Degree Programme in Computer Engineering
First Cycle 15 credits

Evaluating machine learning models for time series forecasting in smart buildings

DIEGO PEREZ LEGRAND
SARUGAN BALACHANDRAN

STOCKHOLM, SVERIGE 2023

Evaluating machine learning models for time series forecasting in smart buildings

Utvärdera maskininlärningsmodeller för tidsserieprognos inom smarta byggnader

Diego Perez Legrand
Sarugan Balachandran

Examensarbete inom datateknik
Grundnivå, 15 hp
Handledare på KTH: Jayanth Raghothama
Examinator: Ibrahim Orhan
TRITA-CBH-GRU-2023:097

KTH
Skolan för kemi, bioteknologi och hälsa
141 52 Huddinge, Sverige

Abstract

Temperature regulation in buildings can be tricky and expensive. A common problem when heating buildings is that an unnecessary amount of energy is supplied. This waste of energy is often caused by a faulty regulation system. This thesis presents a machine learning approach, using time series data, to predict the energy supply needed to keep the inside temperature at around 21 degrees Celsius. The machine learning models LSTM, Ensemble LSTM, AT-LSTM, ARIMA, and XGBoost were used for this project. The validation showed that the ensemble LSTM model gave the most accurate predictions with the Mean Absolute Error of 22486.79 (Wh) and Symmetric Mean Absolute Percentage Error of 5.41 % and was the model used for comparison with the current system. From the performance of the different models, the conclusion is that machine learning can be a useful tool to predict the energy supply. But on the other hand, there exist other complex factors that need to be given more attention to, to evaluate the model in a better way.

Keywords

Machine learning, supervised learning, time series forecasting, neural networks, LSTM, Ensemble LSTM, AT-LSTM, ARIMA, XGBoost

Sammanfattning

Temperaturreglering i byggnader kan vara knepigt och dyrt. Ett vanligt problem vid uppvärmning av byggnader är att det tillförs onödigt mycket energi. Detta energispill orsakas oftast av ett felaktigt regleringssystem. Denna rapport studerar möjligheten att, med hjälp av tidsseriedata, kunna träna olika maskininlärningmodeller för att förutsäga den energitillförsel som behövs för att hålla inomhustemperaturen runt 21 grader Celsius. Maskininlärningsmodellerna LSTM, Ensemble LSTM, AT-LSTM, ARIMA och XGBoost användes för detta projekt. Valideringen visade att ensemble LSTM-modellen gav den mest exakta förutsägelserna med Mean Absolute Error på 22486.79 (Wh) och Symmetric Mean Absolute Percentage Error på 5.41% och var modellen som användes för att jämföra med det befintliga systemet. Från modellernas prestation är slutsatsen att maskininlärning kan vara ett användbart verktyg för att förutsäga energitillförseln. Men å andra sidan finns det andra komplexa faktorer som bör tas hänsyn till så att modellen kan evalueras på ett bättre sätt.

Nyckelord

Maskininlärning, övervakad inlärning, tidsserie-prognostisering, neurala nätverk, LSTM, Ensemble LSTM, AT-LSTM, ARIMA, XGBoost

Acknowledgments

This paper is a result of a bachelor's thesis in the field of computer engineering, at the Royal Institute of Technology, for Iquest.

We would like to thank Farhad Basiri and the entire Iquest team for their support and guidance. We want to thank Ivo Martinac for his advice regarding building services and energy systems. We also want to thank our mentor Jayanth Raghothama for his guidance and support throughout this project.

Table of contents

1 Introduction

Heating and cooling systems in buildings have long been regulated manually and are now up for a change where an automated regulation is the desired form. Inside temperature is an important topic all over the world, some countries may need better cooling systems and others may need better heating systems to have a good inside environment where people can feel comfortable.

Most buildings have some sort of temperature regulation that a supervisor can digitally regulate. This temperature regulation is majorly based on the outdoor temperature and indoor temperature. For the indoor temperature, the influencing factors could be, for instance, the number of people inside the building, because the temperature inside will automatically be a little higher due to the heat emitted from human body. To heat or cool down a building, energy is needed, especially in warmer and colder environment. In Sweden a lot of time is spent inside buildings, we live and work in buildings and 40% of Sweden's energy usage is needed to make this possible [1]. Even though the transitions of energy to the indoor climate and power in the outlets are digitally managed in most of the buildings, the operations are often faulty and could be made more efficient.

IQuest is a company that has designed a sensor that collects data regarding the indoor temperature, outdoor temperature, and the energy supplied to regulate the temperature, from companies that they collaborate with. The data they have collected which is in a time series manner shows that the indoor temperature in most cases exceeds 21 degrees Celsius and sometimes is below 21 degrees Celsius. In the cases where the temperature exceeds the 21-degree boundary, there is both an energy wastage and a loss in capital for the energy supplier.

1.1 Problem

In the beginning, IQuest collected this data so the collaborating companies, by analyzing the data, could manually regulate the energy fed into the buildings or in other words decrease/increase the inside temperature based on what the data tells them. Since they have valuable data for analyzing and correcting faulty operations, IQuest intends to develop a machine learning model that can predict the energy needed for the heating system to keep the inside temperature around 21 degrees, based on the future temperature outside, the temperature inside, and other influencing factors. With a model like this, the regulation would be efficiently automated, and the energy wastage would be minimized.

1.2 Goal of the project

The goal of this thesis project is to evaluate different machine learning models that can mimic the regulation system in use and then use the most generalized model to predict the energy supply needed to acquire an inside temperature of 21 degrees Celsius.

This goal can be broken down into several steps:

- Analyze the time series data.
- Find a suitable machine learning model for time series forecasting.
- Compare and evaluate the models.
- Compare the best model with the existing system.

1.3 Limitations

The architecture of the building and the average number of people that are commonly inside was information not available, and due to that reason, it wasn't possible to analyze the building as profoundly as desired. When predicting the energy supply, for optimal indoor temperature, it is important to consider the size and the average amount of people spending time in those rooms. Since that data was not available for this project they won't be taken into account when predicting, but the influence of some indirect factors is embedded in the total energy feature. Furthermore, if the rooms vary in size, the rooms should be divided into clusters to get a more precise prediction. Due to the time frame of the project, availability and scope the building will be clustered as one big zone.

2 Theory and Background

This chapter will introduce time series analysis, time series forecasting, and the five machine learning models candidates for this thesis.

2.1 Time Series

The dataset that is observed and stored in chronological order is called a time series. The key to this kind of dataset is that the time is the index, which could be observations every year, month, day, hours, seconds, etc. The time interval between the observations is often equal [2].

2.1.1 Time series data - Characteristics

The primary question when having time series data is how the future values can be predicted based on the past. The time series data is called univariate when the value that is being observed is a single variable whereas it is called multivariate when there are multiple variables that are being observed [2].

Time series data can be analyzed and characterized in multiple ways. Some important characteristics of time series are trend, seasonality, and cyclical [2]. A trend exists when a long-term increase or decrease is seen in the dataset, it is called an increasing trend or decreasing trend. When the time series data is influenced by seasonal factors such as summer, it is said that there is seasonal patterns. Seasonality is of fixed and known frequency. In cases where the data display rises and falls that do not have a fixed frequency, a cycle has occurred [3].

The time series data is called stationary when its distribution does not change over time, the data has no trend and deterministic seasonal changes [2].

2.1.2 Time series analysis - Approaches

Analyzing the dataset from a statistical approach where characteristics such as trend and seasonality are analyzed is called time series analysis [2]. Time series analysis (TSA) consists of both descriptive analysis, summarizing characteristics of a dataset, and exploratory analysis, which analyzes patterns, trends, and relationships between variables [2].

Data cleaning, feature engineering, and training a machine learning model are part of TSA but are not considered strict TSA steps because these steps are common for all types of analysis. The steps that are crucial for TSA and considered as central, are understanding the variables, uncovering relationships between variables, and identifying trend and seasonality [2].

Understanding a univariate variable includes the following, inspecting the mean, variance, and plotting the distribution. Standard deviation is a measurement of how much each value deviates from the mean, giving a good understanding of how spread the data is. To get the standard deviation of sample data the standard error (SE) can be used. The confidence interval, whose value indicates how good an estimate is, can be calculated using the help of SE [2].

