Building
CV	Cross validation
Default Prophet model	The simplest prophet model, where no input parameters are tuned
DHW	Domestic hot water
Domestic	Used in this thesis as within households, not as within a certain country
HO	Hotel
MAE	Mean Absolute Error
Manual model set-up	Used as a reference to the data split in the Prophet prediction models.
MAPE	Mean Absolute Percentage Error
MDAPE	Median Absolute Percentage Error
ML	Machine Learning
MSE	Mean Squared Error
NH	Nursing Home
NRMSE	Normalized Root Mean Squared Error
PPS	Pre-Processing Strategy
"Pvv [W]"	Dataframe column name for DHW load in Watts (used only in Python code), (the data are later changed into [kW]).
XGBoost	eXtreme Gradient Boosting

Contents

List of Figures

List of Tables

1 Introduction

Some of the content in this chapter is retrieved from my specialization project, written in the fall of 2020 [3].

1.1 Background

In Norway, Europe and countries all over the world, renewable energy generation is seen as one of the solutions to combat the problem of global warming. This is solidified by the fact that as of mid-2015, 164 countries have renewable energy targets [4].

The renewable energy sources are more uncontrollable than the energy sources they are replacing. Weather conditions, seasons, and climate affect how and when renewable energy sources such as solar power and wind power can produce electricity. When energy generation is more unpredictable, it means that energy will become more unavailable at times, resulting in higher energy prices in certain periods where the overall energy consumption is higher. Another important aspect of the change in energy sources is that all renewable energy sources, apart from bio-energy and solar thermal energy, generates electricity [5]. An increase in electricity consumption is expected, also in Norway. According to [6], the electricity consumption will increase from 136 TWh in 2018 to 159 TWh in 2040. A power system needs flexibility in order to function. Flexibility has previously been provided by the generation side in the form of European fossil fuel and Norwegian hydropower. Because European countries are shifting from fossil fuels to more uncontrollable solar and wind power, the generation can no longer provide the needed flexibility [7]

Without energy storage options, a lot of electricity would have to be transported from production sites to end-users during high consumption periods. This could be a challenge for the Norwegian distribution grid. More energy efficient systems, energy storage solutions, and local flexibility markets could reduce this problem.

Buildings are one of the largest categories of energy consumers in the world. This can be visualized in Figure 1 from [8] where the total energy consumed by the OECD(Organisation for Economic Co-operation and Development) countries is plotted in Mtoe(Million ton oil equivalent) over time.

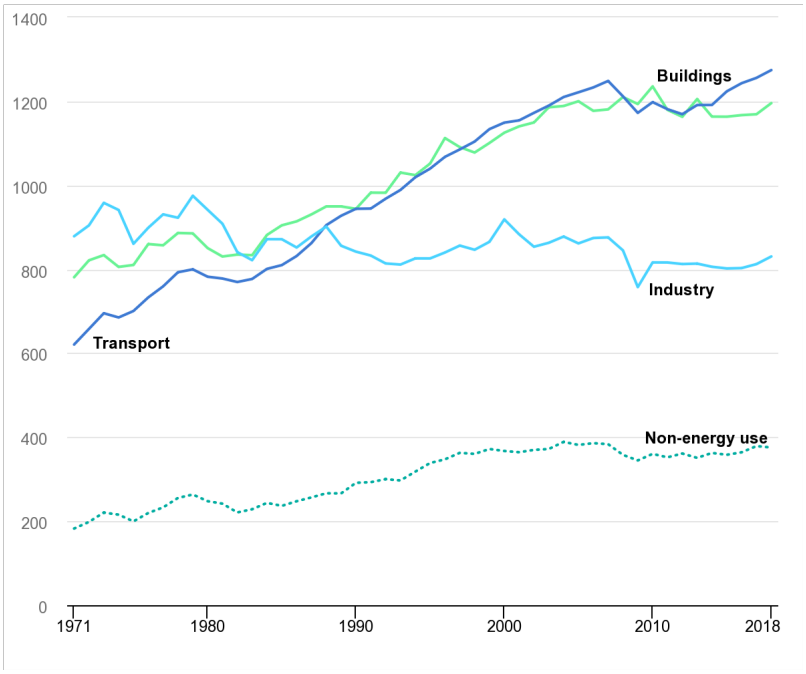


Figure 1: Total Energy Consumption by the OECD countries by Sector [y-axis: Mtoe, x-axis: year](IEA (2020), World Energy Balances: Overview, IEA, Paris https://www.iea.org/reports/world-energy-balances-overview. [8])

According to [8], the total energy consumed by the OECD countries in 2018 was 3784 Mtoe. As can be seen in Figure 1, the Buildings category stands for about 1200 Mtoe, which makes for $\frac{1200}{3784} \cdot 100\% \approx 32\%$ of the total energy consumed by the OECD countries. OECD has 37 member countries including the USA, Canada, Mexico, and most of the European countries, and is therefore seen as a valid example to compare to Norwegian conditions. According to [9], 15% of the total heat use in the EU is associated with DHW use, and 25-35% of the energy consumed in regular buildings is consumed by DHW systems.

With this reasoning, operating DHW systems more energy efficient seems like a useful measure as part of coping with the more renewable energy generation. Also, hot water tanks may be used for storing energy, and providing local flexibility. To be able to run DHW systems more efficiently and offer flexibility, we need to be able to predict the consumption. If we can know when a building needs hot water and when it does not, we can turn up the heating systems when the prices/demand is low, and store energy in the form of heat. When the local demand is high, but the building demand is low, flexibility can be offered in form of turning off heaters and use stored excess heat for supplying the building's own, low demand.

All these measures to run the DHW system more efficiently demands knowledge of future DHW consumption in the buildings. That is why DHW load predictions can be part of finding the solution to the future power systems flexibility demand, and contribute to more sustainable energy consumption in the future.

1.2 Scope

The purpose of this thesis is to investigate how machine learning techniques can be used to forecast DHW consumption in different building types in Norway. If reliable DHW predictions could be made by machine learning algorithms, efficient controlling of DHW systems can be used to provide the flexibility needed in future power systems. In this thesis, DHW consumption for 3 building categories are predicted by two different prediction models each. The models are made on the basis of historical data from the Varmtvann2030 data set [10]. Two different machine learning techniques are to be tested in this thesis. Prophet is an additive model that fits the training data with a trend component, Fourier series to fit the seasonal changes in consumption, and a holiday component to adjust for higher consumption during specified holidays. XGBoost is a mathematical optimization that groups the training data by explanatory variable values using regression tree boosting in order to minimize the prediction error.

The prediction models built in this thesis are built to be applicable for any apartment building(AB), hotel(HO), and nursing home(HO) in Norway. The only input data that is required to use the prediction models are hourly(at least) DHW measurements and the number of units in the building. This thesis will not produce a future prediction, but only validate the models against test data unseen by the models. By the terminology for forecast length presented in [11] and [12], the forecasts performed on test data in this thesis are medium-term forecasts. This term is defined for forecasts ranging from 1 day to 1 year. The prediction periods in this thesis are different for the three building categories, but all predictions fall under the category medium-term forecating.

1.3 Structure of the thesis

Chapter 2 gives an overview of the data sets and an initial analysis of the AB data from my specialization project [3]. In chapter 3, the theory behind the prediction models are explained. This chapter also describes how the DHW systems in the buildings analyzed in this thesis are set up. The method used to format the data, build and tune the prediction models are explained in chapter 4. The results of the two prediction models are presented for all three building categories in chapter 5. In chapter 6, the challenges with this task are discussed, and the result of tuning the Prophet models are displayed. The conclusion in chapter 7 sums up the findings of the thesis.

2 Data

The data used in this thesis is from the Vartmvann2030 project by SINTEF Community [10]. The Apartment Building(AB) data from the Varmtvann2030 data set was analyzed in my specialization project, and some of these analyses are relevant also for this thesis. Therefore, this chapter is partially retrieved from my specialization project [3].

2.1 Overview

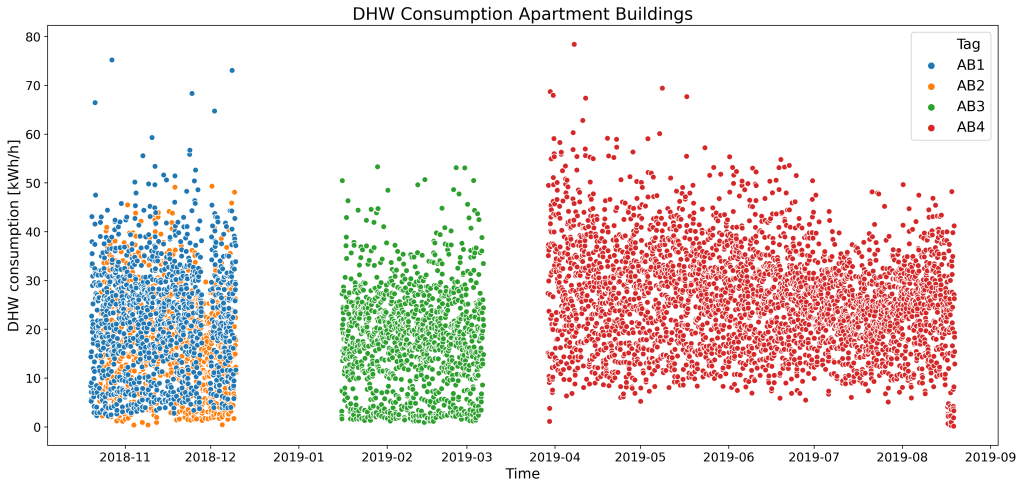
The data sets consist of hot water measurements in 4 apartment buildings, 4 hotels, and 4 nursing homes all located in Oslo, Norway. The data was originally structured as time series stored in csv files. The information about the different buildings in the Varmtvann2030 data set is summarized in Table 2

Table 2: Data information table

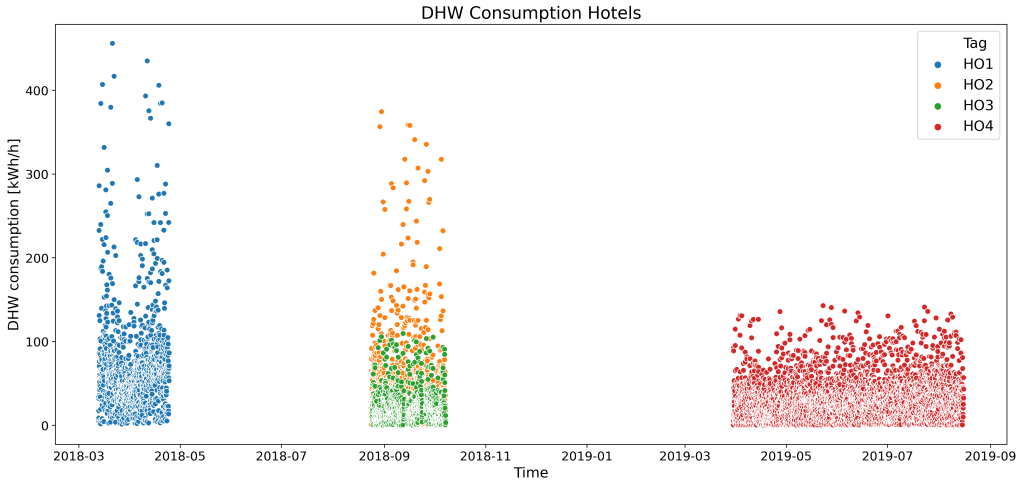
Building Name	Building Category	Number of Units	Floor Area [m^2]	Measuring Period	Mean DHW consumption value [$\frac{kWh}{h}$]
AB1	Apartment block	96	4400	19.10.2018-09.12.2018	22.32
AB2	Apartment block	56	2700	22.10.2018-09.12.2018	17.79
AB3	Apartment block	56	3752	16.01.2019-06.02.2019	18.02
AB4	Apartment block	86	5100	29.03.2019-19.08.2019	26.14
HO1	Hotel	434	21278	12.03.2018-24.04.2018	72.66
HO2	Hotel	355	24500	24.08.2018-07.10.2018	49.11
HO3	Hotel	165	4934	24.08.2018-07.10.2018	21.73
HO4	Hotel	151	7440	31.03.2019-15.08.2019	31.41
NH1	Nursing home	148	11618	25.01.2018-23.02.2018	23.40
NH2	Nursing home	52	3327	31.05.2018-11.07.2018	13.29
NH3	Nursing home	50	6774	26.05.2018-11.07.2018	7.19
NH4	Nursing home	96	10081	16.01.2019-06.03.2019	16.85

The data had a time resolution of measurements every two seconds. However, this data contains a lot of unrealistic values due to error in measurements. Therefore, the data is averaged into hourly values, this is explained in detail in chapter 4

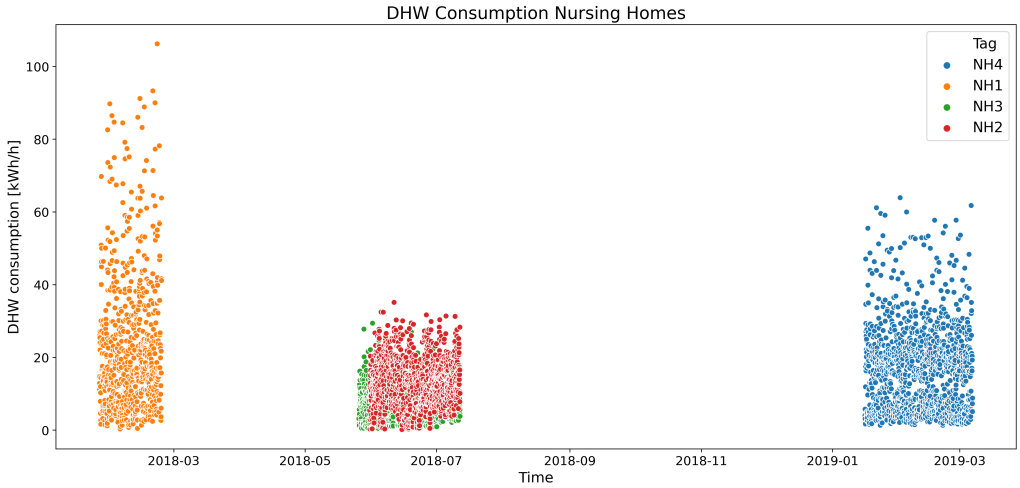
The metadata for the apartment buildings and the nursing homes is limited to number of units and total area in each building. The hotel data also contains information about the number of booked rooms and the number of guests checked in at every timestamp. The initial data contained both NAN(Not A Number) values and negative values. These values are removed, and the data used in the models are plotted with the timestamps in Figure 2.



(a) Apartment buildings



(b) Hotels



(c) Nursing homes

Figure 2: Initial data plotted against timestamps for (a) Apartment Buildings (b) Hotels (c) Nursing Homes (data from Varmtvann2030).

In Figure 2, there are some overlapping values, meaning there are measurements from two or more buildings at the same time. Therefore, to show the complete data sets in full, the data sets are plotted without a meaningful x-axis in Figure 36 in Appendix A.

2.2 Data Analysis

In my specialization project, average daily- and seasonal DHW load profiles for the apartment building data in the Varmtvann2030 data set were created. When tuning the prediction models in this thesis, it can be useful to know the characteristics of these average load profiles.

The mean daily DHW load profile for business days, Saturdays, and Sundays in the AB data are plotted hourly in Figure 3.

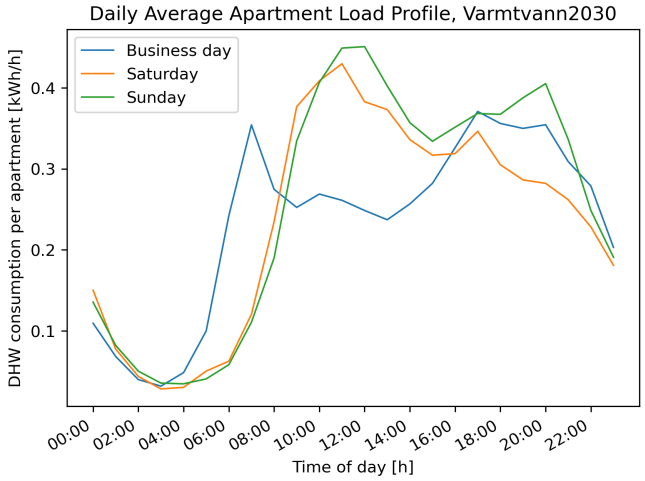


Figure 3: AB mean daily load profile, displayed in per unit consumption (per unit should on the y-axis)

The seasonal variations in daily mean DHW load in the AB data are plotted in Figure 4.

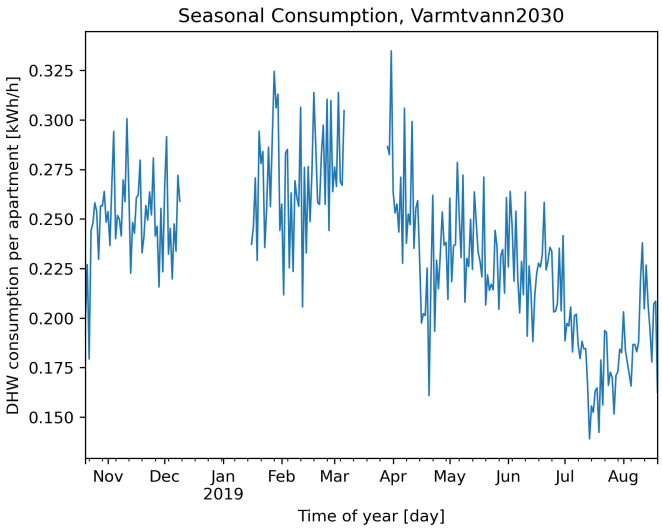


Figure 4: Seasonal variations in AB daily mean DHW load, displayed in per unit consumption (per unit should on the y-axis)

3 Theory

This chapter will introduce the theoretical parts of the thesis

In chapter 3.1, the DHW systems used in the buildings that are analyzed in this thesis are explained. This part is retrieved from my specialization project, written in the fall of 2020 [3].

The theory behind the two prediction models is explained in the chapters 3.2 and 3.3.

3.1 DHW system

This chapter is retrieved from my specialization project [3]. Hot water can be supplied to residential buildings in two different ways, with a central heating system in the building, or with individual hot water supply systems. The system used in the buildings in this thesis is a central heating system. Cold water enters the heating central and is warmed up and distributed to the apartments through a circulation system. The circulation system consists of pipes which transport the water from the heating central to the water taps and back again. The hot water in the pipes will have a heat loss to the environment, causing a temperature drop. The water is circulated back to the heating central to maintain a high temperature so that the residents can get hot water from the water taps almost instantly. Figure 5 shows a sketch of a hot water distribution system with circulation and a measuring setup.

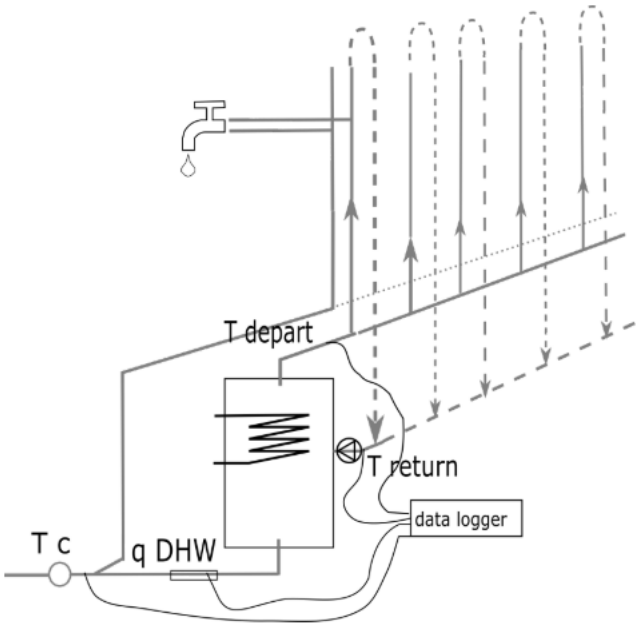


Figure 5: Hot water distribution system in apartment blocks (Figure 1. in [13]).

Here T_C is the temperature of the cold water entering the building, q_{DHW} is the water flow rate, T_{depart} is the temperature of the heated water going to the individual units, and T_{return} is the temperature of the returning circulated hot water. The data logger is logging the measurements. The lines out of the data logger show where the measurements are taken.

If we neglect the heat loss in the circulation system, the energy needed for supplying hot water is found by Equation 3.1.

$$Q = \frac{q_{DHW} \cdot \rho \cdot C_P \cdot (T_{depart} - T_C)}{3600} \tag{3.1}$$

, *where*

q_{DHW} = volume flow rate

ρ = the density of water

T_{depart} = hot water temperature

C_P = specific heat capacity of water

T_C = cold water temperature

[14]

The data sets I received from Varmtvann2030 were already converted into energy data, so the hourly values used in this thesis are hourly values of Q from Equation 3.1. It should be noted that the specific measurement locations vary in some of the buildings. The locations are not provided in the data set. However, correspondence with Varmtvann2030 contact personnel at SINTEF AS confirms that the setup described above reflects the reality for the buildings taken into account in this thesis.