With multivariate variables, the correlation between the variables should be investigated. Importantly the variables should not have any collinearity and feature leakage [2]. The trend, seasonality, cyclic variations, and stationarity are characteristics that can be analyzed using both a graphical approach and a mathematical approach. Seasonality and trend can be estimated using the formula presented in formula 1, where *k* is the degree of the polynomial and *b* is the coefficients being sought [2].

$$f(x) = \sum_{i=0}^{k} b_i x^{i+1} \tag{1}$$

To test for stationarity, in time series, both plotting and statistical approaches can be used. A common statistical method to test for stationarity is the augmented Dickey–Fuller test, which will return a p-value. A p-value below 5% (0,05) indicates that the time series is stationary [2].

2.1.3 Time series forecasting

Forecasting is the process when a machine learning model is trained with the time series data and used to predict future values. Based on collected data and analysis, a suitable model/models are used [4].

2.2 Recurrent Neural Network

Recurrent neural networks (RNN) were introduced around the 1980s when the Hopfield network was invented. The significant difference between RNN and the normal feed-forward network is in the hidden states [5]. RNN can make different predictions by taking into account previously seen data, this is done by temporarily storing information about previous outcomes [6]. Figure 2.1 gives a simple representation of the RNN structure.

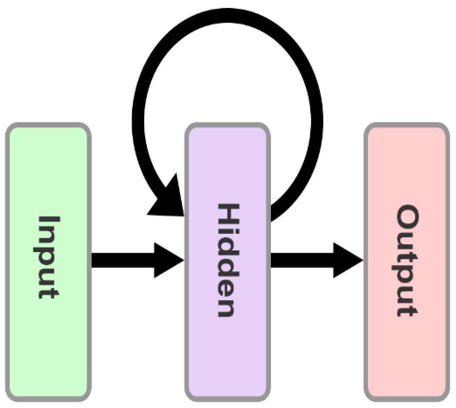


Figure 2.1: Simple example of RNN structure that uses its output from the hidden layer again as input [5].

RNN faces some difficulties regarding forgetting previous important information due to vanishing gradient descent and exploding gradient descent. Vanishing- and exploding gradient descent occurs in the neural network training process and commonly occurs when the length of the processing time series increases [7]. The weights and biases are updated during the training process using a backpropagation algorithm. When this operation is done for multiple time steps with the same set of parameters, the value of the derivative can become too large (exploding gradient descent) or too small (vanishing gradient descent). Since that value is used to update the parameters, a large value can result in undefined weights and biases and a small value can result in no significant update, and thus no learning [5]. The LSTM solves both the vanishing- and exploding gradient descent problem.

2.3 LSTM

In 1997 an advanced RNN model called long short-term memory (LSTM) was proposed by Hochreiter and Schmidhuber [5].

2.3.1 LSTM - Background

LSTM is a type of recurrent neural network that works well with multiple deep learning tasks such as handwriting recognition and generation, speech synthesis, language modeling and translation, and analysis of audio and video [8]. The reason why LSTM stands out in these areas is because it can choose how much of the previously seen data will be reflected in the output, this makes LSTM a good option for predicting time series problems. LSTM neural network was developed to solve the vanishing- and exploding gradient problem that RNN faces. To make this happen the LSTM neural network uses three gates to select what information should be stored, updated, or deleted [6]. These gates will be properly introduced below.

2.3.2 LSTM architecture

The LSTM neural network architecture consists of memory units. The memory units are the middle layer of the neural network, Figure 2.2 shows a detailed image of the memory unit in the hidden layers.

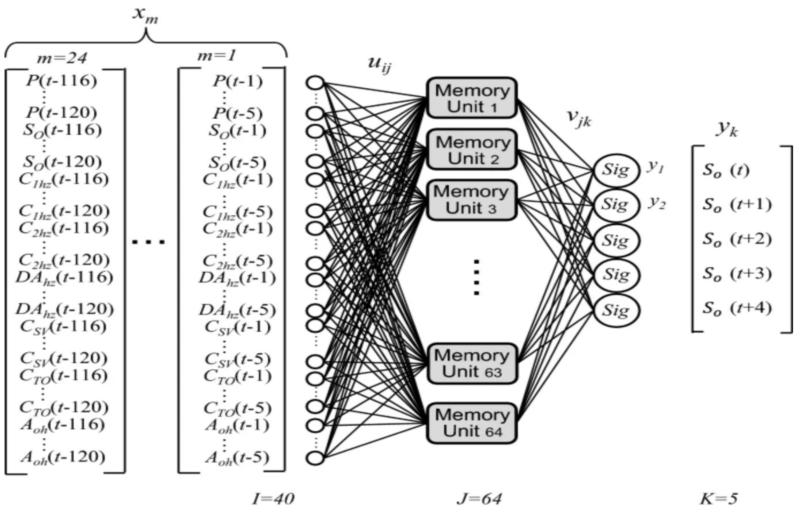


Figure 2.2: An overview of ANN architecture with LSTM where the memory units in the middle layer are presented [6].

A memory unit contains three gates (an input gate, a forget gate, and an output gate), a memory cell, and an input. The structure of one single memory unit is presented in Figure 2.3

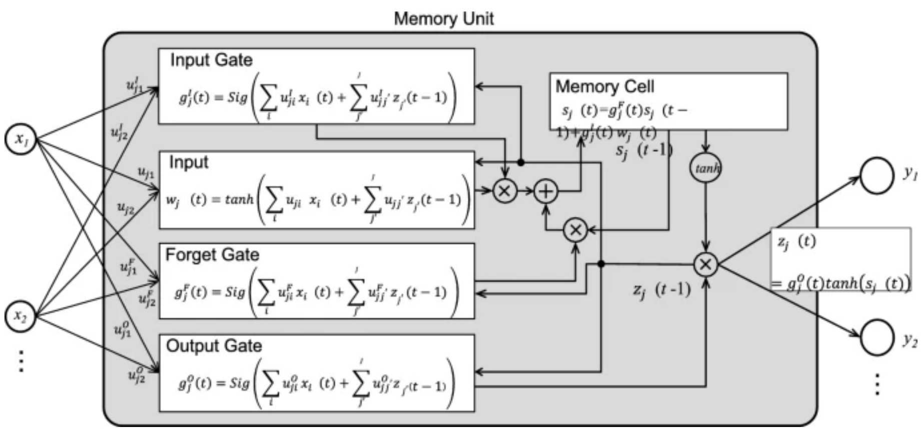

Memory Unit
Input Gate
$g_j^I(t) = Sig\left(\sum_i u_{ji}^I x_i(t) + \sum_{j'}^J u_{jj'}^I z_{j'}(t-1)\right)$
Input
$w_j(t) = tanh\left(\sum_i u_{ji} x_i(t) + \sum_{j'}^J u_{jj'} z_{j'}(t-1)\right)$
Forget Gate
$g_j^F(t) = Sig\left(\sum_i u_{ji}^F x_i(t) + \sum_{j'}^J u_{jj'}^F z_{j'}(t-1)\right)$
Output Gate
$g_j^O(t) = Sig\left(\sum_i u_{ji}^O x_i(t) + \sum_{j'}^J u_{jj'}^O z_{j'}(t-1)\right)$
Memory Cell
$s_j(t) = g_j^F(t) s_j(t-1) + g_j^I(t) w_j(t)$
$s_j(t-1)$
tanh
$z_j(t) = g_j^O(t) tanh(s_j(t))$
$z_j(t-1)$
x_1
x_2
y_1
y_2

Figure 2.3: One LSTM memory unit with three gates and the memory cell, the figure also shows the activation functions as symbols and the path (calculations) they take [6].

The memory cell works like a long-term memory and holds all important previous information. The forget gate specifies what percentage of the long-term memory should remain in the memory cell, this can be calculated by using a sigmoid activation function that will give a result between 0 and 1. The input and the recurrent input will each be multiplied by some weight, get summed up, and then added with a bias. This will work as a parameter for the *sigmoid* activation function. The output of the *sigmoid* activation function will then be multiplied by the long-term memory [6].

The input gate will calculate what percentage of the input will be added to the memory cell with the help of the *sigmoid* activation function and the *tanh* activation function. As in the previous process in the forget gate the input and the recurrent input will each be multiplied by a weight, get summed up, and then add a bias. This number will serve as a parameter for the *tanh* activation function. The *tanh* activation function gives a result between -1 and 1, this result represents how much of the input has the potential to be summed to the long-term memory and is called potential long-term memory. How much of the potential long-term memory will be added to the long-term memory is decided with the help of the *sigmoid* function as in the forget gate. The output of the *sigmoid* activation function will then be summed to the long-term memory [6].