3.2 Prophet

Prophet is an open-source software developed by Facebook [15]. Prophet is implemented in Python and is a procedure for forecasting time series data. The Prophet forecasting model is explained in full by Taylor, S.J. and Letham B. in [16]. This chapter will summarize the theory behind the Prophet forecasting model. The theory and equations are inspired by and retrieved from [16].

The procedure is based on an additive model which fits seasonal effects with a trend component and holiday effects.

The Prophet prediction model can be explained mathematically as shown in Equation 3.2.

$$y(t) = g(t) + s(t) + h(t) \tag{3.2}$$

, *where*

t = Time variable

$y(t)$ = Forecast

$g(t)$ = Trend component

$s(t)$ = Seasonal component

$h(t)$ = Holiday component

In Prophet there are three alternatives for the trend component, $g(t)$, a piece-wise linear model, a saturating logistic growth model and a flat growth. A piece-wise linear regression can be done by finding pieces in the total graph that looks like can have an approximately linear form, and fitting a linear function on this piece of the graph. An example of a mathematical formulation of a piece-wise linear function is shown in Equation 3.3, and a graphical example is shown in Figure 6

$$y(t) = \begin{cases} \beta_{10} + \beta_{11} t & 0 \leq t \leq t_1 \\ \beta_{20} + \beta_{21} t & \tau_1 \leq t \leq \tau_2 \\ \beta_{30} + \beta_{31} t & \tau_2 \leq t \end{cases} \tag{3.3}$$

, *where*

$y(t) =$ Piece-wise linear function

$\beta_{*0}, \beta_{*1} =$ Coefficients

$t =$ Time variable

$\epsilon =$ Error term

$\tau_1, \tau_2 =$ Time values

There are several ways to compute the coefficients for the piece-wise linear function, one method is the least square solution shown in Equation 3.4.

$$\beta_0 = \frac{\sum_i (t_i - \overline{t})(y_i - \overline{y})}{\sum_i (t_i - \overline{t})^2} \tag{3.4}$$

, *where*

$y_i =$ Dependent variable value at observation i

$t_i =$ Time of observation i

$\overline{y} =$ Mean of y

$\overline{t} =$ Mean of t

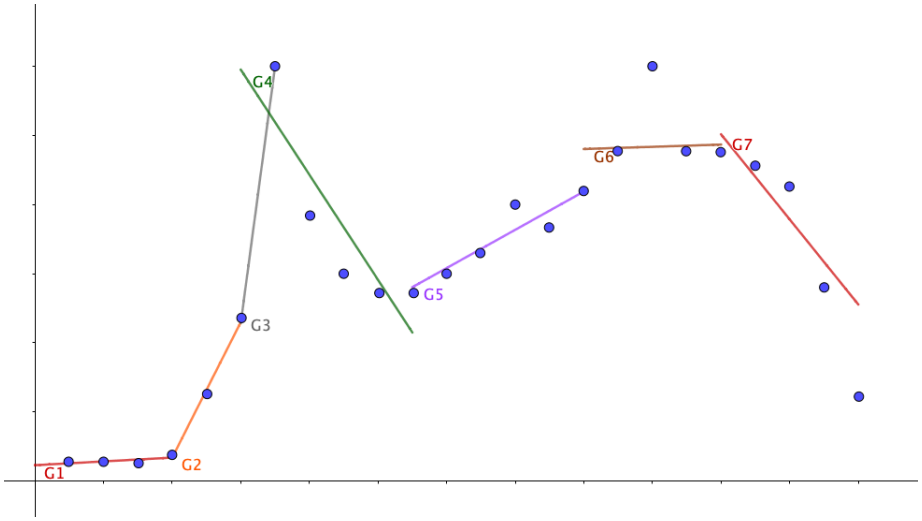


Figure 6: Piece-Wise Linear Regression Example

Saturating logistic growth allows the growth to be non-linear, for example exponential. This option is useful when the time series has a carrying capacity or a maximum level, and/or a floor value [17]. The logistic growth term can be mathematically described as in Equation 3.5.

$$g(t) = \frac{C(t)}{1 + t^{-k(t-m)}} \tag{3.5}$$

, *where*

$g(t) =$ Trend component

$t =$ Time variable

$k =$ Growth rate

$m =$ Offset

The third and last option for the trend component is flat. If the trend component is set to flat growth, the trend component will be a constant term [17].

The seasonal component relies on Fourier series to fit the periodic effects. A Fourier series is composed of a sum of sine and cosine functions with a coefficient for each trigonometric function. The mathematical formulation of a seasonal component can be seen in Equation 3.6

$$s(t) = \sum_{n=1}^{N} (a_n \cos(\frac{2\pi nt}{P}) + b_n \sin(\frac{2\pi nt}{P})) \tag{3.6}$$

, *where*

$s(t)$ = Seasonal component

n = Index value

N = Upper limit to number of Fourier coefficient pairs

t = Time variable

a_n, b_n = Fourier coefficients

P = Time period

In this thesis, there will be more than one seasonal term. The data has both weekly and daily periodic variations, which need to be fit with a Fourier series each. The time variable, t, in Equation 3.6, is different for each different seasonality component. For example, the time variable will have hourly values for the daily seasonality component and daily values for the weekly seasonality component. The same goes for the time period, P, which for the weekly seasonality component will be 7 days, and for the daily seasonality component will be 24 hours.

2N coefficients must be estimated to fit the seasonality component to the training data. N truncates the Fourier series like a low-pass filter for the seasonality. Increasing N will allow the model to fit more frequently changing patterns such as the daily variations in this thesis. This however increases the risk of overfitting. The coefficients a_n, b_n, by default, stays within a normal distribution around 0 with a variance of 10.

The holiday component, $h(t)$, adjusts to higher or lower values based on historic data from specified or default holidays.

A Prophet model fits the function $E(t)$ shown in Equation 3.2 to the training data and continues the curve with the given input, which is just the timestamps in the test data. This creates prediction values for the test part of the data set. To make changes to the Prophet function to improve the predictions are called tuning and is explained in chapter 3.2.1

3.2.1 Tuning the Prophet Model

Prophet has built-in hyperparameters which can be tuned to better fit the trend component, the seasonal periodic variations and the holiday effects in the data.

The most relevant hyperparameters in this thesis are the ones connected to the seasonal component. These hyperparameters decide how to prioritize the yearly, weekly and daily variations in the consumption. According to [18], the hyperparameters that most likely could be tuned to improve the predictions are "changepoint_prior_scale", "seasonality_prior_scale", "holidays_prior_scale" and "seasonality_mode".

The "seasonality_prior_scale" represents the variance in the normal distribution of the Fourier coefficients. By default this parameter is set to 10, and increasing its value will increase the variance in the normal distribution of the Fourier coefficients. The default value of 10 is set high, as according to [18] the model rarely overfits the seasonality because the Fourier series are truncated.

As mentioned above, the Fourier series are truncated at N pairs of coefficients and trigonometric functions. This number may be increased for the individual seasonality components. For example, the N for the daily seasonality component is increased by increasing the value of the "daily_seasonality" input parameter. The same goes for the weekly and yearly seasonalities with the input parameters "weekly_seasonality" and "yearly_seasonality" respectively. As mentioned, increasing these parameters increases the number of Fourier coefficient pairs and trigonometric functions, which allows more frequent changing seasonal components.

3.2.2 Prophet Uncertainty Interval

Prophet automatically forecasts an 80% uncertainty interval along with the actual prediction. The uncertainty comes by default only from the trend component. The uncertainty is estimated using Monte Carlo simulation. The Monte Carlo simulation is explained in full in [19], and is summarized from the same source as

> " *PROCEDURE FOR APPLYING MONTE CARLO*
>
> *1. Determine the pseudo-population or model that represents the true population of interest.*
>
> *2. Use a sampling procedure to sample from the pseudo-population.*
>
> *3. Calculate a value for the statistic of interest and store it.*
>
> *4. Repeat steps 2 and 3 for N trials.*
>
> *5. Use the N values found in step 4 to study the distribution of the statistic.* " [19, p. 2].

3.3 XGBoost

XGBoost stands for eXtreme Gradient Boosting. The theory behind XGBoost is explained in full in the original manifest by Chent, T. and Guestrin, C. in [20]. Parts of the theory behind the method are based on advanced mathematics. This chapter will elaborate on the method used by the XGBoost library, however, all this theory is not a necessity to be able to use the model for forecasting.

3.3.1 Overview

The XGBoost library implements the gradient boosting decision tree algorithm. The concept behind boosting is that each new decision tree added to the model should learn from the errors the last tree made. This process repeats itself until either the preset maximum amount of trees are created, or the new tree is not able to improve the last prediction.

More specifically, the gradient boosting regression tree algorithm creates decision trees/regression trees to predict the errors made by the existing trees using the gradient descent method to minimize the loss when adding new trees. The regression trees are added together sequentially to make the final prediction. For clarification, the difference between decision trees and regression trees are that decision trees are used for classification while regression trees are used for regression. In this thesis XGBoost is used for regression, so only regression trees are made in this thesis.

3.3.2 Detailed/mathematical explanation of the process

This mathematical explanation, including the equations, is inspired by and retrieved from the original manifest [20], and a summary of the process by Leventis, D. in [21].

The first prediction the model does on the training is done in a way that minimizes the sum of residuals/errors over all rows in the training data. For the initial prediction this is simply the mean of all observations. The residuals from this initial guess is then stored to be used in future iterations. These residuals are calculated by the chosen loss function, which for XGBoost is shown in Equation 3.7.

$$\ell_i(y_i, \hat{y}_i) = \frac{1}{2}(y_i - \hat{y})^2 \tag{3.7}$$

, *where*

i = Row index value (timestamp for time series data)

y_i = Observed value at i

$\hat{y}_i$ = Predicted value at i

This loss function is the difference between the observed value and the prediction scaled by a half. This scaling is done to make the math easier later in the process when the loss function is differentiated to find the gradient.

The next iteration creates a regression tree to predict the residuals from the initial guess. The theory behind decision trees/regression trees can be studied in detail in [22]. The basic concept here is to split the data into smaller and smaller groups by separating the data on feature/variable values. An example related to the data used in this thesis of such a regression tree is shown in Figure 7.

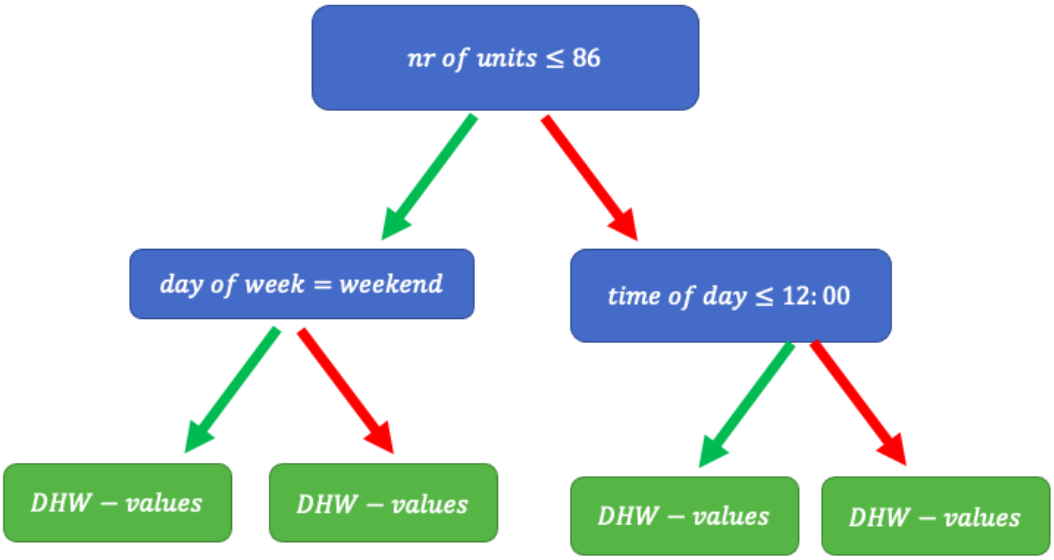


Figure 7: Regression tree Example

In Figure 7, the green nodes are called leaf nodes and contain the residuals for the data rows with the information that is specified in the previous nodes. So the leftmost leaf node contains all residuals between the previous prediction and the real observed hourly DHW consumption values for weekend days in buildings with 86 units or less.

For the next iterations, a new regression tree is made by the argument of minimizing the objective function shown in Equation 3.8. The new regression trees are called learners and have the job to incrementally improve the prediction, or to make the error incrementally smaller.

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{3.8}$$

, *where*

$\mathcal{L}^{(t)}$ = Objective function for iteration t

i = Row index value (timestamp for time series data)

n = Number of data points in the training data

y_i = Observed value at i

$\hat{y}_i$ = Predicted value at i

$f_t(x_i)$ = New learners function

$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda ||w||^2$

In Equation 3.8, ℓ is a loss function, for example like the function described in Equation 3.7.

> " *The second term Ω penalizes the complexity of the model (i.e., the regression tree functions). The additional regularization term helps to smooth the final learned weights to avoid over-fitting. Intuitively, the regularized objective will tend to select a model employing simple and predictive functions.)*" [20, p. 786]

The objective function in Equation 3.7 "cannot be optimized using traditional optimization methods in Euclidean space" [20]. Therefore, to optimize the function, a Taylor's approximation is used. This is done by using a linear approximation to approximate the objective function.

$$f(x) \approx f(a) + f'(a)(x - a) \tag{3.9}$$

In Equation 3.9, f is the loss function shown in Equation 3.7, a is the previous step's predicted value, and $(x - a)$ is the learner we are adding in step t [21]. The f in Equation 3.9 is just an example of a function name, so it not the same f as the f_t in Equation 3.8. Using the Taylor approximation we can write the objective function as a simple function of the new learner. If we now chose a second-order Taylor approximation like in Equation 3.10, we can find the learner which minimizes the loss function at iteration t with Equation 3.11.

$$f(x) \approx f(a) + f'(a)(x - a) + f''(a)(x - a)^2 \tag{3.10}$$

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n}[\ell(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i)) + \frac{1}{2} h_i f_t^2(x_i))] + \Omega(f_t) \tag{3.11}$$

, *where*

$$g_i = \frac{\partial}{\partial \hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) = (y_i - \hat{y}_i^{(t-1)})$$

$$h_i = \frac{\partial}{\partial \hat{y}^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)}) = 1$$

In Equation 3.11 the g_i and h_i are partial derivatives of the loss function, ℓ, in Equation 3.7. It can be observed that g_i becomes just the residual between the observation and the prediction, and that h_i is just 1. This is handy when we go further and actually optimize the Taylor approximated objective function.

When optimizing a function, we can remove the constant terms, as they have no influence on the decision. We now get the simplified expression for the Taylor approximated objective function shown in Equation 3.12

$$\tilde{\mathcal{L}}^{(t)} \approx \sum_{i=1}^{n}[g_i f_t(x_i)) + \frac{1}{2} h_i f_t^2(x_i))] + \Omega(f_t) \tag{3.12}$$

Now, to build a new learner we start in a root node containing all training data. Then we iterate over all features and evaluate each possible split loss reduction, and choose the split with the highest gain. The gain is calculated in Equation 3.13 and must be greater than zero to be an improvement on the last prediction.

$$\text{"}\textbf{gain} = \textbf{loss}(\text{father instances}) - (\textbf{loss}(\text{left branch}) + \textbf{loss}(\text{right branch}))\text{"} \text{ [21]} \tag{3.13}$$

If we now define $I_j = i|q(x_i) = j$ as the instance with the set of leaves j, we can rewrite Equation 3.12 by expanding Ω. This is shown in Equation 3.14

$$\tilde{\mathcal{L}}^{(t)} \approx \sum_{i=1}^{n}[g_i f_t(x_i)) + \frac{1}{2} h_i f_t^2(x_i))] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} ||w_j||^2 \tag{3.14}$$

$$\tilde{\mathcal{L}}^{(t)} \approx \sum_{j=1}^{T}[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \tag{3.15}$$

For a fixed tree structure $q(x)$ we can compute the optimal weight w_j^* of leaf j by Equation 3.16

$$w_j^* = -\frac{(\sum_{i \in I_j} g_i)}{\sum_{i \in I_j} h_i + \lambda} \tag{3.16}$$

From here we can compute the corresponding optimal value of the structure $q(x)$ in Equation 3.17.

$$\tilde{\mathcal{L}}(q)^{(t)} = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T \tag{3.17}$$

Equation 3.17 is used as a scoring function for the different tree structures. This is the function we use to measure how well a tree performs. Looking closer at the function we can see what we discovered in Equation 3.11, that the weight of each leaf is just the mean of the residuals in that leaf when $\lambda = 0$. The scoring function is also only depending on the residuals in that leaf.

This is the mathematical explanation of how the XGBoost algorithm works. How the XGBoost model has been implemented and used in this thesis is explained in chapter 4.5.

3.4 Error Metrics / Key Performance Indicators

To evaluate the performance of a prediction model, there are many different options of error metrics. The different kinds of error metrics are affected by different kind of data, and it is therefore an important process of prediction modeling to find out which error metric that gives the most realistic insight into the model performance.

A common error metric to use is the Root Mean Squared Error (RMSE). RMSE is calculated as shown in Equation 3.18. This is a reliable metric to use when evaluating different tuning of the models, because in practice this error term is only dependent on the real difference between the prediction and the true value.

$$RMSE = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(y_t - \hat{y}_t)^2} \tag{3.18}$$

, *where*

y_t = True value at time t

$\hat{y}_t$ = Predicted value at time t

T = number of data samples/time stamps

The same goes for the Mean Absolute Error(MAE), which basically is the mean of the gap between the true value and the prediction. MAE is calculated as shown in Equation 3.19.

$$MAE = \frac{1}{T}\sum_{t=1}^{T}|y_t - \hat{y}_t| \tag{3.19}$$

, *where*

y_t = True value at time t

$\hat{y}_t$ = Predicted value at time t

T = number of data samples/time stamps

The drawback of using RMSE or MAE as performance indicators is that one can not compare prediction models which take in data of different magnitude with these metrics.

The two prediction models used in this thesis uses data of different magnitude, this is explained in subsection 4.1. It is therefore useful to use a normalized error metric that is independent of magnitude. An example of such an error metric is Mean Absolute Percentage Error (MAPE). This metric is calculated as shown in chapter 3.20

$$MAPE = \frac{1}{T}\sum_{t=1}^{T}\frac{y_t - \hat{y}_t}{y_t} \tag{3.20}$$

, *where*

y_t = True value at time t

$\hat{y}_t$ = Predicted value at time t

T = number of data samples/time stamps

This error metric also has a drawback. The MAPE value is often heavily affected by small true values, because when the true value gets close to zero, the value of the fraction in the MAPE formula gets really large. This will result in a MAPE that does not give a realistic picture of the model performance.

Another normalized error metric is Normalized Root Mean Squared Error (NRMSE). The RMSE can be normalized in several ways, for example by dividing the value with the mean true value as shown in Equation 3.21. This metric is less vulnerable for low true values because the normalizing factor is an averaged value.

$$NRMSE = \frac{RMSE}{\overline{y}} = \frac{1}{\overline{y}}\sqrt{\frac{1}{T}\sum_{t=1}^{T}(y_t - \hat{y}_t)^2} \tag{3.21}$$

, *where*

y_t = True value at time t

$\hat{y}_t$ = Predicted value at time t

T = number of data samples/time stamps

$\overline{y}$ = Mean of y_t over period T

3.5 Literature Review

This section will introduce previous works written on the subject of using machine learning techniques to forecast DHW- or energy consumption. The majority of the literature written on the subject is focusing on forecasting electricity loads. However, there are a lot of similarities between forecasting electricity- and DHW consumption. Therefore, the methods used for predicting electricity loads can be highly applicable for predicting DHW consumption.

In [9], Dmytro Ivanko evaluates two different situations when predicting DHW heat use in Norwegian hotels. In the first situation, only historical DHW consumption data is available. In this situation, the author suggests some model alternatives. Of the suggested models, the Prophet- and XGBoost models were the best performing models with MAE's of 4.46 and 4.11 respectively. In the second situation, additional variables were taken into account. This is a more favorable situation as additional variables yield a better or equal model performance. In this situation, the XGBoost model was the best performing model with a MAE of 3.12 and a MSE of 45.04.

A literature review showing the growth in energy forecasting papers has been done in [11]. A more specific literature review on predicting energy consumption with machine learning methods has been conducted in [23]. In this review, a database containing relevant research on the subject is created by exploring the ISI and Scorpus databases with a specified search algorithm. The database contains 4300 papers and also shows the increase in papers written on the subject over the past years. This increase can be observed in Figure 8.