The output gate is similar to the input gate, both processes use both the *tanh* activation function and the *sigmoid* activation function. The long-term memory will be used as a parameter for the *tanh* activation function, and the output will be the potential output. The *sigmoid* activation function, with the help of the input and the recurrent input, will give what percentage of the potential output will be the output. The output will then be recurred to the memory unit and the whole process will be done again with a new input and new calculated recurrent input [6].

2.4 Attention Mechanism

The attention mechanism in deep learning helps a model to focus on the relevant input features when making predictions. By focusing on the relevant features and ignoring others the models become more efficient and accurate. The basic procedure of the attention mechanism is that, at each time step, it attends every hidden state of all encoder nodes and determines the most relevant information [9].

The attention mechanism consists of query, keys, values, and output which are all vectors. The mechanism works such that it maps a query and a set of key-value pairs to an output. With each value, a corresponding weight is calculated using a function compatible with the query and corresponding key. The weighted sum of values becomes the output [10].

There are several types of attention functions such as additive attention, dot-product attention, and scaled dot-product attention [10]. All three functions are similar with minute changes in calculations. For simplicity and due to high similarities only the scaled dot-product attention will be explained deeply.

The computation works as follows: dot product is executed between all queries with the corresponding keys, and the result is then divided with the square root of the dimension of the queries (or key, both have the same dimension). Following that the *softmax* activation function is applied to obtain the weights for the values and lastly the value vector is multiplied by the weights. The formula for the scaled dot product is presented in formula 2. The dot product is almost identical to the scaled dot - product except for the scaling whereas, for the additive attention, the difference lies in the compatibility function [10].

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2}$$

2.5 AT - LSTM

An attention-based LSTM is as the name conveys a model that uses both the attention model and the LSTM deep learning model. This model was proposed for time series in a research paper by Xun Liang, Aakas Zhiyuli, Shusen Zhang, Rui Xu, and Bo Wu [11] and a simplified representation of the full process is presented in Figure 2.4.

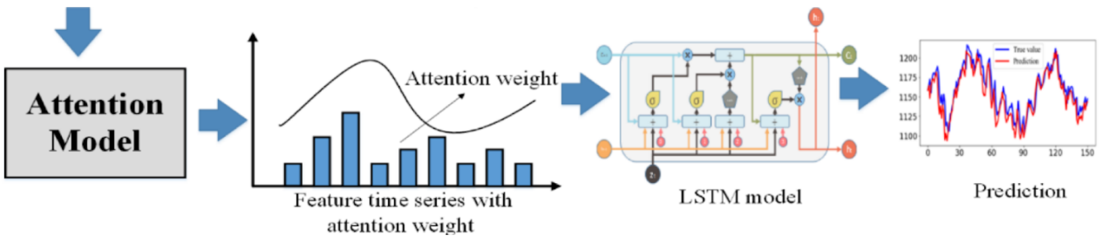


Figure 2.4: An overview of the AT - LSTM process [11]. What the figure conveys is that before the LSTM model gets the input the attention assigns different weights to each input variable so the LSTM model can focus more on them.

The attention model takes the time series data as its input, and using the mechanism of attention the independent values with higher influence on the target value can get higher weights associated with them. The output is then fed into the LSTM model to make a prediction. AT - LSTMs core concept is to adaptively select relevant input features and, in that way, improve the accuracy [11].

2.6 Ensemble Learning

Ensemble learning is when you use multiple learning algorithms and take each into account when making a prediction. Some examples of ensemble learning are Bagging, Boosting, and Stacking.

Stacking consists of multiple learning algorithms and a meta-learner. This meta-learner will take as input the output of the learning algorithms and learn from them all. Bagging will predict with the help of voting; all the learning algorithms will then go through a vote where those with the same prediction will accumulate votes. The prediction with the most votes will be used. Boosting will put more attention on weak learners and try to optimize them, by doing this the models will perform better overall [12].

2.6.1 Gradient Boosting

Gradient boosting is a technique that is used for minimizing the prediction error. The whole idea with gradient boosting is to provide greater emphasis to learners that have worse performances. When using gradient boosting the next model will be trained with the error of the previous model, this will lead to an ensemble of models. To predict on new instances, it is needed to sum all the predictions made from each of the models [13].

2.6.2 XGBoost

XGBoost stands for Extreme Gradient Boosting and is an open-source machine learning library. XGBoost builds upon supervised machine learning, decision trees, ensemble learning, and gradient boosting. This machine learning framework is scalable, can build trees in parallel, use memory efficiently and process big amounts of data at high speed. The reason why XGBoost is scalable is because it uses out-of-core computation. XGBoost can be used for both regression and classification problems due to it being an ensemble learner. To regulate overfitting it uses Regularized objective, shrinkage, and column sub-sampling. The regularized objective also helps the model to be as simple as possible, this is done by penalizing the model if it gets too complex. Column sub-sampling makes the computation of the parallel algorithm faster, this is done by making nodes communicate less data between each other and in that way reduce the communication overhead [14].

2.7 ARIMA

ARIMA (Autoregressive Integrated Moving Average) is a very popular time series forecasting model that consists of three main components, AR (Autoregressive), I (Integrated), and MA (Moving Average) [15][16].

2.7.1 ACF and PACF

Correlation is an indicator that describes the relationship between two different variables whereas autocorrelation describes the relationship of a variable with its previous values called lags [15].

ACF which stands for auto-correlation function is a method that can be used to find both the direct and indirect effect of values in previous time lags. PACF (Partial Auto-Correlation Function) on the other hand is used to find only the direct effect of values in previous time lags [16].

2.7.2 Autoregressive

The autoregressive model predicts the future value of the target variable (y) using a linear combination of previous values of y as shown in formula 3. The amount of previous y values, lags, that are going to be used is denoted with the letter p, AR(p). AR (2), for instance, indicates that the lag is two, so y(t-1) and y(t-2) will be used to predict y(t) [3] [15].

The value of p for the AR model can be chosen using the ACF and PACF methods. Generally, the PACF method is preferred over ACF for AR since it measures the direct correlation between the current and past value, which is exactly what AR takes into account [16].

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t, \tag{3}$$

The last term εt is called white noise [3], also called the error term which plays a key part when it comes to the moving average model.

2.7.3 Moving Average

The Moving Average model also uses past values but instead of using past values of y, it uses the past errors as a linear combination as presented in formula 4. Lags of MA are denoted with a q, MA(q) [15].

For the lags same as for AR the method ACF and PACF can be used but the ACF is preferred here since the MA model uses the error term of past y values so analyzing the direct and indirect effect can be efficient [14].

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}, \tag{4}$$

2.7.4 Integrated

Integrating is the process where a nonstationary series is converted to stationary. There are several methods for making a series stationary, a very popular method is called differencing. Most of the time the differencing will happen multiple times to make the series stationary. The number of times a series is differenced to make it stationary is denoted with a d, I(d) [3].

2.8 Related Work

In a study by Jui-Sheng Chou [17] single models, ensemble models and hybrid models were used to forecast the energy consumption using time series data collected from a residential building. For single models, SARIMA, ANN, SVM, LR, Classiffication & Regression tree (C&R tree) were used. Thereafter for the ensemble models the following was used, SVR + LR and bagging ANN. Lastly there was two hybrid models, SARIMA-MetaFa-LSSVR and SARIMA-PSO-LSSVR. All these models were evaluated using five different metrics, namely, R, RMSE (kWh), MAE (kWh), MAPE (%), MaxAE (kWh). It was shown that the hybrid model SARIMA-MetaFa-LSSVR was the most accurate model for the forecasting using the collected data. The RMSE, was 0.164 (scaled result) for the SARIMA-MetaFa-LSSVR model.

In another study by S. Nazir [18] forecasting energy consumption with time series data of home electricity was performed using different multistep LSTM models. In this study the vanilla LSTM, Bidirectional LSTM, Stacked LSTM and Convolutional LSTM were used, and the evaluation metric was RMSE. The models were evaluated using data from four different households. The result showed that the Convolutional LSTM model achieved overall high predictive accuracy and was also less computationally heavy. The RMSE achieved for the Convolutional LSTM was 23202.

2.8.1 Relevance of related work

The related work focused mainly on studies related to forecasting energy supply and the different types of models utilized. It is important to note that there is not a lot of difference in time series forecasting for energy supply and other types of time series data. The focus has been to study and use models that has been proven to be effective in time series forecasting in general.

3 Methodology

This chapter will cover the chosen methods, procedures, frameworks, and architectures used to analyze the time series data and to construct different machine learning models.

The methods used in this thesis were:

1. Preprocess the data.
2. Build ML models.
3. Evaluate models.

Feature engineering is essential for the performance of machine learning models. Without a proper preprocess, the models will not be able to find patterns and make accurate predictions. The models were built with proper parameters and then trained. The evaluation method was to make tests and observations. Due to that our thesis goal is to evaluate ML models, the method used was the implementation method and for evaluation, tests and observations were done.

3.1 Tools and Frameworks

The programming language used for this project was Python. The chosen coding environment is Jupyter due to its favorable visualization strengths. PyCharm is also used for storing reusable methods like merge dataset, and clean_df, which are methods used by several Jupyter files. Frameworks used are pandas, NumPy, seaborn, sklearn, Keras, xgboost, and normaltest.

3.2 Preprocessing

Most of the work went into preprocessing, both general preprocessing and TSA-based preprocessing [2]. This coming section will cover the whole preprocessing process divided into three main areas.

3.2.1 Data

The data in hand was in four separate CSV files, where each file contained information on one measurement feature. The measurements were of:

- Indoor temperature
- Outdoor temperature
- District heating (Wh)
- Heat pump (Wh)

The indoor temperature consisted of measurement for each room in the building while the outdoor temperature was the measured temperature near the area where the building is situated. District heating and heat pump consisted of the measured amount of energy supplied to heat the building but from different sources as the name indicates. There was a preprocessing done for each separate feature and then another preprocessing was performed after merging all the features to one dataframe.

The idea behind preprocessing them separately was mainly because it would have been extremely complicated to merge them due to their heterogeneity. At the same time the data was in a format that wasn't easily comprehensible, so merging all together would have complicated the analysis and preprocess. A simple representation of the dataset is presented in Figure 3.1.

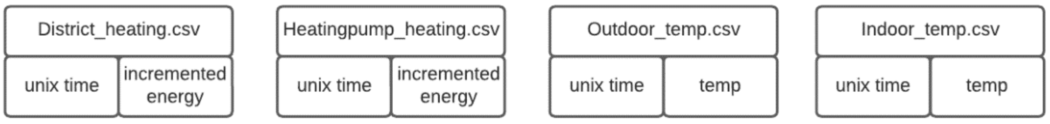


Figure 3.1: Simple representation of the four CSV files and their structure. The indoor temp CSV is simplified a lot, it had multiple columns each representing the indoor temperature of one room.

3.2.2 Cleaning and Feature Engineering - Separate features

Data cleaning, feature engineering, and a limited amount of TSA analysis were done for each CSV file consisting of four different measurements. All four CSV files were read as a dataframe using pandas.

To be able to comprehend the data, the time column for all four features which was in Unix timestamp was converted to datetime, which made it both interpretable and easier to plot. After that, a common cleaning step for all four features was to convert the time series data to some predetermined standard. The initial problem faced with the data was that the measurements were of different time intervals. To make a clear example, the heat pump data was initially measured every hour but in the later days that changed to every minute. Whereas the outdoor temperature was measured every five minutes. There was heterogeneity both within the same feature and between features. Making a good decision regarding what time interval to use was a crucial step when merging, it is important that each row has values that represent that exact time and nothing else.

For the indoor feature, as mentioned, the structure was a bit different, so the process of shifting them to the right rows was executed. Since no clustering had to be done an average of the temperature from all rooms was taken as the indoor temperature. The heating pump and the district heating meter on the other hand were a bit different. The sum of these two at a particular time is the total energy supplied to heat the building. However, the sensor collects the data in a slightly different manner than explained right now, every time the sensor collects data it increments with the amount of energy supplied. That means the difference (delta time) between the energy supplied in the previous hour with the amount it is in now (present) is the total amount of energy supplied for that hour.

The step of differencing each row from the other to get the energy supplied had to be done after a time interval for the features had been chosen. The complication of choosing an interval was that, if the minute interval is chosen all four dataframes must go through some kind of imputation since a lot of the measurements aren't measured in that interval. On the other hand, if hourly intervals are chosen, data would be lost since only the data point for each hour is needed and the rest will be thrown away. For the ten-minute interval, both situations will occur, the data in an hourly manner will be imputed whereas the ones in every minute and every fifth minute will be thrown away. All three scenarios were analyzed and tested using simple models and the chosen time interval will be presented in the result section. Figure 3.2 gives a simple overview of how the dataset looked after cleaning.

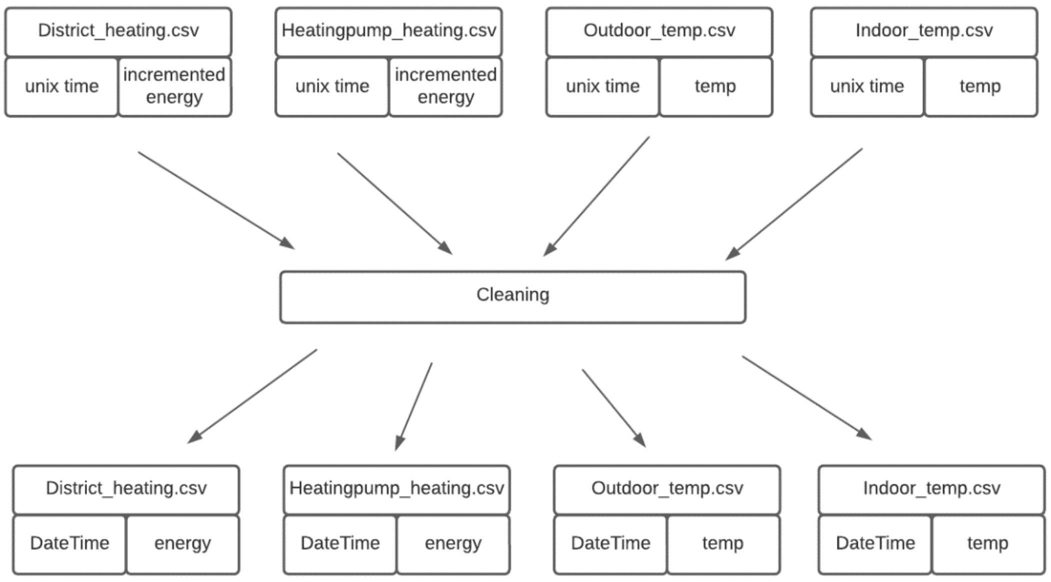


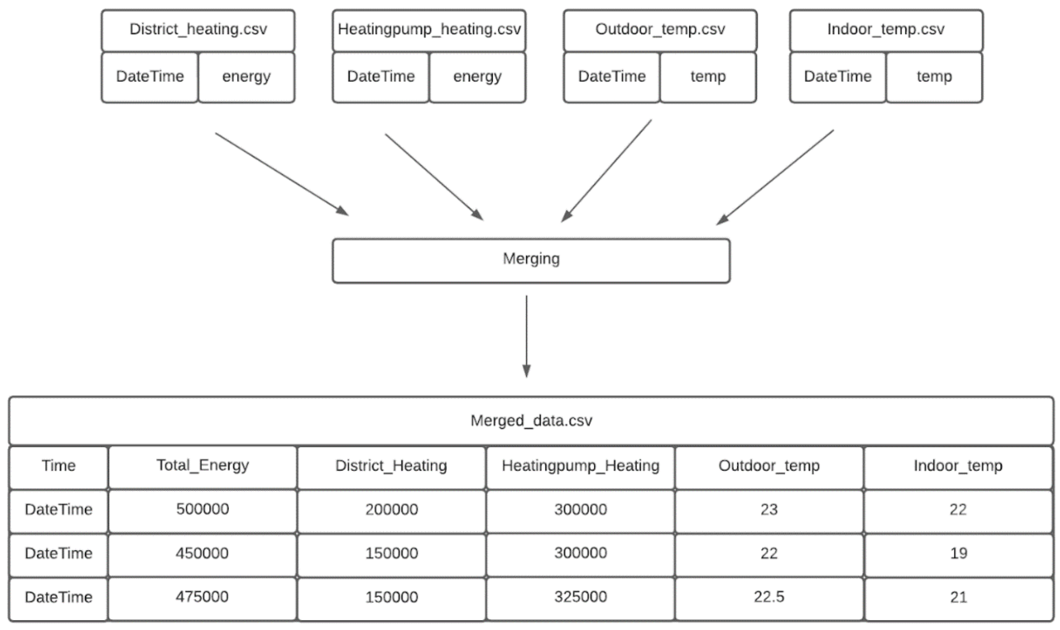
Figure 3.2: A simple representation of the four CSV files after all common cleaning step was performed.