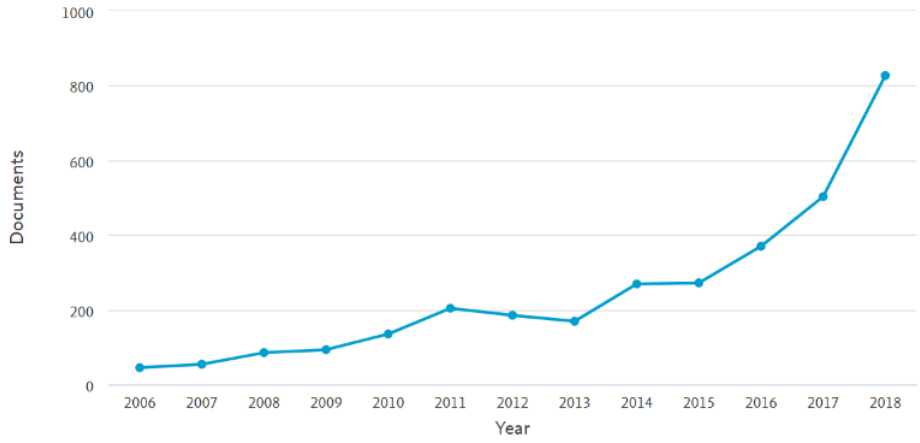


Figure 8: Increase in papers written on Energy consumption prediction using Machine Learning [23]

In [12], European electricity loads are predicted using artificial neural networks(ANN). One of the things that are applicable towards this thesis is the use of error metrics. The authors use MAPE to compare the results in the paper to previous work. The model in the article is compared to a benchmark, which is set by the Mid-term Adequecy Forecast (MAF) model by entso-e. The ANN model developed in [12] performs a MAPE of 2.8% compared to the MAPE of 3.8% performed by the MAF model.

ANN was also used for short-term energy forecasting in [24]. In this paper, the authors used input parameters like the number of residents, apartment area, and electrical appliance consumption to forecast the total energy demand in domestic households in Lisbon, Portugal. The average MAPE result obtained from predicting the total energy consumption in 46 households was 4.2%.

In [25] a clustering-based hybrid model is proposed as a method for short-term electricity demand forecasting in hotels. The authors uses fuzzy c-means clustering to group similar days and hours. Further, these grouped days and hours were used as input in support vector regression and wavelet decomposition. A MAPE result of 3.8% for the best predicted 24 hours, was performed by this model.

4 Method

This chapter describes the processes of importing and analyzing data, building different prediction models, and plotting the results.

To predict 'future' DHW consumption, historical data must be imported and formatted as input in a prediction model. The natural way to go about this is by using a programming language. In this thesis, Python in the Spyder environment, is used both to import, analyze and format data, and to build two different prediction models. Additionally, an attempt was made to build a prediction model with pre-defined blocks of code in Microsoft Azure Machine Learning Studios [26]. This attempt did not produce any results but was something I spent some time on and is therefore included here in chapter 4.3.

4.1 Data processing

In order to process the data, the data from each building is imported into Pandas Dataframes. Next, the NAN and negative values are removed from the data. Then the data are resampled into hourly values by taking the mean of every 2-second DHW load value in each hour. Removing faulty measuring data before resampling is essential to keep as much as possible of the good measurements. The metadata was sent to me in a separate excel-file, so this data was attached in the consumption Dataframe by the code in Figure 9. The name of each building is added to the Dataframe as a "Tag" to be shown when plotting the initial data. This tag is removed after storing the initial data in a csv file. The data later fed into the models are without a tag, as the data are to be seen as uniform. The code used for importing and processing the data is shown in Figure 9.

```
#Reading the data pickle files and creating a Dictionary of Dataframes
    containing the AB data
dfDict = ReadFolderofPicklestoDf('*directory*/AB/')
#Reading in metadata to a dataframe
ByggInfo= pd.read_excel('*directory*/ByggInfoVV2030.xlsx', header=0)
#setting the building names as index in the metadata dataframe
ByggInfo=ByggInfo.set_index('Code')
#Iterating over the dictionary to access the individual dataframes
for key in dfDict:
    #Changing the content of the Dictionary to Dataframes containing only the
        DHW heat use
    #.. and removing NANs
    dfDict[key]=dfDict[key][['Pvv [W]']].dropna(axis=0, how='any')
    #Removing unrealistic negative values in the data set, setting them to zero
    dfDict[key]=dfDict[key][dfDict[key]['Pvv [W]']>=0]
    #Resampling into hourly consumption
    dfDict[key]=dfDict[key].resample('1H').mean()
    #Converting the timestamps to DateTime format
    dfDict[key]['DateTime']=pd.to_datetime(dfDict[key].index, utc=True)
    #Setting the DateTime column as index
    dfDict[key]=dfDict[key].set_index(['DateTime'])
    #Importing building area from metadata
    dfDict[key]['Area']=ByggInfo.loc[key, 'Area']
    #Importing number of units in the building from metadata
    dfDict[key]['Units']=ByggInfo.loc[key, 'Units']

    #setting a tag column to keep track of the different buildings
    dfDict[key]['Tag']=key
    #merging the Dataframes into one Dataframe containing all AB data
    AB_df=pd.concat([AB_df,dfDict[key]])

#saving the Dataframe as a csv file for later plotting of initial data
AB_df.to_csv('AB_initial_data.csv')

#removing the tag to keep data unbiased when saving the data for prediction
AB_df=AB_df.drop(columns=['Tag'])
AB_df.to_csv('AB_df.csv')
```

Figure 9: Importing data into Dataframes and pre-processing

Pandas is an easy-to-use, open-source, data analysis and manipulation tool, built on top of the Python programming language[27]. Dataframes are tabular data structures which is efficient to use when analyzing time series data. To illustrate how the data is structured in Dataframes, an example is shown in Figure 10.

```
In[20]: AB_df.head()
Out[20]:
                           Pvv [W] Area Units Tag
DateTime
2019-01-16 00:00:00+00:00 7473.849077 3752  56  AB3
2019-01-16 01:00:00+00:00 2870.517134 3752  56  AB3
2019-01-16 02:00:00+00:00 3195.886066 3752  56  AB3
2019-01-16 03:00:00+00:00 2131.860057 3752  56  AB3
2019-01-16 04:00:00+00:00 1629.295317 3752  56  AB3
```

Figure 10: First five rows of the Apartment Building Dataframe

4.2 Splitting Data

The prediction models are trained with the data described in chapter 2. In order to have a reference to how well the models perform the data must be split into training data and test data. The model is fitted to the training data and evaluated on the test data. There is much theory on how to split data into training data and test data for time series, but this is not the focus of this thesis. Therefore a convenient split date is set for the three different building categories so that the training portion of the data makes up between 70 and 80% of the data. Performance of default Prophet and XGBoost models were used to find a good split date using between 70 and 80% of the data for training. The split for the AB-data is made by the code shown in Figure 11.

```
#Split into training- and test data
split_date = '2019-05-18 23:00:00'
AB_prUnit_train = AB_prUnit_df.loc[AB_prUnit_df.index <= split_date].copy()
AB_prUnit_test = AB_prUnit_df.loc[AB_prUnit_df.index > split_date].copy()
```

Figure 11: Splitting data into training- and test data

The reason for choosing this data split for all the data fed into the models is so the models would have about the same proportion of training data for all building types. This would make for a more unbiased comparison between the models and the building types. Another consideration taken into account is that all models should be trained with some data from the specific building for which the predictions are made. This data split between training data and test data will hereby be called the manual model set-up, and are different for each of the building categories, but equal for the different models predicting on each building category. The data split resulted in a prediction period of 93 days for the AB data, 58 days for the HO data, and 34 days for the NH data.

4.3 Microsoft Azure Machine Learning Studio

Microsoft Azure Machine Learning Studio is a cloud-based workflow environment that lets you drag and drop different modules of code, both pre-defined and self-written, in order to create a machine learning model. Microsoft Azure Machine Learning Studio is not free, but one can get a free trial month, which is what I did. I no longer have access to my projects, so I do not have lots to show the work I did in this program. However, an overview of the project can be seen in Figure 12, which is a screenshot of one of the models I made in this program.

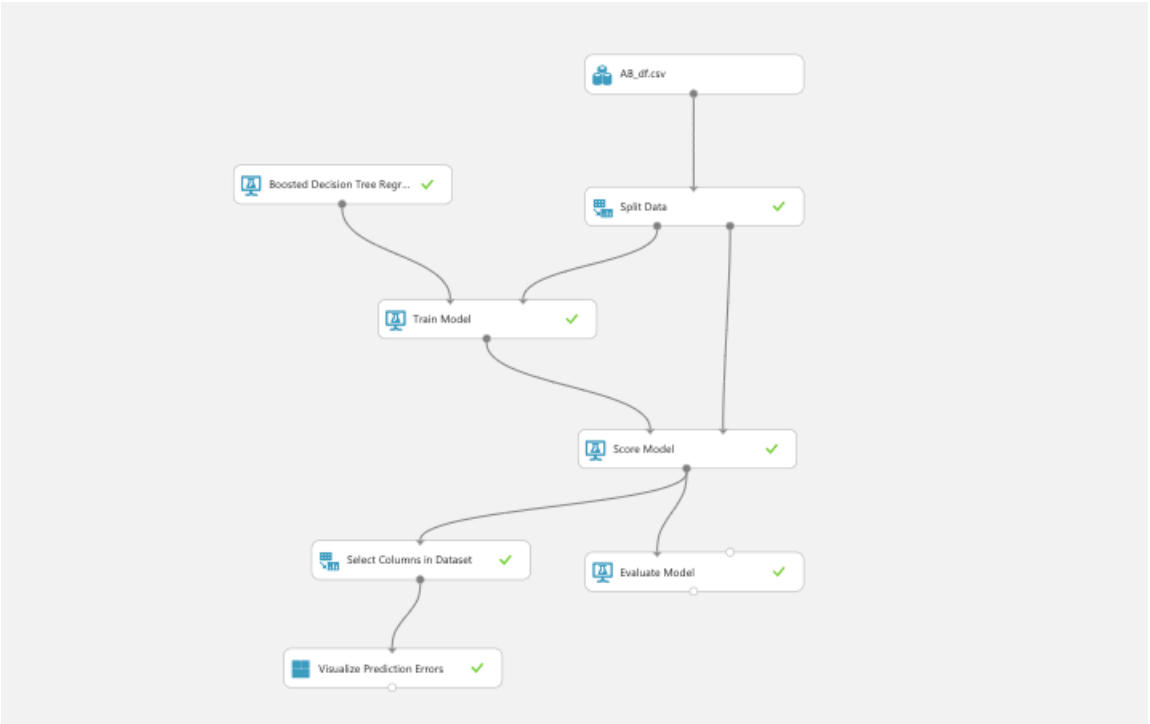


Figure 12: Microsoft Azure ML Studios model

The pre-processed Python Pandas Dataframe is fed into a split data module, which is set to divide the data into 70% training data and 30% test data. The training data is used to train a default built boosted decision tree regression. After that, the model is scored with RMSE and is to be evaluated and visualized in the blocks at the bottom of Figure 12. However, I could not get a visual prediction plot or any error metrics from this program. After a lot of attempts to find an answer to this both searching the internet and reaching out to contacts in SINTEF and NTNU, the project of making a machine learning model in Microsoft Azure Machine Learning Studio was abandoned. Microsoft Azure Machine Learning Studios can be found by following the link in the reference list [26].

4.4 Prophet Prediction

Prophet is an open-source software and was installed from the Prophet library in Python.

Individual Prophet models are trained for each of the three different building types(apartment buildings(AB), hotels(HO), and nursing homes(NH)). The model is then used to predict the consumption in the test part of the data. This section will go into detail on the method used to build the Prophet model.

The Prophet model does not take in any other variables than time. The data must therefore be uniform when it comes to explanatory variables. Within the building categories, the data from the different buildings have different consumption magnitude levels due to differences in area and number of units. In order to feed the Prophet model with data from all 4 buildings, the consumption can be divided into per unit- or per m2 consumption. This gives the data the same reference level.

To decide whether to use per unit- or per m2 consumption in the Prophet models, both options were tested in the default Prophet model, and evaluated by the error metrics MAPE and NRMSE for comparison. For the AB and NH data, the per unit consumption performed best, which can be seen by the metric comparison in Appendix C. For the HO data, there are more options. For this data set, we have additional information about the number of guests and the number of booked rooms in the hotel at any time. According to [28], the number of residents have a stronger correlation to DHW consumption than the number of units and the total area. The error metrics comparison in Table 8 in Appendix C shows that the results from the default Prophet model fed per guest DHW consumption data perform the best MAPE and NRMSE. However, in order to use the Prophet prediction models, the predicted consumption has to be multiplied with the feature it is divided by in this process. If one was to use the HO per guest Prophet model for hotel DHW load prediction, the prediction must be multiplied with the number of guests checked into the hotel at each timestamp to get the total DHW load of the predicted hotel. A problem that occurs here, is that the number of guests checked into the hotel at a timestamp one week into the future, may not be available information. Hotels can have drop-in guests which makes it difficult to use per guest data for DHW load prediction, as the number of guests staying in the hotel in the future may not be known.

For the HO Prophet model, the per m^2 data also performs marginally better than the per unit data. However, since the performances are so close, and the other building categories perform better with per unit data, a choice was made to use per unit data for all Prophet models for easier performance comparison. This means the prediction must be multiplied by the number of units to get the total DHW load of the predicted building.

The simplest Prophet model, hereafter called the default Prophet model, is built with the code shown in Figure 13. (All building category default models have the same code set-up, but the example is from the AB default Prophet model.)

```
#Setup, train and fit model
model = Prophet()
#Fitting model to training data
#..and performing necessary formatting so the model understands the column
    values in training set
model.fit(AB_prUnit_train.reset_index() \
             .rename(columns={'DateTime':'ds',
                              'Pvv pr unit [W]':'y'}))

#Performing necessary formatting so the model understands the column values in
    test set
df=AB_prUnit_test.reset_index() \
                                 .rename(columns={'DateTime':'ds'})
#Forecasting the test data
AB_prUnit_test_fcst = model.predict(df)
```

Figure 13: Python code for creating the default AB Prophet model

By manually tuning some of the parameters listed in [18], the results of the AB Prophet model improved in form of visual inspection and error metrics. Among the input parameters to be changed is the "growth" parameter, which represents the trend component mentioned in chapter 3.2. This parameter is by default set to "linear". By the reasoning in 3.2, setting this parameter to "logistic" would fit data with a logistic floor better. DHW heat use can not

be negative, so the "floor" value is set to 0. Theoretically, there is also a logistic maximum value for these data points, if every water tap uses maximum volume and temperature, the consumption can not increase from that point. However, this point is unknown, and not possible to compute without more information about the water taps in the buildings. Therefore, the highest value of the training data is set as the "cap" value.

The manual tuning of the Prophet models was done by changing the hyperparameters one by one and comparing the model error metrics to the default Prophet model, which for the AB data is shown in Figure 19. This process is time-consuming, and not very efficient. However, this was the method that yielded the best results achieved by the different AB Prophet models, but the method did not improve on the default model prediction for the HO- and NH data.

Tuning up the daily_seasonality value helps the model capture the low night load values and high morning peaks. Tuning up the weekly_seasonality value helps the model distinguish between the different consumption patterns in the different weekdays. There seems to be a trade-off between these two seasonalities when tuning the model. If the weekly seasonality is high, then the model distinguishes more differences between the different days of the week. If the daily seasonality is high, then the model captures more of the frequent daily variations in DHW consumption. However, the trade-off lies in the fact that if both these parameters are set high, the model can easily overfit the training data, which increases the RMSE in the test data prediction.

As can be observed by comparing the daily and weekly variations to the seasonal variations in Figure 3 and Figure 4 respectively, the daily variations are more dominant than the weekly and yearly variations in DHW load. Specifically, this can be seen by comparing the magnitude of the variations, the variations within a day are about 0.35 kWh/h per unit. The difference between a business day and a weekend day can be seen as weekly variations and seems to be about 0.15 kWh/h per unit at most. Therefore, setting the "daily_seasonality" parameter higher than the "weekly_seasonality" parameter will logically yield a better prediction.

The variations across a year can be observed in Figure 4, where the mean daily AB DHW load has been plotted for the entire measuring period. Here the variations are about 0.20 kWh/h per unit. Another factor in the hyperparameter tuning is the short time span of the data used in this thesis. For each of the possible seasonality components, the training data must contain enough data to capture the seasonality, at least a year for yearly seasonality, at least a day for daily seasonality, and so on [29]. None of the building categories have data spanning an entire year, therefore the "yearly_seasonality" parameter is not changed, and is defaulted to False. The seasonal changes can still be caught by the trend component in the Prophet model.

The Python code for the manually tuned AB Prophet model is displayed in Figure 14. Manual tunings with similar codes were also attempted for the HO- and NH Prophet models, but the results from these models did not improve on the prediction results from the default models, which is stated in chapter 5.2.1 and 5.3.1.

```
#Setup, train and fit model
model = Prophet(growth='logistic', seasonality_mode='multiplicative',
    weekly_seasonality=20, daily_seasonality=60)

#Setting cap and floor for logistic growth in training data
AB_prUnit_train['cap']=AB_prUnit_train['Pvv pr unit [W]'].max()
AB_prUnit_train['floor']=0

#Fitting model to training data
#..and performing necessary formatting so the model understands the column
    values in training set
model.fit(AB_prUnit_train.reset_index() \
            .rename(columns={'DateTime':'ds',
                             'Pvv pr unit [W]':'y'}))

#Performing necessary formatting so the model understands the column values in
    test set
df=AB_prUnit_test.reset_index() \
                                .rename(columns={'DateTime':'ds'})
#Setting cap and floor for logistic growth in test data
df['cap']=AB_prUnit_train['Pvv pr unit [W]'].max()
df['floor']=0

#Forecasting the test data
AB_prUnit_test_fcst = model.predict(df)

#Setting default coincident interval to not go below zero
#... as that is unrealistic
AB_prUnit_test_fcst['yhat_lower']=AB_prUnit_test_fcst['yhat_lower'].clip(lower=0)
```

Figure 14: Manually tuned AB Prophet model

It should be noted that the average profiles mentioned above, do not necessarily reflect the true seasonality of the data. There are also uncertainties in the data, this will be discussed in more detail in chapter 6

The manually tuned AB Prophet model is shown in Figure 14. This is the model that produced the AB Prophet results displayed in the Result chapter. Manual tuning did not improve the HO- or NH Prophet models, as the best results for these building categories came from the default model or the CV tuning process.

To gain more insight into how the Prophet model predicts the consumption, the hyperparameter values, both default, manually tuned, and tuned by the CV process, can be visualized by a component plot. This plot is created by this line of code

```
model.plot_components(AB_prUnit_test_fcst).
```

The resulting component plots from this code can be seen in chapter 5.

A default, country specific, holiday effect may be added to the Prophet models by adding this single line of code;

```
model.add_country_holidays(country_name='NO').
```

This line of code adjusts for somewhat increased consumption during the national holidays. The default holiday effect improved the Prophet predictions slightly for AB and NH in form of a reduced RMSE, but had no effect on the HO Prophet predictions. The effect of the holiday component in the AB- and NH Prophet models can be seen in Figure 20 and Figure 31, respectively. The Prophet package includes functionality for creating self-defined holiday effects, but this was not attempted in this thesis. The reason for this is that the holidays did not seem to be an important error factor as the default holiday component had little effect. Also, from sorting the hourly absolute prediction errors in descending order, the holidays do not seem to dominate the top of the list. This can be seen in Appendix D, where the 10 largest absolute error values for the hourly prediction made by the best performing AB-, HO- and NH Prophet models without holiday effects are shown inTable 10, Table 11 and Table 12 respectively. The holidays in Norway in 2019 (the year the predictions were conducted for) are shown in Table 13 in Appendix E for comparison with the worst predicted hours.

4.4.1 Cross Validation Tuning of the Prophet Models

The Prophet package includes functionality for cross-validation to measure forecast error. This functionality is explained in [29]. The cross validation is performed automatically using the "cross_validation"-function. The function takes in a parameter, "horizon", which specifies the length of the forecast horizon. The function then defaults selects the training period, "initial", to be 3 times longer than the forecast horizon. The dates where the training period ends and the forecast period starts are called "cutoff" dates. The spacing between the cutoff dates is connected to the parameter "period", which by default is set to half of the forecast horizon. The result of the "cross_validation"-function is a Pandas Dataframe with the observations (y) and the out-of-sample predictions (yhat) for every timestamp from cutoff to cutoff+horizon.