Before merging, some univariate feature engineering procedures were done. The basic procedure of checking the mean, variance, and standard deviation, and following that, scatter plots and histograms were plotted. With the separate features, finding outliers was a main goal so that less cleaning and more analyzing could be performed after merging. Apart from outliers, missing dates could be found using plotting methods. There were up to two months of missing data, specifically for the indoor temp. Having the energy supplied for those months is useless if the indoor temperature does not exist, the same implies for all other features too. The method used for solving will be discussed in the merging section. To explore the behaviors of the features, trend, and seasonality were checked. Since the data is based on temperature and energy, there should be a clear sign of trend and seasonality.

3.2.3 Merging

After thorough cleaning and analysis for the separate features, the next process was to merge the data to get one dataframe with independent variables (X) and the target variable (y) as presented in Figure 3.3.

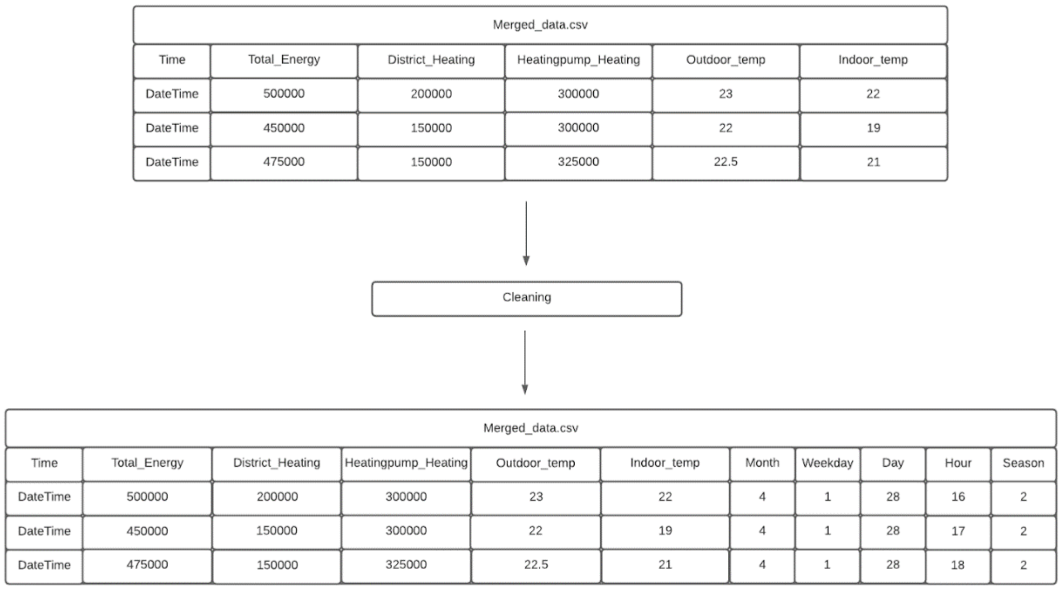
The merging has to be done with respect to the index, which is the datetime. Since there was some datetime missing, a merge based on the index will mean a throwaway of data in places where some features are missing but other features are available. Throwing away data was not sustainable because the time series data received originally was already very small, approximately one and a half years' worth of data.



Time	Total_Energy	District_Heating	Heatingpump_Heating	Outdoor_temp	Indoor_temp
DateTime	500000	200000	300000	23	22
DateTime	450000	150000	300000	22	19
DateTime	475000	150000	325000	22.5	21

Figure 3.3: The simple structure of the merged dataset.

The method chosen was to make an imputation and since the dataset was very close to two years the decision was to expand the data to two years. Before making the expansion and imputation some extra independent features were added. The only independent features originally at hand were the indoor and outdoor temperatures. The reason to add extra features was that using two features was not enough, even though they play a major role in deciding the amount of energy supplied. The other independent features that were added were time periods. The features that were chosen were the hour, day, month, weekday, and season. Since machine learning models cannot handle any text-based inputs, all these features were so-called one-hot encoded and received their numerical representation. The target column called total energy was also added which was the sum of district heating and heat pump. Figure 3.4 shows a simple representation of the merged dataset.



Merged_data.csv

Time	Total_Energy	District_Heating	Heatingpump_Heating	Outdoor_temp	Indoor_temp
DateTime	500000	200000	300000	23	22
DateTime	450000	150000	300000	22	19
DateTime	475000	150000	325000	22.5	21

Merged_data.csv

Time	Total_Energy	District_Heating	Heatingpump_Heating	Outdoor_temp	Indoor_temp	Month	Weekday	Day	Hour	Season
DateTime	500000	200000	300000	23	22	4	1	28	16	2
DateTime	450000	150000	300000	22	19	4	1	28	17	2
DateTime	475000	150000	325000	22.5	21	4	1	28	18	2

Figure 3.4: The final merged dataset with additional features.

With the dataframe ready with all the needed features, it was expanded using pandas with a specified interval of date and with the frequency of an hour. The approach that was taken was to fill in the missing months with the previous/following years data. The decision of using the previous or following came down to availability. However, the reason that the data was not of full two years, some datetime couldn't be filled using neither the previous nor the following, for that the method from pandas called *forwardfill* (ffill) was used. The imputation could confidently be performed using the techniques above since the values aren't random, they have close relationships with each other on both hourly and daily basis but also depending on the season.

3.2.4 Feature Engineering and TSA - Merged data

The merged data has the actual valuable information, and it was possible to analyze the relationship between variables. Since the univariate analysis is performed, uncovering the relationship was the first objective. To get a good insight on the correlation between variables and specifically between the independent variables and the target variable a correlation matrix was used. With the dataframe a simple correlation table can be created using pandas built-in function, but to get a clearer view of how correlated each feature is the seaborn library was used to make a heatmap. The seaborn heatmap was then plotted using matplotlib, specifically using the pyplot package. Discovering the relationship was useful to check the correlation between the target variable and the independent variables and to check for any collinearity between the independent variables.

Because the total energy variable was derived using the sum of two existing features, which already were cleaned, no further cleaning was necessary. Using pyplot and pandas *describe* method the mean, and standard deviation were rechecked [2]. A histogram plot was made to give a good view of the distribution of the energy supply and the frequency.

The three important steps when it comes to TSA are to identify the trend, seasonality, and stationarity [1]. Checking for stationarity was specifically important since one of the models planned to use was ARIMA, which performs optimally when data that is stationary [3]. To identify the trend and seasonality the *polyfit* method from numpy was used. Using a fit function with the X and y feature and a specified degree makes it possible to generate the trend and seasonality as showed in Figure 3.5 [2].

```
def fit(X, y, degree=3):
   coef = polyfit(X, y, degree)
   trendpoly = np.poly1d(coef)
   return trendpoly(X)


def get_season(s, yearly_periods=12, degree=3):
   X = [i % (365 / 4) for i in range(0, len(s))]
   seasonal = fit(X, s.values, degree)
   return pd.Series(data=seasonal, index=s.index)


def get_trend(s, degree=3):
   X = list(range(len(s)))
   trend = fit(X, s.values, degree)
   return pd.Series(data=trend, index=s.index)
```

Figure 3.5: Methods for creating trend and seasonality [2]. Both methods use the function fit for finding season and trend.

A test for stationarity was done using the augmented Dickey-Fuller [2] and for this test the *stattools* from the package statsmodels.tsa were used. The return of the p-value decides if the time series data is stationary.

Most of the analyzing methods were performed by plotting different types of figures, with different time intervals. Simple histograms and scatter plots present a simple overview of the time series data and help detect evident outliers, an example can be seen in Figure 3.6.