In this thesis, all cross-validation is used for hyperparameter tuning. The horizon is set to be the same amount of days as in the test data in the manual set-up. The choice of hyperparameters to tune, and the combination of values are the same for all three building types' Prophet models. The grid which the hyperparameter combinations are to be made of is shown in Figure 15. The entire Python code for the CV tuning is shown for the AB data in Figure 37 in Appendix B. The CV tuning code has the same set-up for all building categories, and therefore, only the AB CV Python code is included in the appendix.

```
param_grid = {
    'changepoint_range' : [0.8, 0.95],
    'changepoint_prior_scale': [0.001, 0.01, 0.1],
    'seasonality_prior_scale': [0.1, 1.0, 10],
    'weekly_seasonality': [0, 10.0, 50, 100],
    'daily_seasonality': [0, 10.0, 50, 100],
}
```

Figure 15: Code for creating hyperparameter grid (see Figure 37 for full CV code)

According to [29], the parameters that most likely could make a better model by tuning them is "changepoint_prior_scale", "seasonality_prior_scale", "holiday_prior_scale" and "seasonality_mode". However, in contrast to this advice, the best results accomplished by the Prophet method were achieved by tuning the "Parameters that would likely not be tuned" [29], such as "weekly_seasonality" and "daily_seasonality". The reason for this is that the default Prophet model does not overfit the training data, the default model is conservative

in the sense that it fails to capture the highs and lows in the daily variations. Empirical evidence in the work of this thesis suggests that the hyperparameters, "weekly_seasonality" and "daily_seasonality", affecting the length of the Fourier series in the seasonal components, could be tuned to improve predictions. This is why these hyperparameters are included in the parameter grid used in the CV tuning process, which as mentioned is shown in Figure 15.

All different combinations of hyperparameters shown in Figure 15, were tested in all three Prophet models. The results from the cross validation hyperparameter tuning are shown in the individual parts of chapter 5. It should be pointed out that there are 288 different combinations of hyperparameters in the code I have used. Therefore, the CV codes take several hours to run on my computer. This long running time limits the number of hyperparameters I am able to tune, and the range of values to tune for each hyperparameter.

4.5 XGBoost

XGBoost is an open-source software. XGBoost was installed from the XGBoost library in Python.

Individual XGBoost models are trained for each of the three different building types. The model is then used to predict consumption for each building type on the test sets. This section will go into detail on how the XGBoost models were built.

The data handling for the XGBoost model is pretty much the same as for the Prophet model. However, the data is not divided by the number of units, as the number of units, along with all other variables is taken into the model as explanatory variables. A function retrieved from [30] creates features from the timestamp index and the other variables in the data. The Python code used for this purpose is shown in Figure 16.

```
#Creating features from DateTime
def create_features(df, label=None):
    df = df.copy()
    df['date'] = df.index
    df['Hour'] = df['date'].dt.hour
    df['dayofweek'] = df['date'].dt.dayofweek
    df['quarter'] = df['date'].dt.quarter
    df['month'] = df['date'].dt.month
    df['year'] = df['date'].dt.year
    df['dayofyear'] = df['date'].dt.dayofyear
    df['dayofmonth'] = df['date'].dt.day
    df['weekofyear'] = df['date'].dt.weekofyear

    #Adding all the features to the X-variable
    X = df[['Hour','dayofweek','quarter','month','year',
           'dayofyear','dayofmonth','weekofyear', 'Guests', 'BookedRooms',
               'Area', 'Units']]
    if label:
    #adding target to the y-variable
        y = df[label]
        return X, y
    return X

#This function extracts the features and returns them to the X-variable,
    simultaneously setting the target as y

X, y = create_features(HOall_df, label='Pvv [W]')

features_and_target = pd.concat([X, y], axis=1)
```

Figure 16: Python code used to create features from timestamp index and other variables for the HO XGBoost model (code inspired by [30])

It should be noted that in the feature extraction for the HO XGBoost model(Figure 16), the variables "Guests" and "BookedRooms" are added as features. This is different than for the other building categories, which have no such variables in their belonging data sets.

The models are fitted on the training data. Setting up the XGBoost model itself is a simple line of code, which in this thesis is retrieved from [30]. The code is shown in Figure 17.

```
# Setup, train and fit model
model = xgb.XGBRegressor(n_estimators=1000)
model.fit(X_train, y_train,
       eval_set=[(X_train, y_train), (X_test, y_test)],
       early_stopping_rounds=50,
      verbose=False)
```

Figure 17: XGBoost model (model set-up from [30])

The only input parameter changed in the XGBoost models is the "n_estimators" parameter. This parameter limits the number of trees to be made by the model. After tuning this parameter, the result did not change considerably and the parameter had little to no effect on the prediction. The "early_stopping_rounds" parameter is made to prevent overfitting the training data, something that will increase the prediction error on the test set.

In order to visualize what features affect the DHW consumption the most, a feature importance plot is created by the code in Figure 18. This code is equal for all three building types, but the example shown in Figure 18 is for the HO XGBoost model. Here the function "plot_importance" simply sums up how many times each feature is used as a data split in the regression trees [30].

```
ax= plot_importance(model, height=0.9)
plt.savefig('HOall_XG_FeatImp.png', dpi=300, bbox_inches='tight')
plt.close()
```

Figure 18: Python code used to plot feature importance (code inspired by [30])

The XGBoost models in this thesis are simple, meaning not many parameters have been tuned to improve the prediction results. The results from the AB, HO, and NH XGBoost predictions are shown in chapter 5.1.4, 5.2.4 and 5.3.4, respectively.

4.6 Error Metrics

Model performance can be measured with different performance indicators. In this thesis, the models have been tuned to minimize the RMSE. In order to compare the performance of the two model types for the different building categories, the normalized metric NRMSE is also calculated for all the models. MSE, MAE and MAPE are calculated for comparison to previous work.

In all forecasting, the "Null prediction" works as a reference point in terms of minimum performance. In the case of the DHW load forecasting in this thesis, the null prediction is just using the mean of the true DHW load in the training set as the prediction for the DHW load in the test set.

The codes for calculating and storing the error metrics are built up the same way for all the models produced in this thesis. The code for calculating and storing the error metrics for the AB Prophet model is shown as an example in Figure 39 in Appendix G.

5 Results

In this chapter, the result of the prediction modeling will be presented for each building type. The DHW consumption in all three building types is predicted using the two different methods introduced in section 3.

NB! In all prediction plots in this chapter, the true values are shown as a scatter-plot (red dots in Prophet, orange dots in XGBOOST), and the prediction is shown as a blue line.

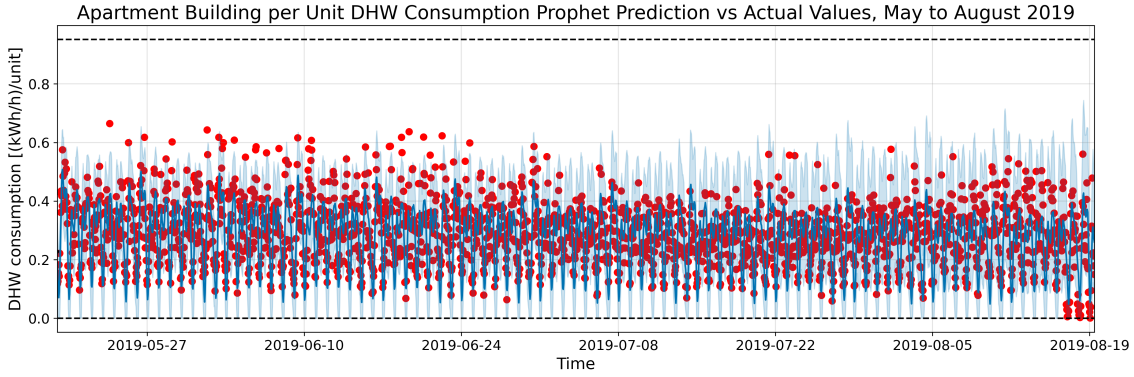
5.1 Apartment Buildings

The apartment building metadata contains the number of units and total floor area for each of the four buildings. The results for the Prophet prediction and the XGBoost prediction are presented in Figure 19 and Figure 23 respectively.

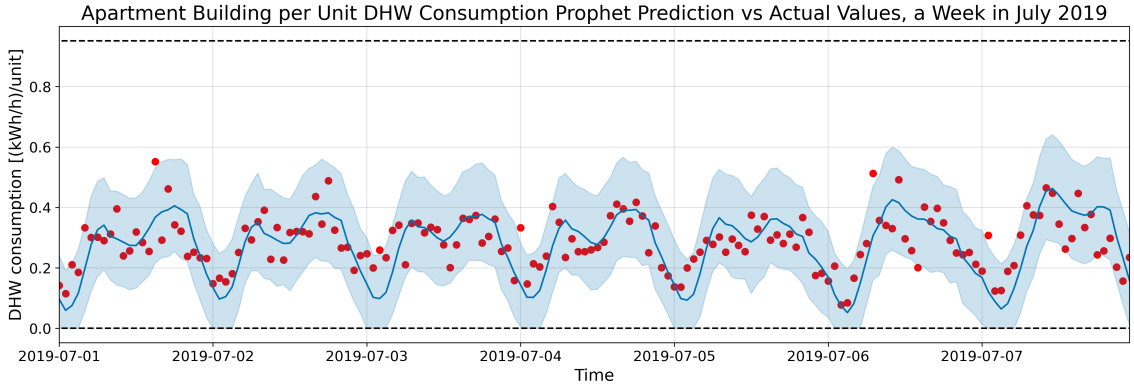
5.1.1 Apartment Building Prophet Prediction

This chapter will display prediction plots from the best performing AB Prophet model in terms of error metrics, which is the manually tuned AB Prophet model.

The consumption data fed into the Prophet prediction model is as mentioned divided by the number of units, so the consumption is in $[(\frac{kWh}{h})/unit]$. The manually tuned Prophet prediction for the apartment building consumption is plotted over the entire prediction period and for a single week in July 2019 in Figure 19 (a) and (b) respectively.



(a)



(b)

Figure 19: Apartment building per unit manually tuned Prophet DHW load prediction [(kWh/h)/unit], for (a) The entire prediction period (b) 1 week in July.

In Figure 19, the dotted horizontal lines are the logistic "floor" and "cap" values, which are set to zero and the highest consumption value in the training set, respectively. These lines are plotted when the "growth" parameter is set to "logistic". From visual inspection, the Prophet model predicts the AB consumption reasonably well. From the detail-studied week in July, shown in Figure 19(b), it can be observed that not many data points are outside the prediction confidence interval. Over the 3 month long prediction period, there are 105 true observations which are higher than the prediction upper confidence interval, and 24 true observations lower than the prediction lower confidence interval. There is a total of 2217 hourly true DHW load values in the test set, which means the 80% uncertainty interval of the prediction cover $\frac{2217-(24+105)}{2217} = 94\%$ of the true values.

The component plot from the manually tuned AB Prophet model is shown in Figure 20. The plot shows how the different components contribute to the final prediction.

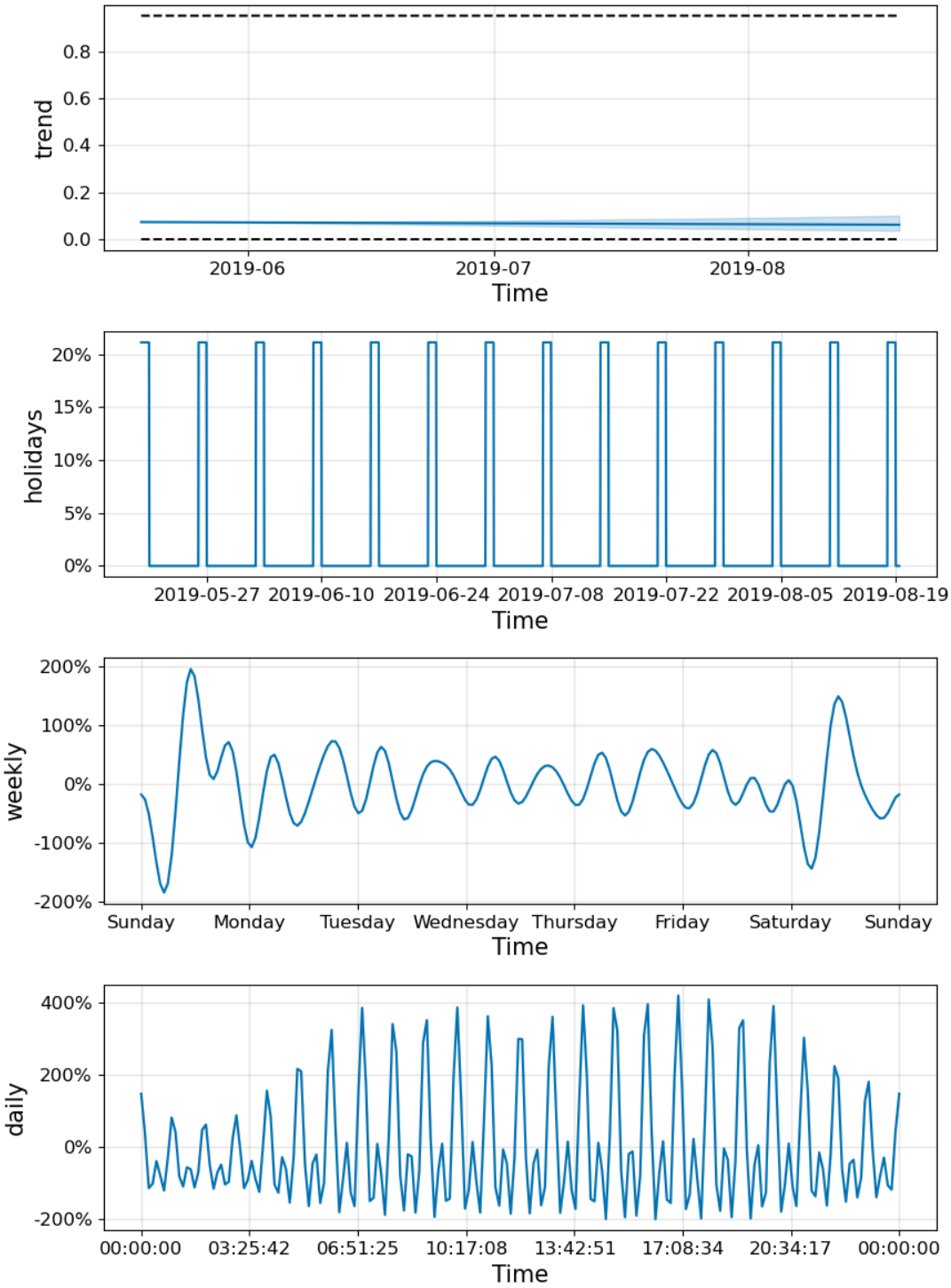


Figure 20: Components in the manually tuned AB Prophet model.

For example, from this component plot, we can read that the DHW load prediction for Monday 07/01-2019 at 07:00 would be

$\approx 0.063(\text{trend component}) + 0.6(\text{weekly component}) \cdot 0.063(\text{trend component}) + 4(\text{daily component}) \cdot 0.063(\text{trend component}) + 0(\text{holiday component}) \cdot 0.063(\text{trend component}) = 0.3528 \frac{kWh}{h}$.

In this calculation, the seasonal components are multiplied with the trend component, because in the tuned model, shown in Figure 14, the "seasonality_mode" parameter is set to 'multiplicative'. By default, the seasonality_mode is 'additive', which means the components are just added to the trend to produce the final prediction. With a multiplicative seasonality mode, each component value is multiplied by the trend component as shown in the calculations above. This calculation is made from visual inspection of the component plot and is therefore not very accurate. However, by inspecting the prediction for that particular time in Figure 19 (b), the calculation result seems to be close to the prediction.

The component plot for the default Prophet model is shown in Figure 21.

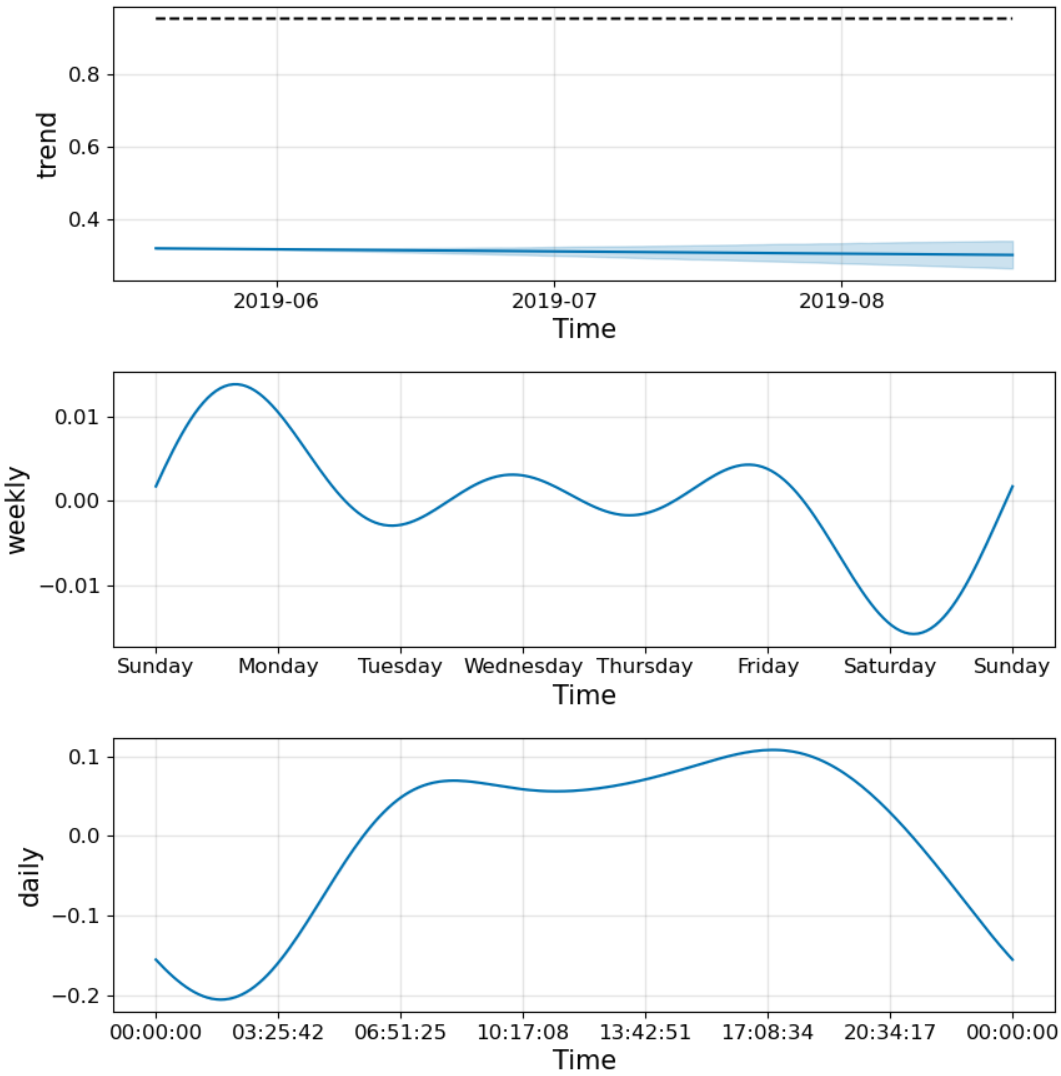


Figure 21: Components in the default AB Prophet model.

Comparing the two different component plots, the variations in the daily and weekly components are much more frequent and larger in magnitude for the manually tuned model than for the default model. The daily component in the default model is more similar to the average load curves shown in Figure 3. Since the manual tuned model predicts the DHW consumption more accurately according to all error metrics(shown in Table 3), it suggests that the true

DHW load varies frequently during a day and that the mean load profiles does not capture the true DHW load curve in an apartment building. It can be seen that the daily component has a top for every data point, and a low for the spacing between them, this seems unnecessary, but the result however shows that this seasonality captures the true load better than the default model which has more of an average profile consumption line.

5.1.2 Cross Validation Tuning of the Apartment Building Prophet Model

The output from the cross validation tuning for the AB Prophet model is shown in Figure 22. The output shows the optimal hyperparameters of the combinations made by the CV code in Figure 37 in Appendix B.

```
Best combination of hyperparameters: {'changepoint_range': 0.8,
    'changepoint_prior_scale': 0.1, 'seasonality_prior_scale': 0.1,
    'weekly_seasonality': 10.0, 'daily_seasonality': 10.0}

Models tested: 288
Script runtime: 2:36:45.176724
```

Figure 22: Python command window results for CV Tuning of AB Prophet Model.

The CV tuning process yields a set of hyperparameters to be tested in the manual Prophet model setup, using the same dates as the manually tuned and default Prophet models. The RMSE of this model, fitted with the hyperparameter values from the CV optimization, is shown in the "Hyperparameters from CV-tuning"-column in Table 3.

5.1.3 Error Metrics Comparison for Different Apartment Building Prophet Models

For comparison, the error metrics for the different AB Prophet models and the percentage change compared to the default AB Prophet model are displayed in Table 3.