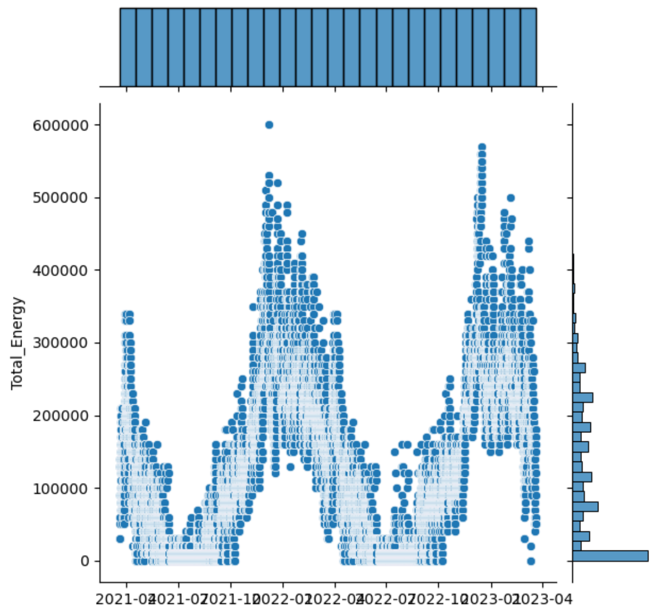


Figure 3.6: A scatter plot combined with a histogram.

To get to learn more about the behavior of the data under different time intervals the seaborn boxplot method was used. Utilizing the boxplot method and with all the period features that were added, five boxplots were plotted. The y-axis of the boxplot is the total energy, and the x-axis is the different period features. To know how the period features vary and how correlated they are to the total energy this boxplot method was used. Although the correlation matrix gives a good interpretation with values and color gradient, using the boxplot, as seen in Figure 3.7, had the advantage of showing patterns that were even more comprehensible.

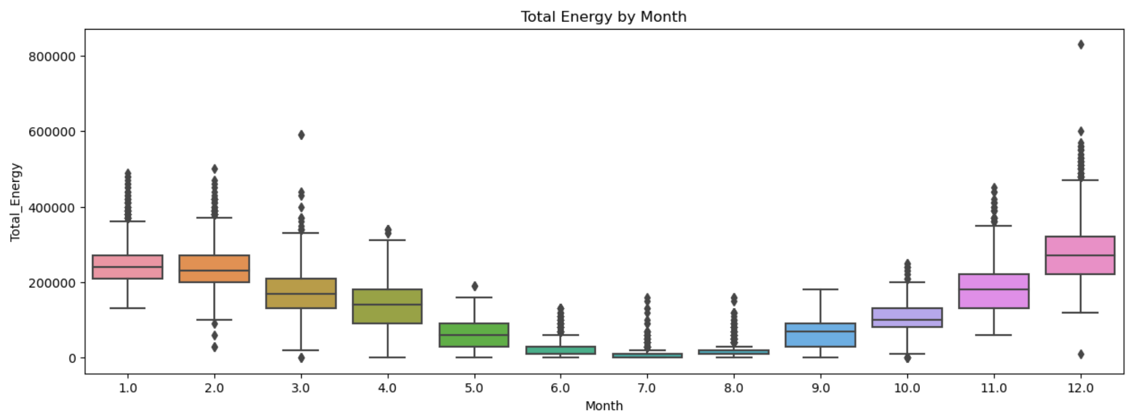


Figure 3.7: An example of a boxplot for the total energy supplies each month.

3.2.5 Preparing the time series data for training

The time series data has to be handled a bit differently compared to a normal dataset. That applies specifically when splitting the data, the method of splitting was different for different areas of use.

X_train, X_test, y_train, y_test, split with different ratios were performed using the *train_test_split* method from sklearn. Train test split shuffles the data by default and this is not optimal for time series because the data needs to be sequential to be able to predict the future, therefore the shuffle was deactivated.

This simple split approach was performed to quickly get an overview of how the models perform so that an unnecessary amount of time was not wasted on tuning a model that performs very weakly. Thereafter to build and train the model another splitting approach was taken, which was, the train test validate method, the same train_test_split function from sklearn was used but this time the split was performed twice. A third splitting approach was also used, which was for cross-validation and will be described in a later section.

The independent values are near the range of -15 to 40 C but the range for the target variable is between 10000 to 800000 so to overcome the problem the *minmaxscaler* from sklearn preprocessing was used. Testing showed that only scaling the target variable was effective, so only the total energy column was scaled. The *minmaxscaler* scales the value between zero and one.

3.3 ML - Models

In this section, the models used for forecasting and how they were constructed will be introduced. The final cleaned and merged data was divided into input data (X) and output data (y). The input data consists of indoor temperature, outdoor temperature, season, month, weekday, day, and hour. The output data is the total energy supplied. The models explained below will use all or some of the data to make predictions.

3.3.1 ARIMA - Building and Training

Only the target variable will be used to train the ARIMA model, and the lags will decide how many y values from the past to consider [3,15,16]. Verifying that the data is stationary was completed at preprocessing.

Before training the model, firstly the number of lags for the AR and the MA had to be decided, for that the *auto_arima* method from the pmdarima library was imported. The *auto_arima* method taking the target variable performs a stepwise search to minimize the AIC. Stepwise search to minimize AIC refers to the process of selecting optimal parameters for the ARIMA model based on the data in hand. Generating a low AIC value relies on finding the right value of lags for the AR, I, and MA. The generated *(p,d,q)* values were used to build the ARIMA model [3,15,16]. Before building the ARIMA model the target data was simply split using the *iloc* method from pandas. The splitting was performed with different ratios so that it was possible to evaluate the models abilities to perform on different sizes.

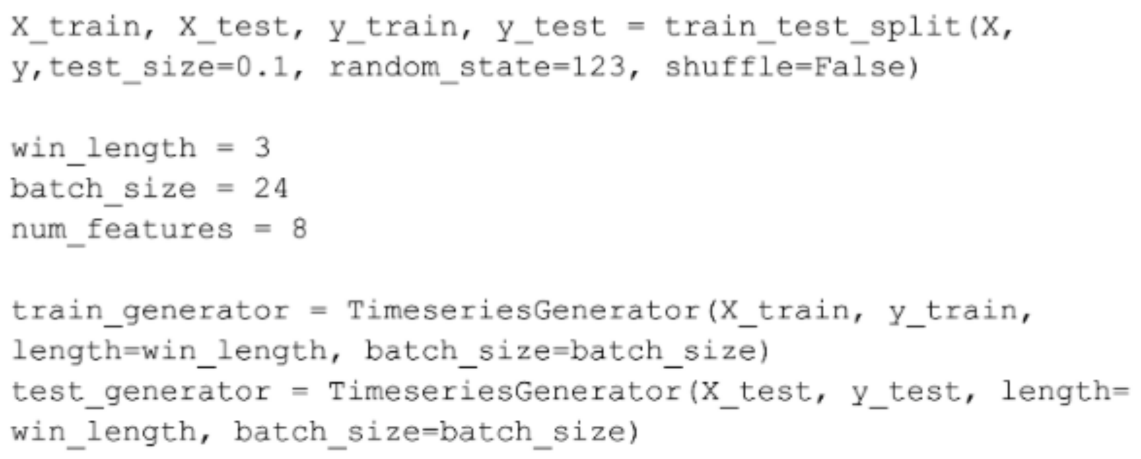
Building the ARIMA model was accomplished using the *statsmodels* library, and the *p,d,q* value received from the *auto_arima* method was fed in as the order parameter for the model. For assurance, a simple trial and error based method was also performed where some random *p,d,q* values, close to the values generated from the *auto_arima*, was chosen to test the models performance.

The model was fitted using default parameters and the metrics used for evaluating the model were Mean Absolute Error and Symmetric Mean Absolute Percentage Error. Although metrics values can be very descriptive, simple plotting with *matplotlib* was also performed to compare the result visually.

3.3.2 LSTM - Building and Training

As mentioned in section 3.2.5, the train test split method from *sklearn* was used to split the data into train and test. For all the LSTM models two types of approaches were used to create the train and test data. Both approaches utilized the train test split method but one of the approaches had one extra step.