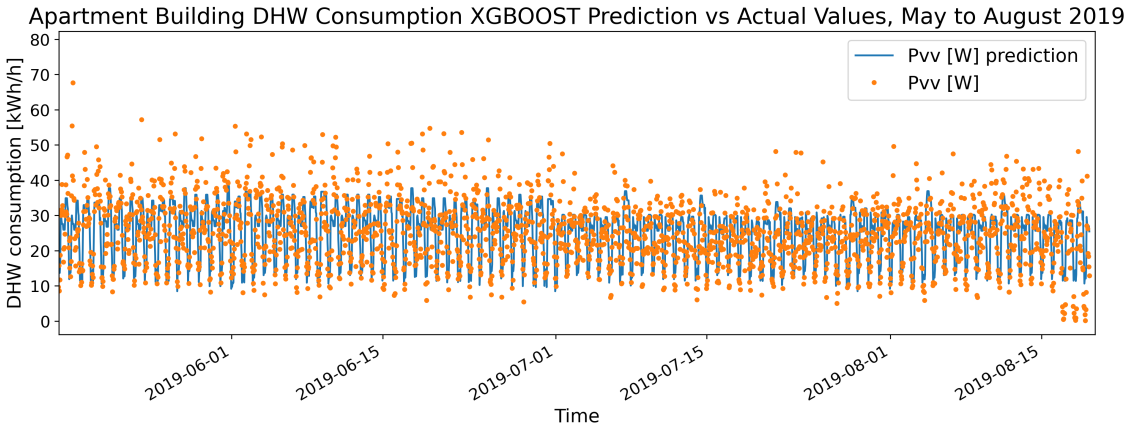
Table 3: Error metrics for different AB Prophet models and the percentage change compared to the default model.

AB Prophet models					
Model	Default	Hyperparameters from CV-tuning		Manually tuned	
Metric	Value	Value	Change compared to default [%]	Value	Change compared to default [%]
RMSE	0,088	0,093	6,5 %	0,081	-7,6 %
MAE	0,071	0,077	7,7 %	0,064	-10,8 %
MAPE	38,2	41,3	8,2 %	30,2	-20,9 %
NRMSE	0,301	0,320	6,5 %	0,278	-7,6 %

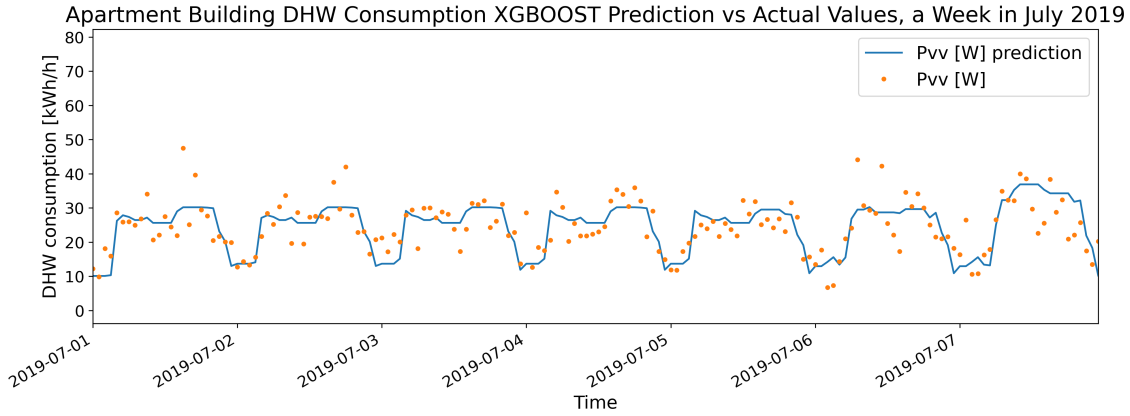
As mentioned, the manually tuned AB Prophet model is the best performing model for the apartment building category. Therefore, the predictions from the manually tuned AB Prophet model are plotted in Figure 19, and the error metrics from this prediction are included in Table 6.

5.1.4 Apartment Building XGBoost Prediction

The data fed into the XGBoost model is not the same as in the Prophet model. Here the consumption data is fed directly into the model along with the metadata, which includes the number of units and floor area in the buildings. The XGBoost prediction for the apartment building consumption is plotted over the entire prediction period and for a single week in July 2019 in Figure 23 (a) and (b) respectively.



(a)



(b)

Figure 23: Apartment building XGBoost DHW load prediction [kWh/h](a) Predicted period (b) 1 week in July.

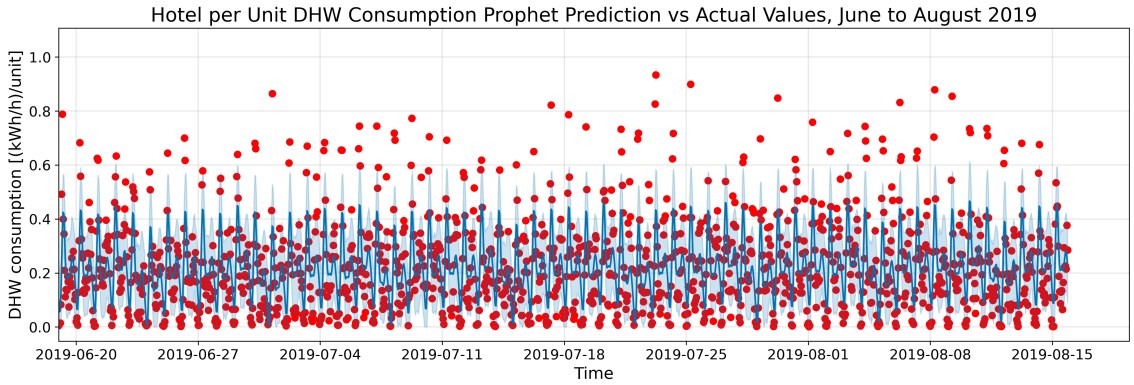
It can be seen by observing Figure 19 that the prediction curve follows the true DHW load values for most of the day. However, the model does not catch some of the afternoon load peaks, for example, the afternoon DHW load for July first and second, shown in Figure 19 (b). Also, the prediction of the weekend shown in Figure 19 (b) (07/06 and 07/07), is off. Compared to the HO and NH data, the AB data seems to have a flatter mid-day load curve, which the model seems to pick up on.

5.2 Hotels

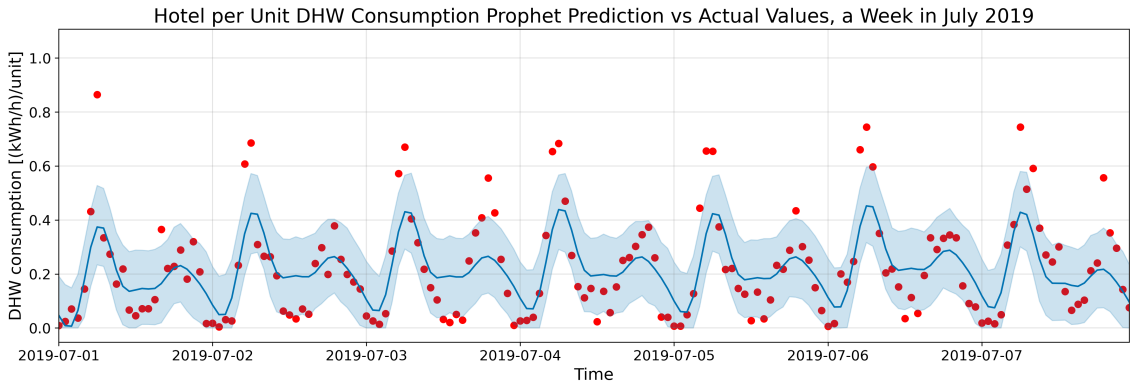
The hotel metadata contains the number of units, total floor area for each of the four hotels. In addition to this, there is also information on the number of booked rooms and the number of guests checked in at all timestamps. The results for the HO Prophet prediction and the XGBoost prediction are presented in Figure 24 and Figure 28 respectively.

5.2.1 Hotel Prophet Prediction

The consumption data fed into the Prophet prediction model is divided by the number of units, so the consumption is in $[(\frac{kWh}{h})/unit]$. The default HO Prophet prediction was the best performing HO Prophet model, so the prediction from this model is plotted over the entire prediction period and for a single week in July 2019 in Figure 24 (a) and (b) respectively. It must be pointed out that Prophet model cannot take into account the other explanatory variables in this data set. Another aspect regarding the HO Prophet model is that the Prophet model using per guest data actually performs better, the prediction plots and error metrics for this model are shown in Figure 38 and Table 14 in Appendix F. However, with the reasoning in chapter 4.1, this prediction model may not be as easy to use, since the number of guests information may not be available for future time values.



(a)



(b)

Figure 24: Hotel per unit default Prophet DHW load prediction [(kWh/h)/unit] for (a) The entire prediction period (b) 1 week in July.

It can be observed from Figure 24 that the consumption pattern seems to be captured by the model, meaning the peak load timing and the timing of the night-time low consumption seems to be correct. However, the model struggles to capture the highest and lowest consumption values. This is reflected when counting true observations outside of the prediction confidence interval. There are 160 true values over- and 52 values under the confidence interval, which makes for an uncertainty interval coverage of $\frac{1390-(160+52)}{1390} = 85\%$.

The components creating the default HO Prophet model are plotted in Figure 25.

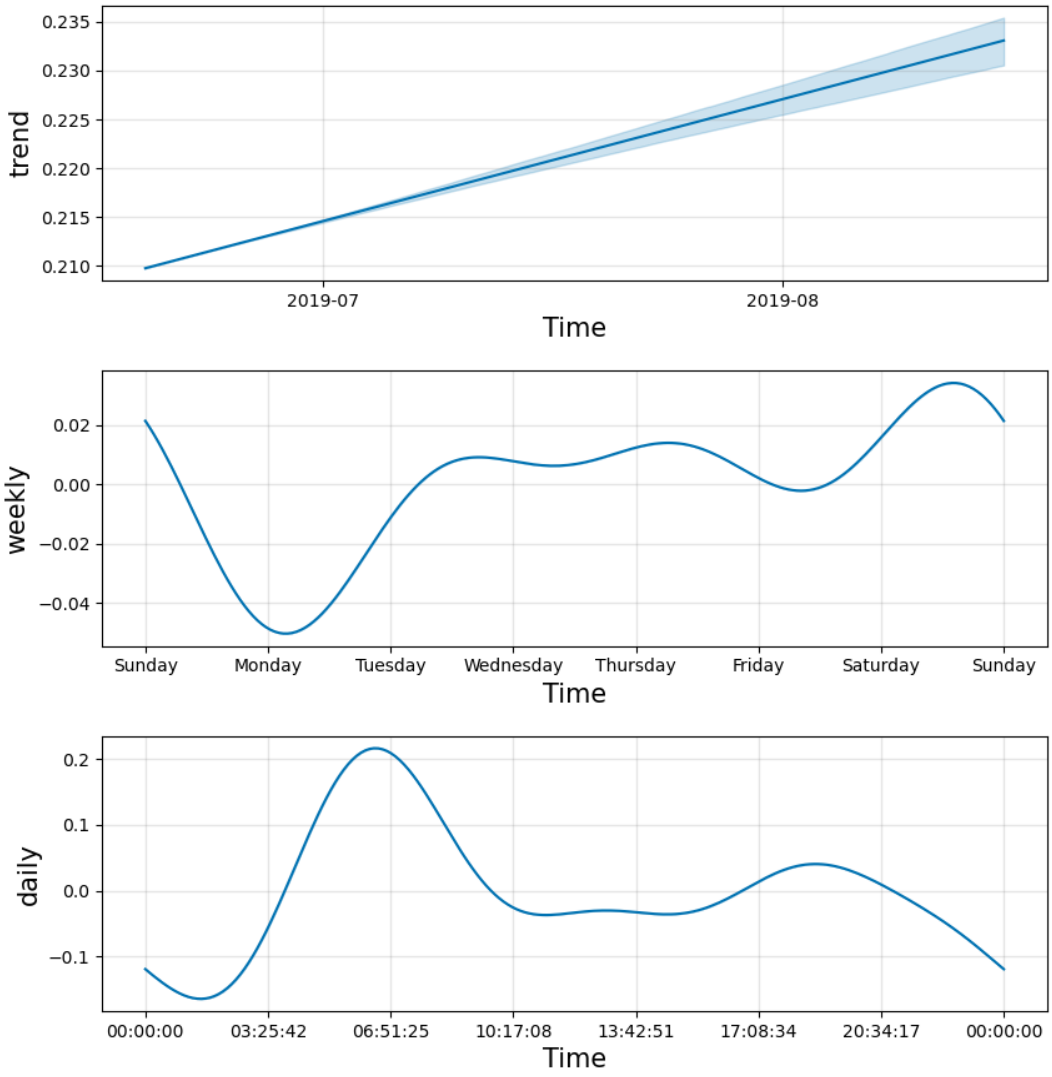


Figure 25: Components in the default HO Prophet model.

5.2.2 Cross Validation Tuning of the Hotel Prophet Model

The CV process for the HO Prophet model is done similarly as for the AB model where the code is displayed in Figure 37. The output from the HO CV-tuning can be seen in Figure 26.

```
Best combination of hyperparameters: {'changepoint_range': 0.8,
    'changepoint_prior_scale': 0.001, 'seasonality_prior_scale': 0.1,
    'weekly_seasonality': 50, 'daily_seasonality': 0}

Models tested: 288
Running time: 4:22:31.834759
```

Figure 26: Python command window results for CV Tuning of HO Prophet Model.

The running time of the script, shown in Figure 26, was over 4 hours, which is a problem in this thesis, as the models cannot be fully optimized due to lack of time and/or lack of computational power. This is further discussed in chapter 6.

The values of the hyperparameters from the CV optimization of the HO Prophet model is put into the manual Prophet model setup, and the components creating the CV-tuned HO Prophet model are plotted in Figure 27.

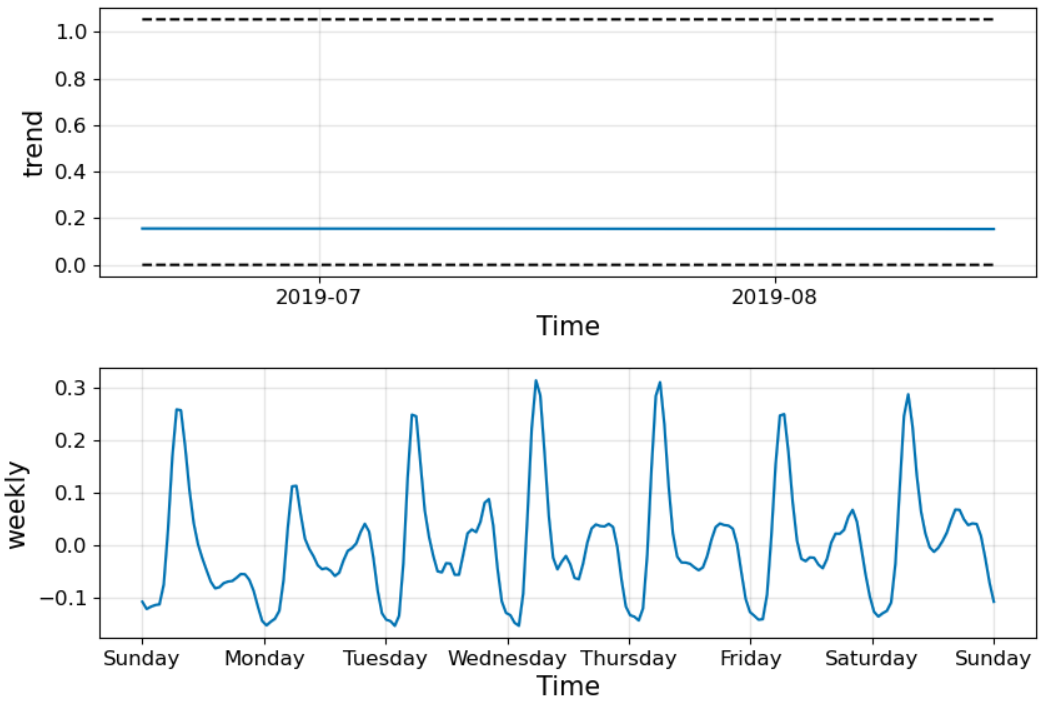


Figure 27: Components in the CV-tuned HO Prophet model.

As can be seen in Figure 26, the "weekly_seasonality" is optimized to 50, and the "daily_seasonality" is optimized to 0 in the CV-tuned model. The reason this produces the lowest RMSE of the combinations in the CV parameter grid is that the weekly seasonality is tuned up to 50 to capture both weekly and daily load variations in one Fourier series. This weekly seasonality component can be seen in Figure 27. However, the default model performs better according to RMSE, MAE and NRMSE, which can be seen in Table 4, where the CV-tuned model error metrics are compared to the error metrics from the default HO Prophet prediction. This paragraph might be confusing in terms of why the CV-optimization does not choose the default

model by setting all hyperparameters to default values, when this is the model that produces the lowest RMSE. The reason for this is that the parameter grid used in the CV-tuning process (shown in Figure 15), does not include all the default values for the different hyperparameters. Because of the long running time, all of the default values, along with many other values that could have been tested, are not included in this parameter grid.

5.2.3 Error Metrics Comparison for Different Hotel Prophet Models

To find out what version of the HO Prophet model performs the best predictions, the error metrics for the default HO Prophet model are compared to the error metrics from the CV-tuned HO Prophet model in Table 4.

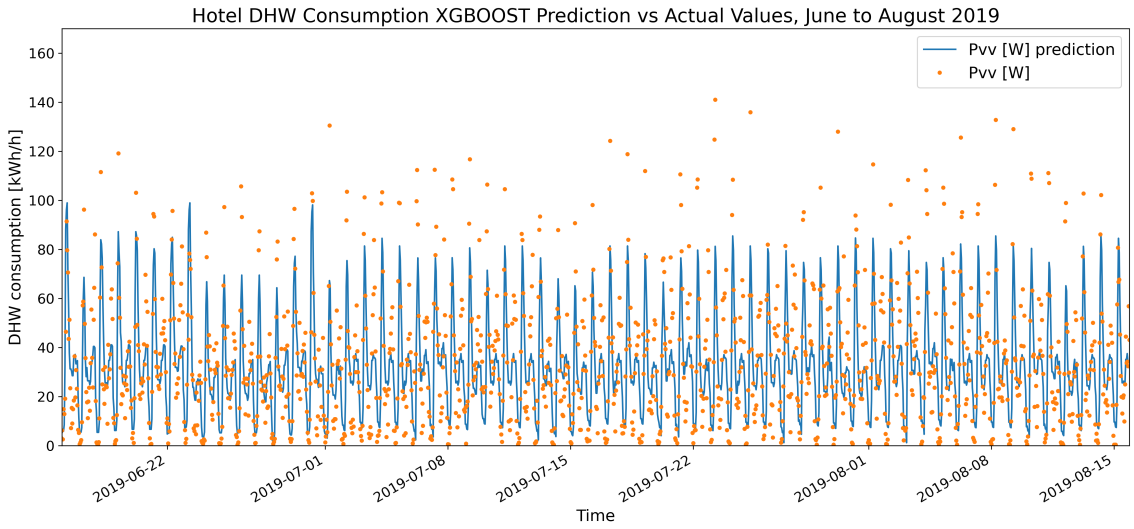
Table 4: Error metrics for different HO Prophet models and the percentage change compared to the default model.

HO Prophet models			
Model	Default	Hyperparameters from CV-tuning	
Metric	Value	Value	Change compared to default [%]
RMSE	0,112	0,130	15,7 %
MAE	0,085	0,091	6,8 %
MAPE	168,6	63,3	-62,4 %
NRMSE	0,491	0,568	15,7 %

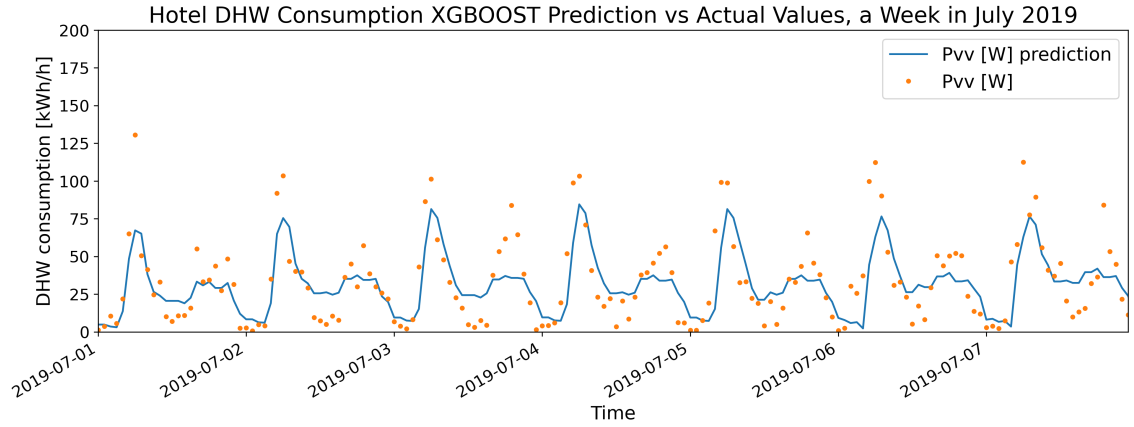
Manual tuning of the HO Prophet model did not improve the predictions either, and thus, the default Prophet model is the best performing HO Prophet model. Therefore the default HO Prophet prediction is plotted in Figure 24, and the error metrics from this prediction are displayed in Table 6.

5.2.4 Hotel XGBoost Prediction

The data fed into the XGBoost model is not the same as in the Prophet model. Here the consumption data is fed directly into the model along with the metadata, which includes the number of units and total area in the hotels. The XGBoost model also takes in the information about booked rooms and guests checked in at all times. The XGBoost prediction for the hotel DHW consumption is plotted over the entire prediction period and for a single week in July 2019 in Figure 28 (a) and (b) respectively.



(a)



(b)

Figure 28: Hotel XGBoost DHW load prediction [kWh/h](a) Predicted period (b) 1 week in July.

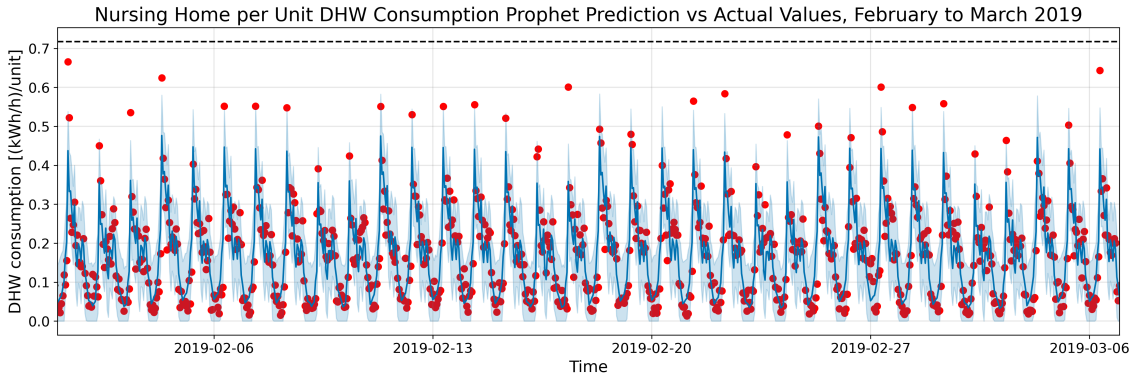
One might expect that the HO XGBoost model should be one of the top-performing models in this thesis as this is the model which takes in the most information in form of explanatory variables. However, as can be observed in Figure 28, the model fails to capture the height of the morning peak load and the mid-day low consumption values. The timing of the prediction however seems to be on point. The prediction values are conservative, but the curve of the prediction follows the curve of the true consumption.

5.3 Nursing Homes

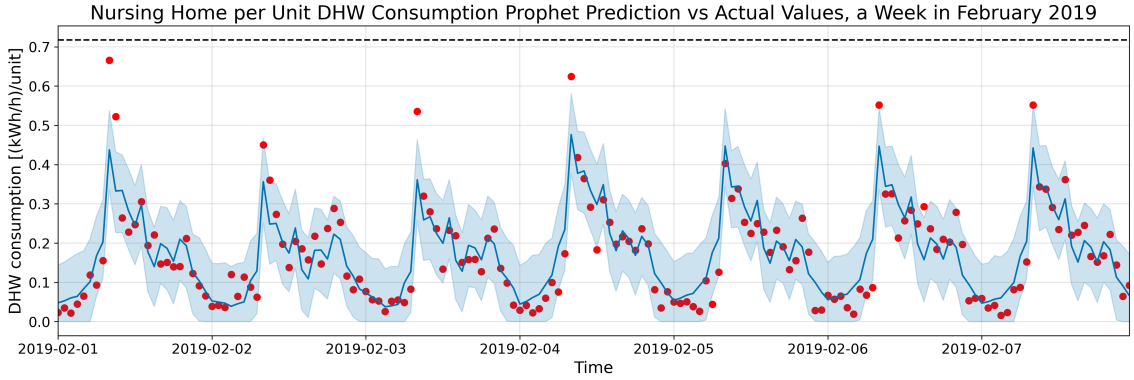
The nursing home metadata contains the number of units and total floor area for each of the four nursing homes. The results for the Prophet prediction and the XGBoost prediction are presented in Figure 29 and Figure 33 respectively.

5.3.1 Nursing Home Prophet Prediction

The consumption data fed into the Prophet prediction model is as mentioned divided by the number of units, so the consumption is in $[(\frac{kWh}{h})/unit]$. The Prophet prediction for the nursing home consumption is plotted over the entire prediction period and for a single week in February 2019 in Figure 29 (a) and (b) respectively.



(a)



(b)

Figure 29: Nursing home per unit CV tuned Prophet DHW load prediction [(kWh/h)/unit] for (a) The entire prediction period (b) 1 week in February.

From Figure 29 (b), it can be seen that the CV-tuned NH Prophet prediction performs good predictions as the prediction line is close to most of the true DHW load values. Only three peak load values in the first week of February can be visually separated from the uncertainty interval. From the entire prediction period, 33 true values are higher than the upper uncertainty interval, and 14 true values under the lower uncertainty interval. This means that the uncertainty interval covers $\frac{816-(33+14)}{816} = 94\%$ of the true values in the prediction period.

5.3.2 Cross Validation Tuning of the Nursing Homes Prophet model

The Cross validation process for the NH Prophet model is done similarly as for the AB model where the code is displayed in Figure 37. The output from the NH CV-tuning can be seen in Figure 30.

```
Best combination of hyperparameters: {'changepoint_range': 0.95,
    'changepoint_prior_scale': 0.1, 'seasonality_prior_scale': 0.1,
    'weekly_seasonality': 10.0, 'daily_seasonality': 50}

Models tested: 288
Running time: 5:50:16.031621
```

Figure 30: Python command window results for CV Tuning of NH Prophet Model.

The running time of this script was almost 6 hours. The optimal hyperparameters from the CV-tuning shown in Figure 30 are entered in a NH Prophet model using the manual Prophet model set-up. Meaning the same training- and test sets that are used to fit the default NH Prophet model.

The components creating the CV-tuned NH Prophet model are plotted in Figure 31.

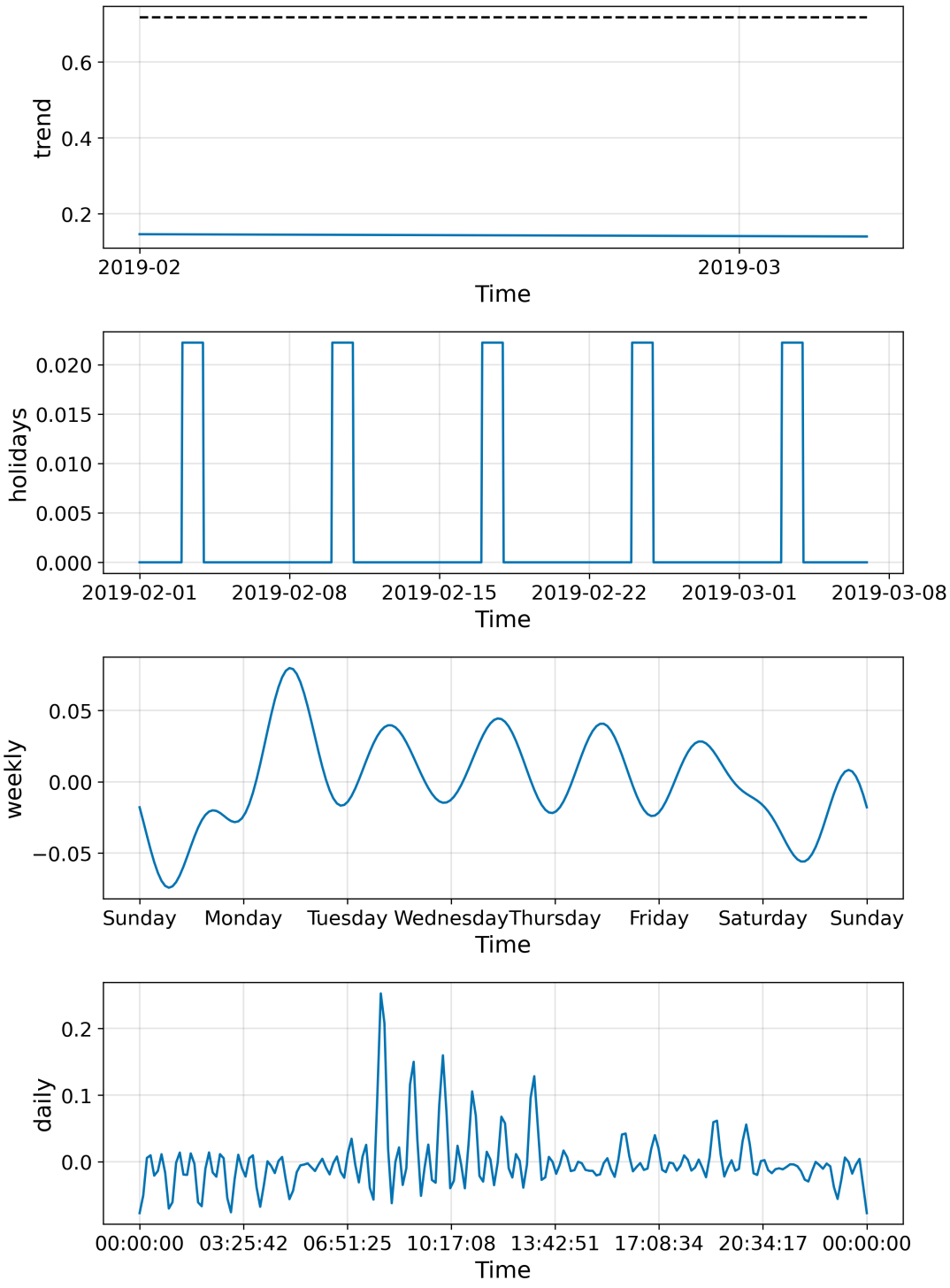


Figure 31: Components in the CV-tuned NH Prophet model.

For comparison, the components in the default NH Prophet model are plotted in Figure 32.

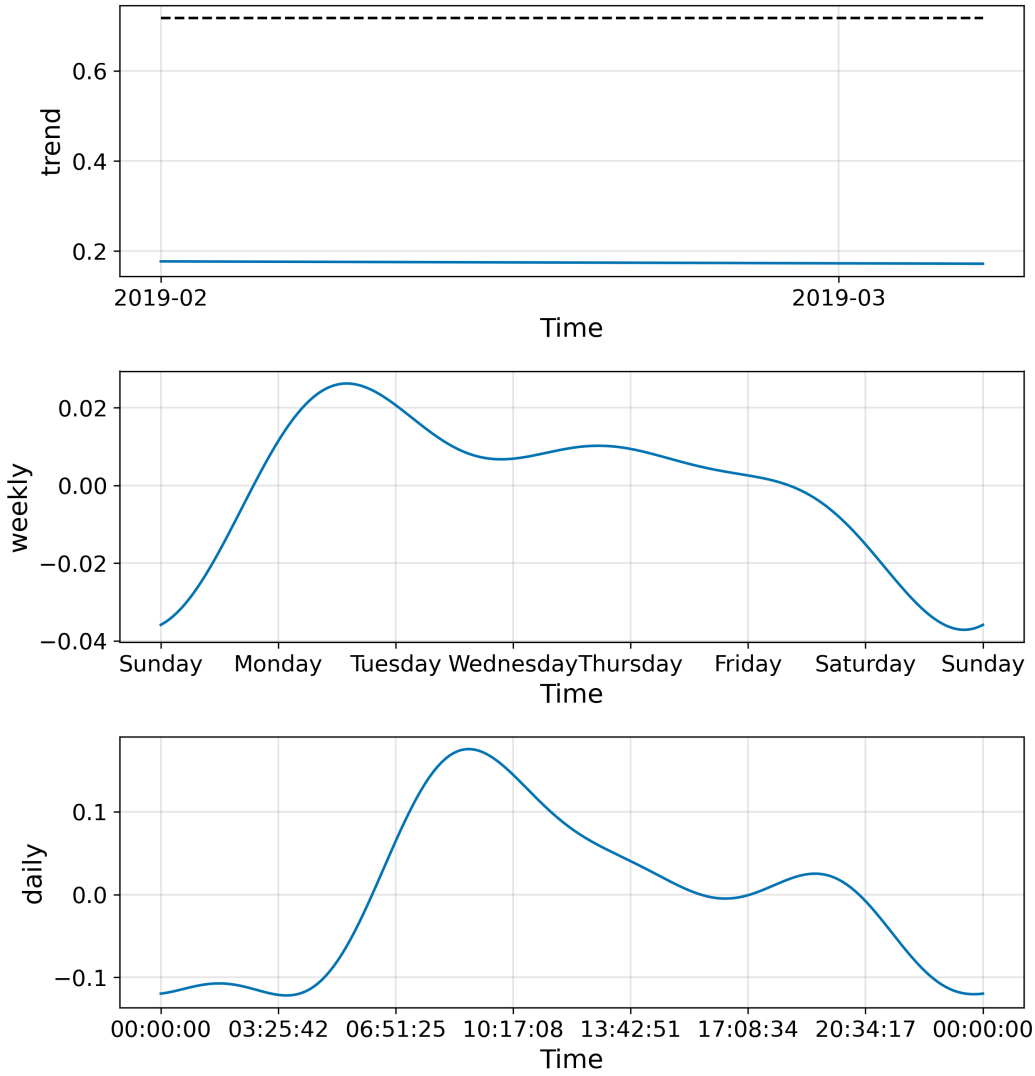


Figure 32: Components in the default NH Prophet model.

Both the weekly and daily seasonality have a higher value in the CV tuned model than in the default model. This can be seen by comparing the frequency and magnitude of the changes in the components in Figure 31 and Figure 32. As for the manually tuned AB Prophet model, the CV-tuned NH Prophet model has a daily seasonality component with high frequency changes, and it can be seen in Figure 31, that the daily component has a peak for every hour and a low for the gaps between the hourly values. One might think that this is a sign of overfitting since the data set only contains hourly values. However, the error metrics in Table 5 shows that this works better than the smooth daily component in the NH default Prophet model, shown in Figure 32.

5.3.3 Error Metrics Comparison for Different Nursing Home Prophet Models

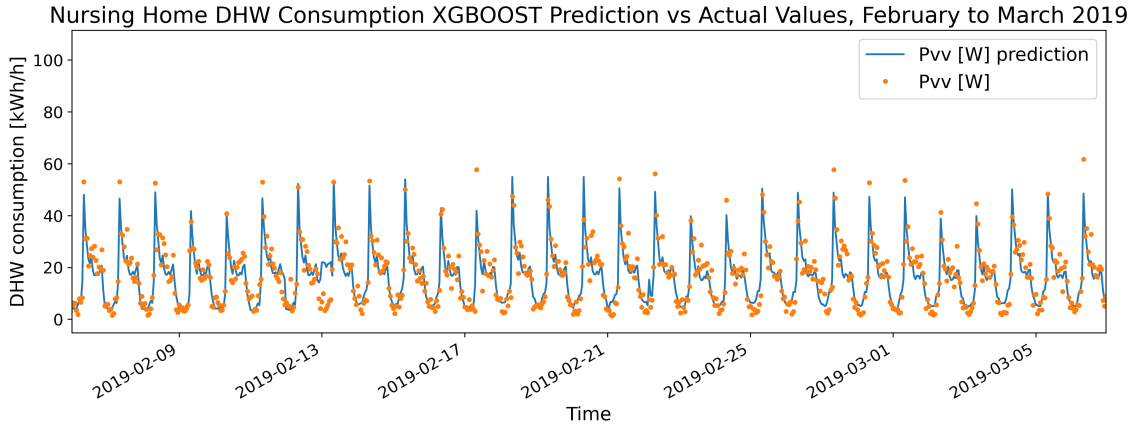
The error metrics from the CV-tuned NH Prophet model are compared to the error metrics from the default NH Prophet model in Table 5. Using the hyperparameters from the CV optimization process produces the best results. Therefore, the results in chapter 5.3.1 and in Table 6 comes from the model tuned with the optimal hyperparameters from the CV process.

Table 5: Error metrics for different NH Prophet models and the percentage change compared to the default model.

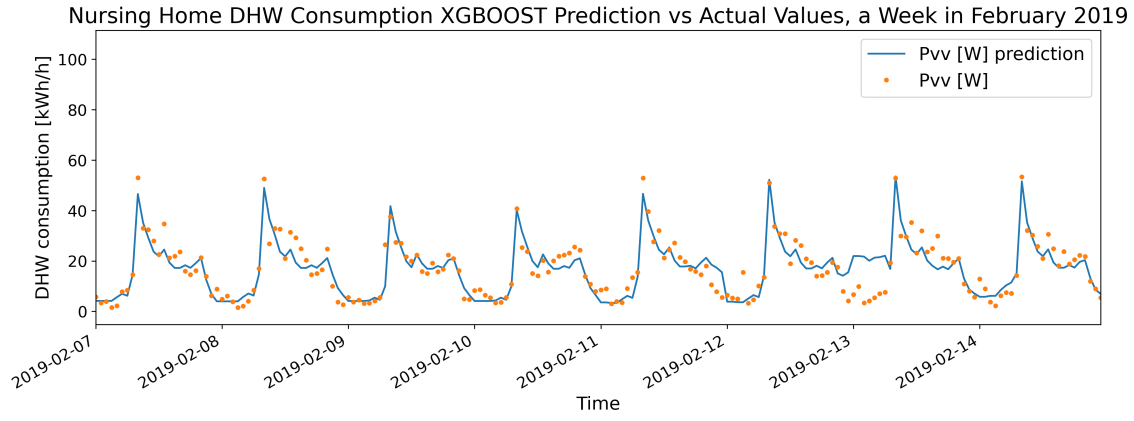
NH Prophet models			
Model	Default	Hyperparameters from CV-tuning	
Metric	Value	Value	Change compared to default [%]
RMSE	0,065	0,053	-19,2 %
MAE	0,047	0,040	-15,0 %
MAPE	44,1	38,6	-12,5 %
NRMSE	0,375	0,303	-19,2 %

5.3.4 Nursing Home XGBoost Prediction

The data fed into the XGBoost model is not the same as in the Prophet model. Here the consumption data is fed directly into the model along with the metadata, which includes the number of units and floor area in the nursing homes. The XGBoost prediction for the nursing home DHW consumption is plotted over the entire prediction period and for a single week in February 2019 in Figure 33 (a) and (b) respectively.



(a)



(b)

Figure 33: Nursing home XGBoost DHW load prediction [kWh/h](a) Predicted period (b) 1 week in February.

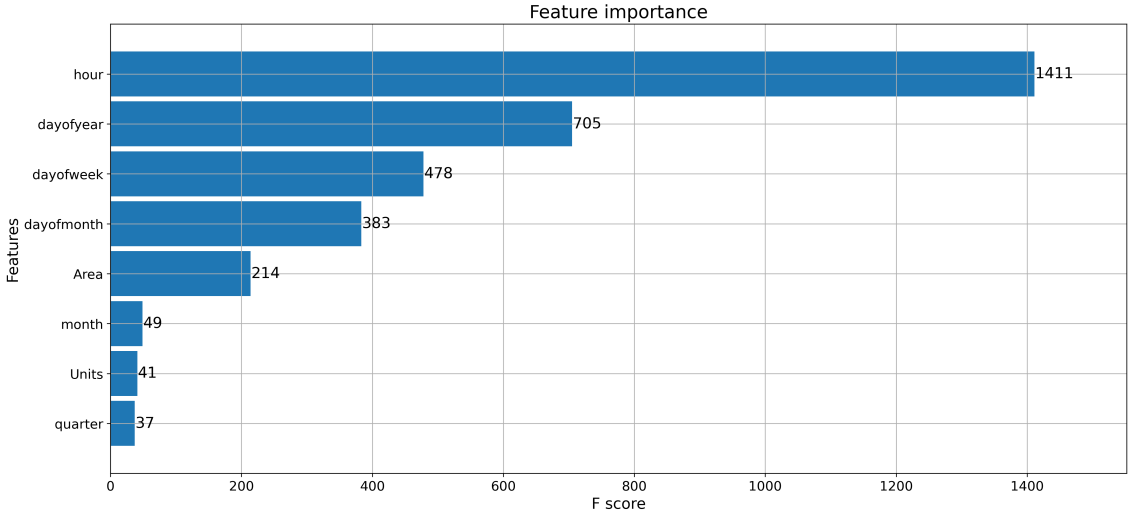
From visual inspection of Figure 33 (b), the NH XGBoost prediction looks good for most of the days, as the prediction line seems close to all observations. The greatest errors in the prediction of the second week in February are during the night hours of Friday the 13th.