The first approach had two steps, the first step was the simple train test split which generates the X_train, X_test, y_train, y_test, and the next step was to use the TimeSeriesGenerator from the Keras library were the splitted data was converted into a train generator and test generator. The procedure of creating train and test generators is presented in Figure 3.8.



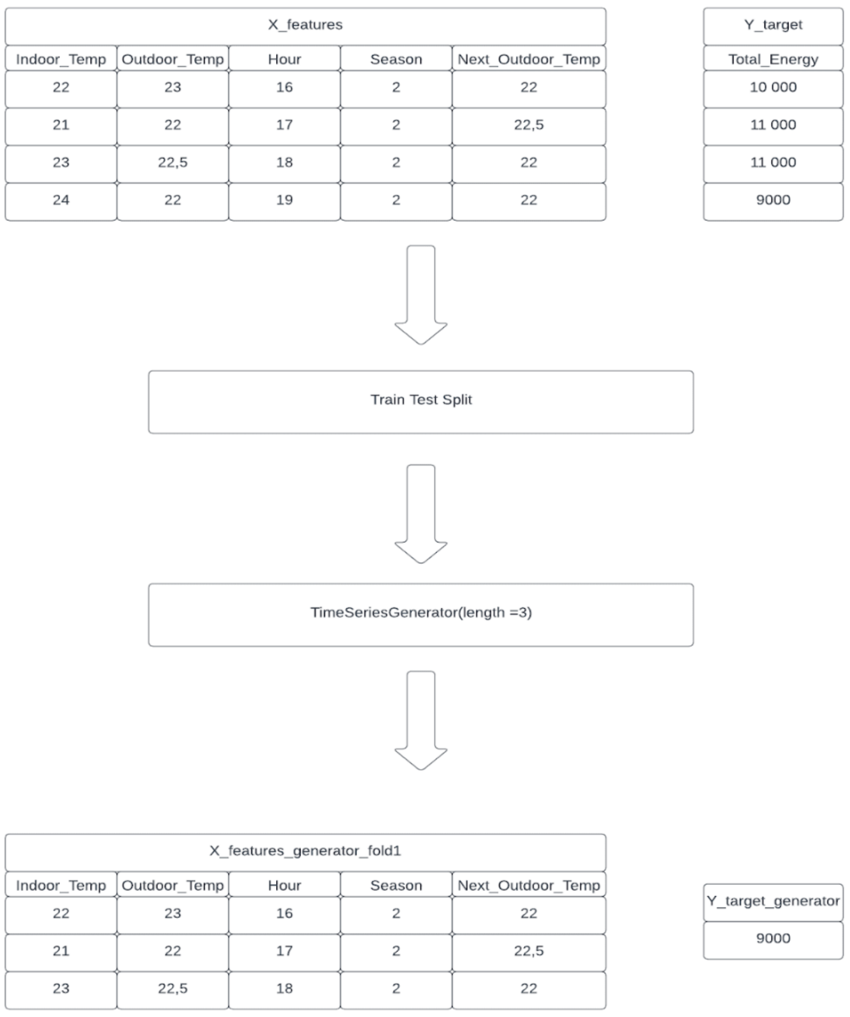
```
X_train, X_test, y_train, y_test = train_test_split(X,
y,test_size=0.1, random_state=123, shuffle=False)

win_length = 3
batch_size = 24
num_features = 8

train_generator = TimeseriesGenerator(X_train, y_train,
length=win_length, batch_size=batch_size)
test_generator = TimeseriesGenerator(X_test, y_test, length=
win_length, batch_size=batch_size)
```

Figure 3.8: The code for creating the train and test generator with a specified length and batch size.

TimeSeriesGenerator (TSG) is specifically designed for time series data and the key characteristic is that it will generate lags based on the length parameter. Using TSG has the advantage of finding temporal dependencies and has a good training efficiency since it generates overlapping sequences which provides more training samples. A length of three means that *X [1,2,3]* will be used to predict the *y[4]*. The difference compared to the ARIMAs approach of lags is that the LSTM will use the independent feature (X) instead of the target (y). When using this lag approach one key feature from the X will be missing, which is the outdoor temperature at that hour. To deal with this, a new column called next_outdoor_temp was created. The Figure 3.9 presents a simple overview of how TSG works.



X_features				
Indoor_Temp	Outdoor_Temp	Hour	Season	Next_Outdoor_Temp
22	23	16	2	22
21	22	17	2	22,5
23	22,5	18	2	22
24	22	19	2	22

Y_target
Total_Energy
10 000
11 000
11 000
9000

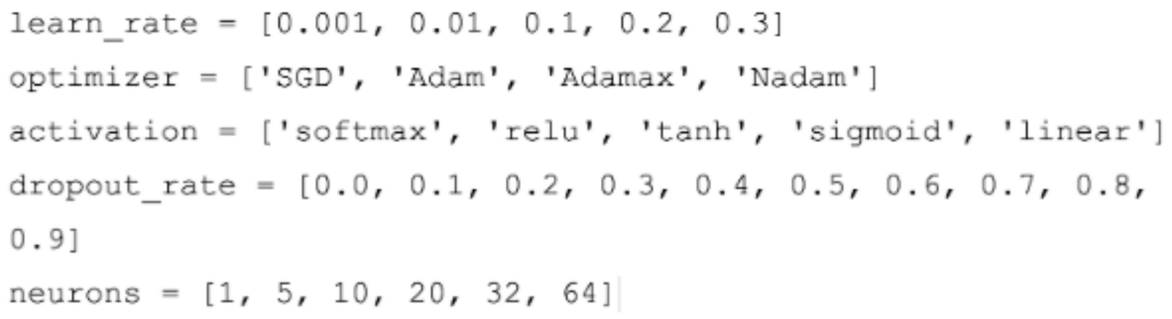
X_features_generator_fold1				
Indoor_Temp	Outdoor_Temp	Hour	Season	Next_Outdoor_Temp
22	23	16	2	22
21	22	17	2	22,5
23	22,5	18	2	22

Y_target_generator
9000

Figure 3.9: A simple representation of how the TimeSeriesGenerator divides the data into lags. The first three columns are used to predict the fourth target value. Note, that the column Next_Outdoor_Temp uses the value from the coming row of Outdoor_Temp.

The second approach was to simply split the data into X_train, X_test, y_train, and y_test. Since the LSTM requires the shape to be 3d the X_train and X_test was converted to 3d shape. For the TimeSeriesGenerator the X feature is not converted into a 3d shape because the TimeSeriesGenrator does it automatically. Both these two approaches of splitting were used for the training to see if the models perform any differently.

After a simple training with a basic construction was executed the train test validate method was used along with hyperparameter tuning to test different values for the number of neurons, learning rate, activation, and dropout rate. The hyperparameter tuning and train test validate method helped to build the most accurate model and to properly validate them. The chosen parameters are presented in Figure 3.10.



```
learn_rate = [0.001, 0.01, 0.1, 0.2, 0.3]
optimizer = ['SGD', 'Adam', 'Adamax', 'Nadam']
activation = ['softmax', 'relu', 'tanh', 'sigmoid', 'linear']
dropout_rate = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
0.9]
neurons = [1, 5, 10, 20, 32, 64]
```

Figure 3.10: The parameters and values used for hyperparameter tuning.

The LSTM model was constructed based on the result from the hyperparameter tuning and with some trial-and-error approach.

3.3.3 Ensemble LSTM - Building and Training

For the ensemble LSTM, five models with the exact same construction as for the simple LSTM were trained and the average of their prediction was used as the predicted value. Pseudocode of how the ensemble LSTM was build is presented in Figure 3.11. The idea behind this approach was to utilize the fact that different initial conditions on the five different models could lead to better predictions. This model was also trained with two different approaches as mentioned in the previous section.

```
function build_model()
    Builds the LSTM model
    Returns the model

models = []
For five iterations
    Call build_model()
    Train with the received model
    Append the model to the list models

y_pred = []
For five iterations
    Take the trained model from the list
    Make predictions
    Append to y_pred

Take mean
```

Figure 3.11: Pseudocode for building the ensemble LSTM.

3.3.4 AT-LSTM - Building and Training

The AT-LSTM model was constructed due to its proven success in time series forecasting and for the attention, the scaled dot product attention was implemented [11][10]. The model consisted of two main building blocks, the attention mechanism, and the LSTM. The scaled dot product attention is intended to give higher weights to independent variables that have a higher influence on the target [11]. Using the input shape of the time series data, the query, key, and value was created with the help of the Dense layer from keras library. The *relu* activation function was used with a specified unit size. Following that the scaled dot product was executed [10]. The LSTM model was built with the same parameters and layers as the previous LSTM models and was trained only with the simple train and test, using no lags.