5.4 Model Performances

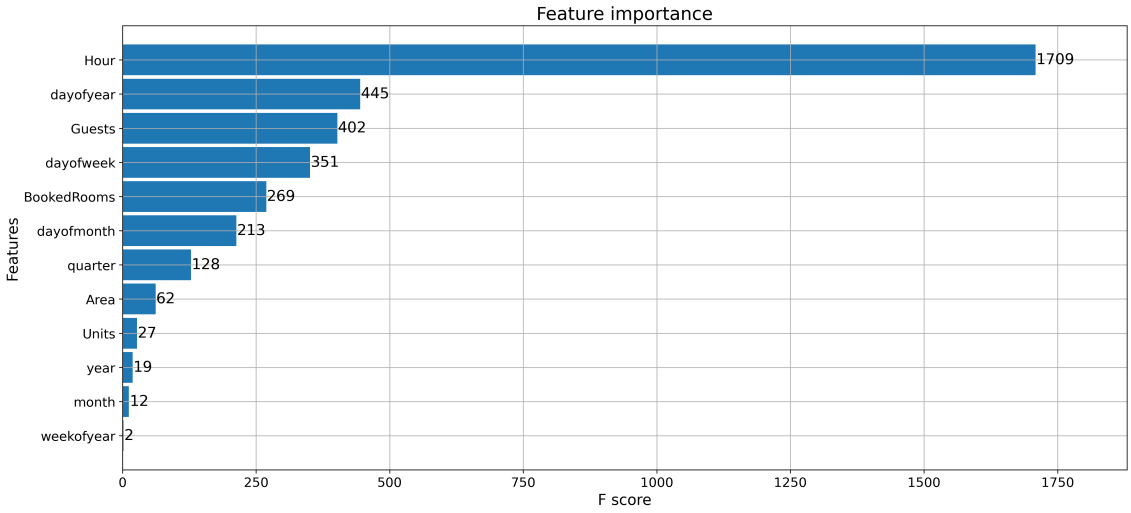
This section will evaluate the model performances on the different building types.

5.4.1 Feature Importance XGBoost

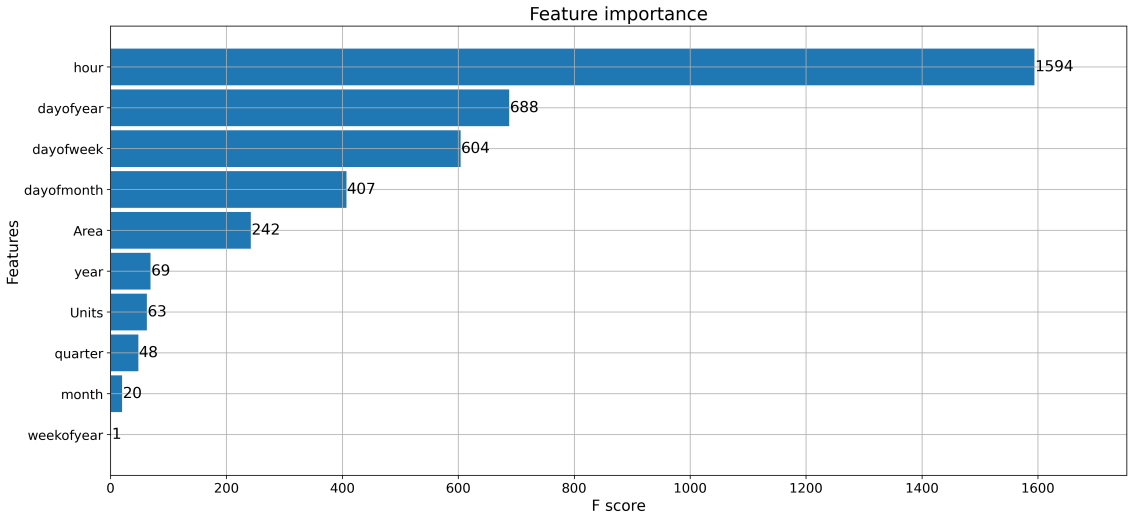
As mentioned, XGBoost takes in all variables as features. The importance that each feature has on the DHW consumption is measured and plotted in Figure 34.



(a) Apartment Buildings



(b) Hotels



(c) Nursing Homes

Figure 34: Feature importance XGBoost model for (a) Apartment buildings (b) Hotels (c) Nursing Homes.

As can be observed in Figure 34, the "hour-of-day" feature is the stand out important feature. This is also easily identified through looking at the plotted predictions as the daily variations are the dominating pattern in the consumption. The hourly count of the number of guests checked in is an important feature for the hotel DHW predictions. The data has been split 402 times on this feature in order to produce the final prediction. Interestingly the number of units is less important than the total area of the building for all building categories. This is contradictory to the findings in the Prophet part of the thesis. As the error made by the AB- and NH Prophet models using per m2 consumption is larger than the error made by the same Prophet models using per unit data. This is further discussed in chapter 6.

5.4.2 Error Metrics Summary

This chapter summarizes the error metrics for the best performing versions of both model types (Prophet and XGBoost) for all three building types.

In Table 6, the error metrics of the best performing versions of the two models are displayed for all three building categories. The error metrics for the NULL prediction for the different buildings are also displayed in the table for comparison. The NULL prediction is made by taking the mean of the training data. It should be pointed out that the NULL prediction uses the original data magnitude, meaning the consumption is not per unit.

Table 6: Error metrics table, best performing versions of the models compared to the NULL prediction.

Building	Model\Metric	MSE	MAE	MAPE	RMSE	NRMSE
AB	**Prophet**	0,007	0,064	30,20	0,081	0,278
	XGBoost	38,0	4,78	32,44	6,17	0,245
	NULL	102,3	8,03	54,85	10,11	0,402
HO	**Prophet**	0,013	0,085	168,62	0,112	0,491
	XGBoost	257,2	12,08	122,45	16,04	0,468
	NULL	756,3	22,40	513,78	27,50	0,802
NH	**Prophet**	0,003	0,040	38,62	0,053	0,303
	XGBoost	20,5	3,41	37,27	4,53	0,269
	NULL	141,0	9,37	108,58	11,87	0,706

The NRMSE error metrics for the different best performing models are visually displayed in a bar plot in Figure 35, which is made using the numbers in Table 6.

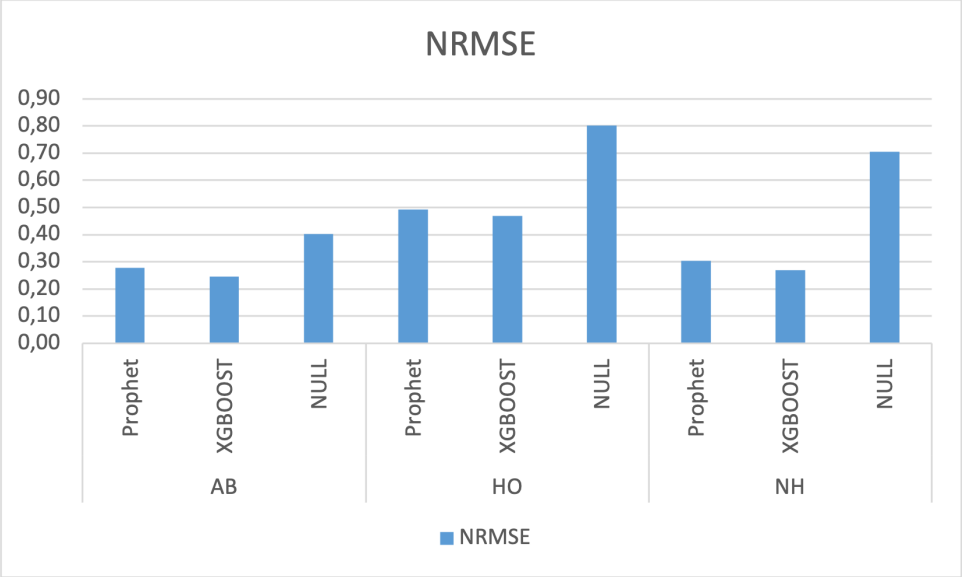


Figure 35: NRMSE bar plot for the best performing versions of the models compared to the NULL prediction for all three building categories, made by Table 6.

An important thing to keep in mind when looking at the error metrics in Table 6 is the magnitude difference between the predictions in the different models. As stated in section 4, the data fed into the Prophet model is in per feature format. The AB and NH data are in per unit format, while the HO data are in per guest format. Therefore, the MAE, MSE and RMSE from the Prophet model are not comparable to the same metrics from the XGBoost- or the NULL model.

Inspection of the normalized error metrics shows that the XGBoost model performs better both according to the MAPE- and NRMSE metric for hotels and nursing homes. For the apartment building DHW prediction, the Prophet and XGBoost perform approximately equal error metrics. The normalized error metrics for the AB Prophet model prediction are MAPE $\approx$ 30% and NRMSE $\approx$ 0.28. The normalized error metrics for the AB XGBoost model prediction are MAPE $\approx$ 32% and NRMSE $\approx$ 0.25.

6 Discussion

6.1 Model Performance

The XGBoost model seems to perform best on all building types according to the NRMSE in Table 6. This coincides with what Dmytro Ivanko found in his paper [9]. This result is also reflected in the MAPE, but for the apartment building models, the MAPE value is marginally lower for the Prophet prediction. Generally, the models made for the apartment buildings and nursing homes perform better than the models made to predict hotel DHW consumption. This can be seen both in the prediction plots and in the error metrics table in Table 6. The building category with the worst prediction result is the hotel category, where the XGBoost model performed a RMSE = 16.04 and a NRMSE = 0.47. The NRMSE indicates that the error is large, more precisely that the RMSE is almost 50% of the mean observed value.

Looking at the forecast results in terms of error metrics, the MAPE error metric can be strongly affected by observed DHW consumption values near zero, as previously mentioned in 5.4.2. It can be observed in Table 6 in the results for the HO models, that the MAPE value is much higher than the NRMSE. (The MAPE is given as a percentage, but the NRMSE is given as a fraction but can be seen as a percentage when multiplied with 100%.) When comparing the prediction plots for the HO data and the AB data in Figure 2, the hotels have more consumption values close to zero. The high MAPE values are likely to be caused by these low consumption values.

The apartment building models seem to benefit from the amount of training data compared to the other building categories, especially compared to the hotel data. One advantage of predicting DHW consumption in nursing homes compared to apartment buildings and hotels is that the residents in different nursing homes may be more alike than the residents in different hotels and apartment buildings. Another factor here is that the showering of the residents more or less follows a schedule. This makes the DHW consumption more repetitive and easier for a model to adapt. This could explain why the XGBoost nursing home model performs so well compared to other models in this thesis.

As shown in Figure 35 and Table 6, the simple XGBoost models outperform the tuned and adjusted Prophet models. The reason why the XGBoost models were not tuned is the time pressure of this thesis. As the initial results from the XGBoost models were so good compared to the Prophet models, tuning the Prophet models as an attempt to achieve decent prediction results were prioritized. Time pressure and focus on the Prophet models are the reason why the XGBoost models are not tuned to optimize the results.

6.2 Comparison to previous research

There are not a lot of previous works on predicting DHW consumption using machine learning techniques. The model performances on the hotel data in this thesis can however be compared to the work of Ivanko et al. in [9] which predicted the DHW consumption in HO4 which also is studied in this thesis. As stated in chapter 3.5, the best XGBoost model in this article performed a MAE of 3.12. However, this model was only fed data from the specific hotel for which the predictions were made. In this thesis, the goal was to create a general model able to predict consumption based on historical data from multiple other hotels. The MAE value for the best HO model in this thesis is 12.08, which is about 4 times larger. This can be explained by the disturbance from training data not from the specific hotel in which the consumption has

been predicted on. The conclusion for hotel DHW consumption is however that none of the models in this thesis were able to give good enough predictions to be used for smart control of DHW systems. Therefore, the method used in [9] seems more reliable and a better option for predicting hotel DHW consumption.

6.3 Data pre-processing

One thing I discovered while working with the prediction models is that how the data is pre-processed can have a large impact on the prediction results. Early in the process, the NAN values were set to zero (hereby referred to as PPS 1), and I thought that this would be the intuitive way to deal with missing data. However, using PPS 1 would yield a faulty image of the true consumption data. The NAN values should be removed, as there is no way of telling which value should have been where there is a NAN value. If the data rows with NAN values are removed totally from the data set(hereby referred to as PPS 2), the result improves.

It is also clear that some error metrics are more affected by these NAN values which were set to zero in PPS 1. As mentioned in chapter 3.4, the MAPE is heavily affected by consumption values close to zero, and when the consumption value actually is zero, the term becomes infinitely large. Another error metric that is affected by this is the NRMSE. This makes sense, as the NAN values in PPS 1 is set to zero, which is the lowest possible value allowed, which will reduce the mean consumption value in the test data. This again will result in a higher NRMSE, than if the values are removed. This can be seen easily by studying the formula for NRMSE in Equation 3.21. (To clarify, PPS 2 is the strategy used in this thesis, meaning NAN values were removed from the data.)

6.4 Data features

Among the variables available in the data, there is information about the number of units and the total area in all buildings analyzed in this thesis. The hotels also has information about the number of guests checked into the hotel, and the number of booked rooms at any time. The question is which of these variables tells us the most about the magnitude of the DHW consumption. According to [28], the DHW consumption has the strongest correlation to the number of residents in the building, then the heated area of the building, and lastly the number of units in the building.

Looking at Figure 34, this seems to be the correct assessment. As stated in chapter 5.4.1, the "Guests" feature in the HO data is more important to the model than the "Units", "BookedRooms" and "Area". Also, "Units" is less important than the "Area". This discussion is relevant also for the Prophet models, because if the feature importance plot in the XGBoost model is a factor to count on as a correlation measure, then the Prophet model may perform better when fed per m^2 data than per unit data. This however is not the case. For all building categories, the Prophet model performs better with per unit data, than with per m^2 data, this can be observed by looking in the error metric comparison tables in Appendix C

There could however be other explanations to these Prophet results. One thing to keep in mind about how the models are being fed the data is that the data is concatenated into a single dataframe, creating a false effect that all data comes from the same building. This could lead to a phenomenon where a difference in magnitude occurs at a time because subsequent data points are measurements from two different buildings with a different number of units or different building areas. The fact that per unit data performs better in the Prophet models could be because the magnitude difference because of number of unit differences are picked

up by the model as something else, for example that the difference are picked up as a change in weekday. The per m2 data does not necessarily have the same magnitude difference, and that may lead to the Prophet model not being able to pick up the change, and the model accuracy decreases because of this. Using the data in the way that is done in this thesis has such disadvantages as there can be hidden correlations and coincidences can play a part in how well the model fits the data in the end. However, if the models were to use data from only one apartment building, or one hotel, the results would not be very uniform, and the models would have less a portion of training data, or else the prediction periods would have to be shorter.

6.5 Model Identification Process

To further develop the Prophet models, taking in more data and performing cross validation tuning with more hyperparameters and more values for each hyperparameter is essential. It can be observed in Figure 22, Figure 26 and Figure 30 that the CV scripts take several hours to run, and therefore, choosing which parameters to tune were necessary in this thesis because of the time it takes to run the scripts. With more computational power and more time at hand, more of the hyperparameters, and a wider range of values for each of the hyperparameters in the models can be tuned to optimize the results. For example, an input parameter grid with all input parameters and a large range of values could have been created and run in a supercomputer. This would have made sure that the best input parameters are being used by the models, and this would probably increase the prediction accuracy of all the models in this thesis.

Weather data may also be added in the XGBoost model in order to increase the number of explanatory variables and possibly improve the prediction. This was not done in this thesis, as the geographical locations of many of the buildings were lacking.

7 Conclusion

In this thesis, Prophet and XGBoost machine learning models have been tested for medium-term DHW load prediction. The models have been set up, trained, and validated for apartment buildings, hotels, and nursing homes. The models are trained and validated on measurements from the Varmtvann2030 data set [10]. The Prophet models are trained only on DHW consumption data in per unit format and the time variables (year, quarter, month, week, day of the year, day of the week, hour), meaning no other explanatory variables were added to these models. The XGBoost models for the apartment buildings, hotels, and nursing homes utilize the number of units in the buildings and the total area of the buildings in addition to the time variables. The hotel XGBoost model also takes in the number of booked rooms- and the number of guests checked in at any time.

The XGBoost model produced strong prediction results for the apartment buildings and nursing homes with NRMSE around 0.25. The hotel consumption is not captured properly by either of the models in this thesis, which is clear from the fact that non of the models performed a NRMSE under 0.45. The differences in consumption magnitude between the different hotels may have played a part in this. The Prophet model predicts the apartment building data with a NRMSE of about 0.28. The XGBoost models in the thesis have not been tuned much, so they are easy to apply to new data, and perform the best predictions of the models studied in this thesis. XGBoost is a good option in terms of medium-term DHW load prediction for apartment buildings and nursing homes.

As mentioned above, and in chapter 6.1, the XGBoost models outperform the Prophet models. Even with the Prophet models being tuned and adjusted they did not perform as well as the close to default XGBoost models.

7.1 Learning Outcome

Going into this work I did not have a lot of experience with either data processing, machine learning techniques, or Python in general. So I have learned a lot about how to efficiently handle data, and to say the least, there is a lot I would handle differently from the start knowing what I know now. For example, spending more time analyzing the data before feeding it to the different prediction models would have been beneficial. Knowing the data in and out helps you realize where the model goes wrong, or what may cause problems in the prediction modeling.

Feeding per unit- or per guest consumption data uniformly into a model, is not the only way to go about this problem, and the shift in which building the data comes from could confuse the model, creating false seasonalities. My original thought was that the best way to take full advantage of the data sets was to train the models on as much data as possible, using per unit or per guest consumption data was a solution that allowed me to utilize data from all 4 buildings in each building category. However, the different buildings may have different types of residents with different habits of DHW consumption.

In hindsight, I believe that the task of predicting DHW consumption for all three building types with two different models turned out to be a little too ambitious. The tuning of the Prophet models took a lot of work, and gaining insight into both the Prophet model and XGBoost model was a lot of work for me, as I do not have a lot of experience in machine learning. The model results would probably be better if I could focus on making one model for one of the building types.

References

[1] *FME ZEN*. URL: https://fmezen.no/?lang=no (visited on 06/14/2021).

[2] *SINTEF Community - SINTEF*. URL: https://www.sintef.no/community/#/ (visited on 06/14/2021).

[3] Henrik Waterloo. "Domestic Hot Water Consumption Characteristics". Trondheim, 2020.

[4] IRENA. "Renewable Energy Target Setting". In: *Irena* June (2015), p. 80. URL: https://www.irena.org/publications/2015/Jun/Renewable-Energy-Target-Setting.

[5] K. B. Lindberg et al. "Long-term electricity load forecasting: Current and future trends". In: *Utilities Policy* 58 (June 2019), pp. 102–119. ISSN: 09571787. DOI: 10.1016/j.jup.2019.04.001. URL: https://doi.org/10.1016/j.jup.2019.04.001.

[6] Dag Spilde et al. *Strømforbruk mot 2040*. Tech. rep. Oslo, 2019, p. 24. URL: http://publikasjoner.nve.no/rapport/2019/rapport2019_22.pdf.

[7] NVE et al. *Langsiktig kraftmarkedsanalyse 2020-2040*. Oslo, 2020. URL: http://publikasjoner.nve.no/rapport/2020/rapport2020_37.pdf.

[8] *Total final consumption by source, OECD, 1971-2018 – Charts – Data & Statistics - IEA*. URL: https://www.iea.org/data-and-statistics/charts/total-final-consumption-by-source-oecd-1971-2018 (visited on 05/19/2021).

[9] Dmytro Ivanko, Åse Lekang Sørensen, and Natasa Nord. "Selecting the model and influencing variables for DHW heat use prediction in hotels in Norway". In: (2020). DOI: 10.1016/j.enbuild.2020.110441. URL: https://doi.org/10.1016/j.enbuild.2020.110441.

[10] "VarmtVann2030: Energi til tappevann i det norske lavutslippssamfunnet". In: *Data obtained from personal communication with Harald Taxt Walnum, SINTEF, in September 2020.* (). URL: https://www.sintef.no/projectweb/varmtvann/.

[11] Tao Hong et al. "Energy Forecasting: A Review and Outlook". In: *IEEE Open Access Journal of Power and Energy* 7.October (2020), pp. 376–388. DOI: 10.1109/oajpe.2020.3029979.

[12] Christian Behm, Lars Nolting, and Aaron Praktiknjo. "How to model European electricity load profiles using artificial neural networks". In: *Applied Energy* 277.July (2020). ISSN: 03062619. DOI: 10.1016/j.apenergy.2020.115564. URL: https://www.sciencedirect.com/science/article/pii/S030626192031076X.

[13] O Gerin, B Bleys, and K De Cuyper. "Domestic hot water consumption in apartment buildings". In: *International Symposium of CIB W062 Water Supply and Drainage for Buildings* 41 (2015), pp. 400–407. URL: https://www.irbnet.de/daten/iconda/CIB_DC30620.pdf.

[14] Linn Stengård and Thomas Levander. "Mätning av kall- och varmvattenanvändning i 44 hushåll". In: (2009). URL: https://energimyndigheten.a-w2m.se/Home.mvc?ResourceId=2418.

[15] *Prophet — Forecasting at scale*. URL: https://facebook.github.io/prophet/ (visited on 05/12/2021).

[16] Sean J Taylor and Benjamin Letham. "Forecasting at Scale". In: (Sept. 2017). DOI: 10.7287/peerj.preprints.3190v2. URL: https://doi.org/10.7287/peerj.preprints.3190v2.

[17] *Time Series Analysis with Facebook Prophet: How it works and How to use it — by Mitchell Krieger — Towards Data Science*. URL: https://towardsdatascience.com/time-series-analysis-with-facebook-prophet-how-it-works-and-how-to-use-it-f15ecf2c0e3a (visited on 06/14/2021).

[18] *Facebook Prophet. (Almost) everything you should know to... — by Moto DEI — The Startup — Medium*. URL: https://medium.com/swlh/facebook-prophet-426421f7e331 (visited on 05/20/2021).

[19] Jean Pierre Signoret and Alain Leroy. "Monte Carlo Simulation". In: *Springer Series in Reliability Engineering* August (2021), pp. 547–586. ISSN: 2196999X. DOI: 10.1007/978-3-030-64708-7{_}32. URL: https://www.researchgate.net/publication/326803384_MONTE_CARLO_SIMULATION.

[20] Tianqi Chen and Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System*. DOI: 10.1145/2939672.2939785. URL: http://dx.doi.org/10.1145/2939672.2939785.

[21] *XGBoost Mathematics Explained. A walk-through of the Gradient Boosted... — by Dimitris Leventis — Towards Data Science*. URL: https://towardsdatascience.com/xgboost-mathematics-explained-58262530904a (visited on 05/18/2021).

[22] Wei Yin Loh. "Classification and regression trees". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1.1 (Jan. 2011), pp. 14–23. ISSN: 19424795. DOI: 10.1002/widm.8. URL: https://onlinelibrary.wiley.com/doi/full/10.1002/widm.8%20https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.8%20https://onlinelibrary.wiley.com/doi/10.1002/widm.8.

[23] Amir Mosavi and Abdullah Bahmani. "Energy consumption prediction using m[1] A. Mosavi and A. Bahmani, "Energy consumption prediction using machine learning; a review," Energies, no. March, pp. 1–63, 2019." In: *Energies* March (2019), pp. 1–63. DOI: 10.20944/preprints201903.0131.v1. URL: https://www.preprints.org/manuscript/201903.0131/download/final_file.

[24] Filipe Rodrigues, Carlos Cardeira, and J. M.F. Calado. "The daily and hourly energy consumption and load forecasting using artificial neural network method: A case study using a set of 93 households in Portugal". In: *Energy Procedia*. Vol. 62. Elsevier Ltd, Jan. 2014, pp. 220–229. DOI: 10.1016/j.egypro.2014.12.383.

[25] Yibo Chen, Hongwei Tan, and Umberto Berardi. "Day-ahead prediction of hourly electric demand in non-stationary operated commercial buildings: A clustering-based hybrid approach". In: *Energy and Buildings* 148 (Aug. 2017), pp. 228–237. ISSN: 03787788. DOI: 10.1016/j.enbuild.2017.05.003.

[26] *Microsoft Azure Machine Learning Studio (classic)*. URL: https://studio.azureml.net/ (visited on 03/13/2021).

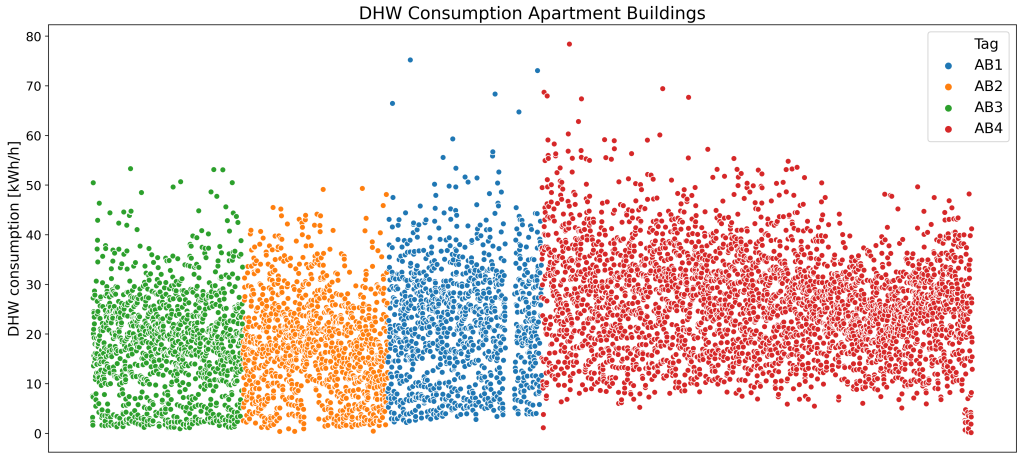
[27] *pandas - Python Data Analysis Library*. URL: https://pandas.pydata.org/ (visited on 05/06/2021).

[28] Dzintars Grasmanis, Aldis Grekis, and Normunds Talcis. "Heat Consumption Assessment of the Domestic Hot Water Systems in the Apartment Buildings". In: *Construction Science* 14.October 2013 (2013). ISSN: 2255-8551. DOI: 10.2478/cons-2013-0006.

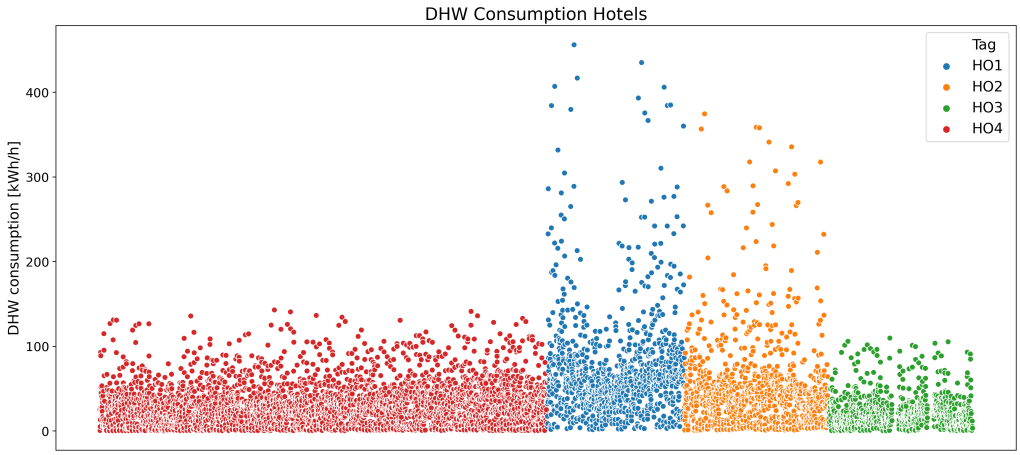
[29] *Diagnostics — Prophet*. URL: https://facebook.github.io/prophet/docs/diagnostics.html (visited on 05/25/2021).

[30] *[Tutorial] Time Series forecasting with XGBoost — Kaggle*. URL: https://www.kaggle.com/robikscube/tutorial-time-series-forecasting-with-xgboost/notebook (visited on 04/21/2021).

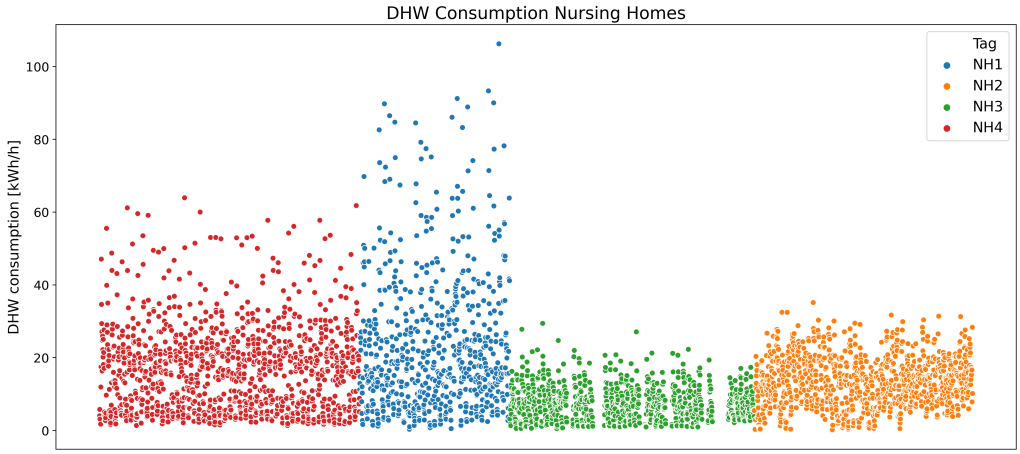
A Initial Data Without Time-axis



(a) Apartment Buildings



(b) Hotels



(c) Nursing Homes

Figure 36: Initial data plotted without timestamps for (a) Apartment Buildings (b) Hotels (c) Nursing Homes (data from Varmtvann2030).

B Cross Validation Prophet Code

```
#Cross validation

#Starting clock to track script running time
from datetime import datetime
startTime = datetime.now()

#Creating grid of hyperparameters
param_grid = {
    'changepoint_range' : [0.8, 0.95],
    'changepoint_prior_scale': [0.001, 0.01, 0.1, 0.5],
    'seasonality_prior_scale': [0.1, 1.0, 10],
    'weekly_seasonality': [0, 10.0, 50, 100],
    'daily_seasonality': [0, 10.0, 50, 100],
}

#Generating all combinations of hyperparameters
all_params = [dict(zip(param_grid.keys(), v)) for v in
    itertools.product(*param_grid.values())]
# For storing the RMSE's
rmses = []



#Using CV to evaluate all parameter combinations
for params in all_params:
    m = Prophet(growth='logistic', **params).fit(AB_prUnit_train.reset_index() \
            .rename(columns={'DateTime':'ds',
                             'Pvv pr unit [W]':'y'})) # Fit model with given
                                 params
    AB_prUnit_cv_tuned = cross_validation(m, horizon='92 days')
    AB_prUnit_cv_tuned_metrics = performance_metrics(AB_prUnit_cv_tuned,
        rolling_window=1)
    rmses.append(AB_prUnit_cv_tuned_metrics['rmse'].values[0])

#Finding and printing the best hyperparameter combination
ABtuning_results = pd.DataFrame(all_params)
ABtuning_results['rmse'] = rmses
print(ABtuning_results)
ABtuning_results.to_csv('AB_Prophet_CVtuning.csv')

#Printing running time
print(datetime.now() - startTime)
```

Figure 37: Code for cross validation tuning of AB Prophet model, same procedure is used for HO- and NH Prophet models (code inspired by [29])

C Per Feature Prophet Error Metrics Comparison

Table 7: Error metric comparison for different data fed into default AB Prophet model

Default AB Prophet models			
Data format	pr unit	pr m2	
Metric	Value	Value	Change compared to pr unit [%]
RMSE	0,088	0,002	NOT COMPAREABLE
MAE	0,071	0,001	NOT COMPAREABLE
MAPE	38,2	42,6	11,6 %
NRMSE	0,301	0,330	9,9 %

Table 8: Error metric comparison for different data fed into default HO Prophet model

Default HO Prophet models							
Model	pr unit	pr m2		pr guest		pr booked room	
Metric	Value	Value	Change compared to default [%]	Value	Change compared to default [%]	Value	Change compared to default [%]
RMSE	0,112	0,002	NOT COMPAREABLE	0,080	NOT COMPAREABLE	0,145	NOT COMPAREABLE
MAE	0,085	0,002	NOT COMPAREABLE	0,062	NOT COMPAREABLE	0,107	NOT COMPAREABLE
MAPE	168,6	158,6	-6 %	116,9	-30,7 %	104,8	-38 %
NRMSE	0,491	0,488	-1 %	0,473	-3,7 %	0,522	6 %

Table 9: Error metric comparison for different data fed into default NH Prophet model

Default NH Prophet models			
Data format	pr unit	pr m2	
Metric	Value	Value	Change compared to default [%]
RMSE	0,065	0,001	NOT COMPAREABLE
MAE	0,047	0,000	NOT COMPAREABLE
MAPE	44,1	44,8	1 %
NRMSE	0,375	0,399	7 %

D Worst Prediction Errors for Best Prophet Models

Table 10: AB manually tuned Prophet model, 10 worst prediction hours

Timestamp	True value	Prediction	Error	Absolute error
2019-05-29 05:00:00	0,60180125	0,2865686	0,31523265	0,31523265
2019-05-29 04:00:00	0,50385159	0,19938544	0,30446615	0,30446615
2019-06-22 07:00:00	0,62278691	0,31991641	0,30287051	0,30287051
2019-06-19 08:00:00	0,6369537	0,35038108	0,28657262	0,28657262
2019-05-31 04:00:00	0,47812062	0,19486283	0,28325779	0,2832577
2019-06-21 05:00:00	0,5572436	0,27630997	0,28093365	0,28093365
2019-06-01 08:00:00	0,64339547	0,37361293	0,26978254	0,26978254
2019-06-10 09:00:00	0,57833579	0,31352675	0,26480904	0,26480904
2019-06-20 04:00:00	0,46005812	0,19965488	0,26040323	0,26040323
2019-06-23 07:00:00	0,53346513	0,2816131	0,25185203	0,25185203

Table 11: HO default Prophet model, 10 worst prediction hours

Timestamp	True value	Prediction	Error	Absolute error
2019-07-23 06:00:00	0,93393061	0,43317444	0,50075617	0,50075617
2019-07-01 06:00:00	0,8643162	0,37407979	0,49023641	0,49023641
2019-07-23 05:00:00	0,82591449	0,35716463	0,46874986	0,46874986
2019-07-25 06:00:00	0,89968126	0,44703231	0,45264895	0,45264895
2019-06-19 05:00:00	0,78866055	0,35072287	0,43793768	0,43793768
2019-08-08 06:00:00	0,879284	0,45266882	0,42661518	0,42661518
2019-08-09 06:00:00	0,85462951	0,43734479	0,41728472	0,41728472
2019-07-30 06:00:00	0,84783986	0,43599269	0,41184716	0,41184716
2019-08-06 06:00:00	0,83169023	0,43881095	0,39287928	0,39287928
2019-07-22 05:00:00	0,69657371	0,30761848	0,38895523	0,38895523

Table 12: NH CV tuned Prophet model, 10 worst prediction hours

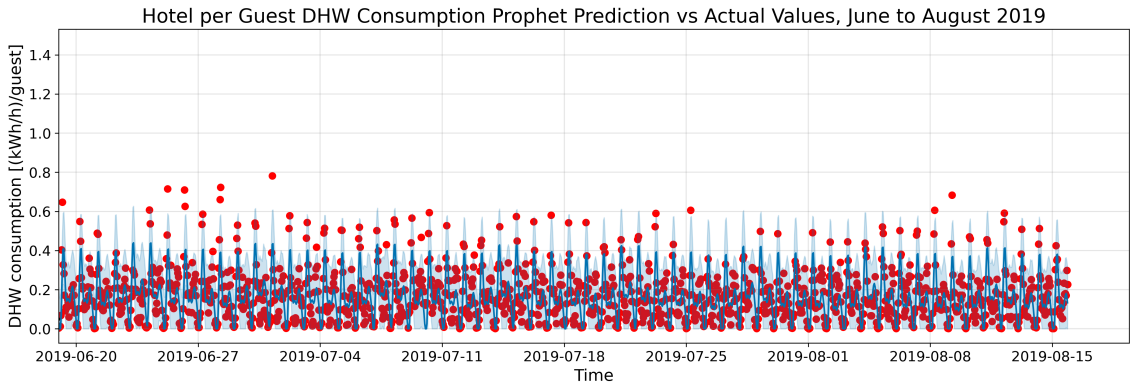
Timestamp	True value	Prediction	Error	Absolute error
2019-02-17 08:00:00	0,60114009	0,35857396	0,24256613	0,24256613
2019-02-01 08:00:00	0,66563479	0,43747703	0,22815777	0,22815777
2019-03-06 08:00:00	0,64325663	0,44194905	0,20130758	0,20130758
2019-02-16 09:00:00	0,44152519	0,24595579	0,1955694	0,1955694
2019-02-23 15:00:00	0,2986779	0,10530077	0,19337713	0,19337713
2019-02-01 09:00:00	0,52266063	0,33272911	0,18993152	0,18993152
2019-02-03 08:00:00	0,53567874	0,36101216	0,17466658	0,17466658
2019-02-27 08:00:00	0,60124643	0,44316815	0,15807829	0,15807829
2019-02-20 14:00:00	0,35260636	0,19513395	0,15747241	0,15747241
2019-02-21 14:00:00	0,34663092	0,18968	0,15695092	0,15695092

E Norwegian Holidays 2019

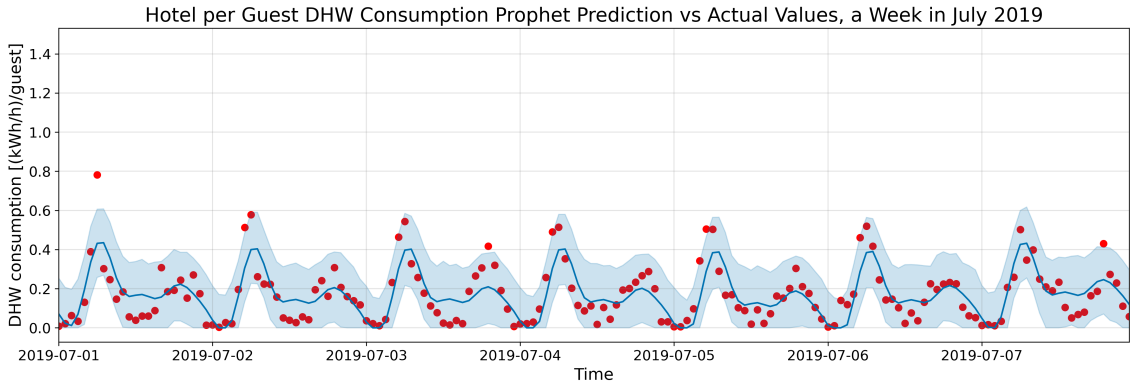
Table 13: Norwegian Holidays 2019

Date	Holiday Name
2019-01-01	New Year's Day
2019-04-18	Maundy Thursday
2019-04-19	Good Friday
2019-04-21	Easter
2019-04-22	Easter Monday
2019-05-01	Labour Day
2019-05-17	Constitution Day
2019-05-30	Ascension Day
2019-06-09	Pentecost Sunday
2019-06-10	Whit Monday
2019-12-25	Christmas Day
2019-12-26	Second Day of Christmas

F HO per Guest Prophet Predictions



(a)



(b)

Figure 38: Hotel per guest default Prophet prediction [(kWh/h)/guest] for (a) The entire prediction period (b) 1 week in July.

Table 14: HO per guest default Prophet prediction error metrics

HO per Guest Prophet Prediction Error Metrics	**Value**
Mean Squared Error	0,00626405
Mean Absolute Error	0,0597389
Mean Absolute Percentage Error	85,8155651
Root Mean Squared Error	0,07914575
Normalized Root Mean Squared Error	0,46669041

G Python Code for Calculating and Storing Error Metrics

```
#Error metrics
#Mean squared error
AB_prUnit_test_fcst=AB_prUnit_test_fcst.set_index(pd.to_datetime(AB_prUnit_test_fcst['ds']))

#Error metrics

#MSE
MSE_AB_prUnit=mean_squared_error(y_true=AB_prUnit_test['Pvv pr unit [W]'],
                y_pred=AB_prUnit_test_fcst['yhat'])

#RMSE
RMSE_AB_prUnit=math.sqrt(MSE_AB_prUnit)

#Normalized RMSE
NRMSE_AB_prUnit=RMSE_AB_prUnit/AB_prUnit_test['Pvv pr unit [W]'].mean()

#Mean absolute error
MAE_AB_prUnit=mean_absolute_error(y_true=AB_prUnit_test['Pvv pr unit [W]'],
                y_pred=AB_prUnit_test_fcst['yhat'])

#Mean absolute percentage error
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

MAPE_AB_prUnit=mean_absolute_percentage_error(y_true=AB_prUnit_test['Pvv pr unit
    [W]'],
                y_pred=AB_prUnit_test_fcst['yhat'])

#Tabulating error metrics
AB_prUnit_Prophet_error_metrics={'Mean Squared Error':MSE_AB_prUnit,\
                        'Mean Absolute Error':MAE_AB_prUnit,\
                            'Mean Absolute Percentage Error':MAPE_AB_prUnit,\
                                'Root Mean Squared Error':RMSE_AB_prUnit,\
                                    'Normalized Root Mean Squared Error':
                                        NRMSE_AB_prUnit}

#Making a dataframe of the error metric table
AB_prUnit_Prophet_error_metrics=pd.DataFrame(data=AB_prUnit_Prophet_error_metrics,
    index=[0])
AB_prUnit_Prophet_error_metrics=(AB_prUnit_Prophet_error_metrics.T)
AB_prUnit_Prophet_error_metrics.columns=['Value']
#Storing the error metrics in an Excel-file
AB_prUnit_Prophet_error_metrics.to_excel('AB_prUnit_Prophet_error_metrics.xls')
```

Figure 39: Python code for calculating and storing the error metrics in Excel-file, this particular code is for the AB Prophet Model error metrics, but the procedure is the same for the other models.