3.3.5 XGBoost - Building and Training

The train test split method used for the XGBoost model divided the train and test data into equal amounts.

Hyperparameter tuning was made with the sklearn method *GridSearchCV*. The parameters analyzed were eta, gamma, max_depth, alpha, min_child_weight, and lambda. Eta is used to prevent overfitting and can be a value between 0 - 1. Gamma regulates how many splits the tree will make. Alpha and lambda specify how conservative the model will be. The min_child_weight parameter specifies the minimum number of samples that each child node needs to have. With the help of X_train and y_train the best value for each parameter was found.

The model was built with the help of XGBoost method *XGBRegressor* with all the hyper-tuned parameters among others like n_estimators, early_stopping_rounds, and learning_rate. The model was then trained using X_train and y_train. The loss function used was Root Mean Squared Error (RMSE) and the metric functions were Mean Absolute Error (MAE) and Symmetric Mean Absolute Percentage Error (SMAPE). The predictions from XGBoost were then compared with the actual data and visualized with plots.

3.3.6 Evaluation

The validation methods used were cross-validation, plotting and the metrics Mean Absolute Error (MAE) and Symmetric Mean Absolute Percentage Error (SMAPE).

Using MAE, and SMAPE was efficient when trying to improve the model since a lower score indicates a better model. So, each time new layers or neurons were added the focus was to see if the MAE, and SMAPE improves and as well as compare those values with the other models. In some cases, the metric MAE is not very effective to judge the models performances. In those cases, SMAPE was the metric utilized.

The first method used to evaluate how well the predictions were, was using a simple plot of the prediction and the actual values. Because the model is intended to predict on an hourly basis it is important to get a closer view rather than only checking the whole test dataset.

For that, multiple plots in an interval of 24 hours were made to see how well it performs. To get a better understanding of the models performances each season must be evaluated separately. To accomplish that, cross-validation was used.

Dividing the data into multiple folds of train and test, was done with the help of the method *TimeSeriesSplit* from sklearn. This method takes into account that data in time series format should not be able to predict past values using future values, which is why it preserves its original sequence. The TimeSeriesSplit for the first fold splits the data into a train and test and then for the next fold uses both the train and test from the first fold as train and shifts to new test data and so on. Figure 3.12 shows a TimeSeriesSplit with five folds. Using this approach, the model was validated for all four seasons. Those predictions accompanied by the actual values were then plotted for a better understanding of the performance and each fold was also validated using SMAPE and MAE.

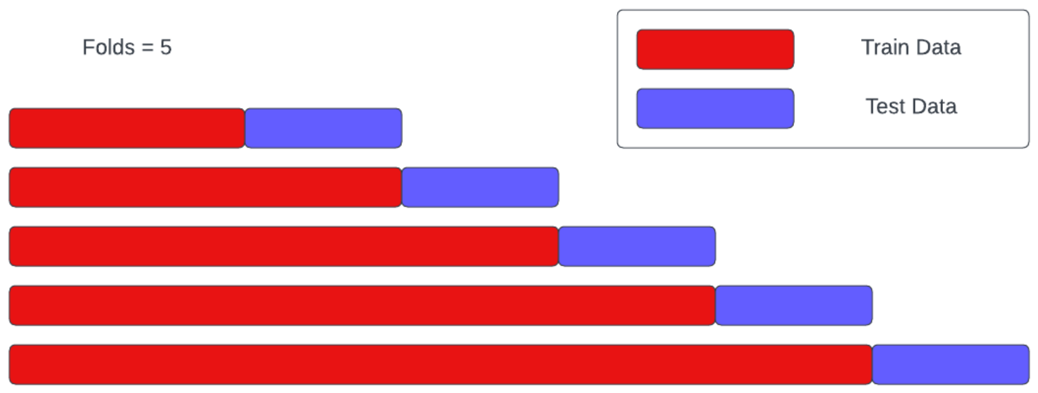


Figure 3.12: A representation of cross-validation using TimeSeriesSplit with five folds.

To evaluate the best model further evaluation was performed. The evaluation consisted of the model predicting the energy supply needed to achieve the indoor temperature of 21 degrees Celsius. Using one-year data, the original indoor temperature values were replaced with 21 and then used by the model to predict. With the prediction and the original data, a simple plot was used to get a visual comparison.

3.4 REST- API

A REST- API was built using the Flask library to simplify the process for external devices and applications to predict using the trained models. Saving the neural network models was done using the inbuilt save function whereas the XGBoost was saved using *pickle*. Two approaches were used to get the outdoor temperature, the first one lets the client specify the outdoor temperature and the second one fetches data from SMHI (Sveriges Meteorologiska och Hydrologiska Institut). To get the outdoor temperature for the area where the building is located, the coordinates for the nearest station in that area will be specified when fetching the data from the SMHI API. Three different API calls were made for the models, first one predicts the energy supply needed for a specific hour, the next one predicts the energy supply needed for each hour for a full day, and the last one predicts the energy supply for each hour for a whole week. Figure 3.13 below presents a simple overview of the architecture.

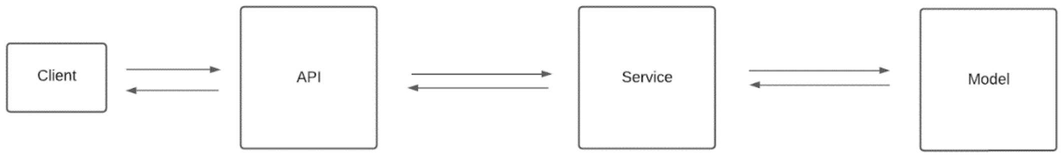


Figure 3.13: Architecture of the REST-API.

4 Results

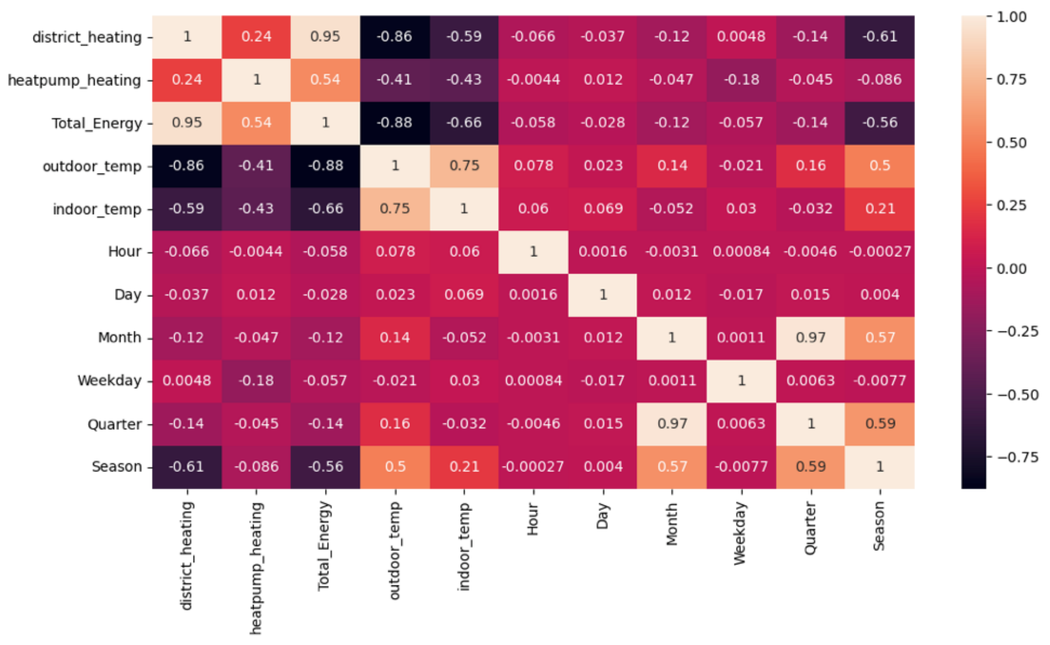
This chapter presents the result of the project. The time interval result is presented in section 4.1. In section 4.2 the result of the correlation analysis is presented. Sections 4.3 to 4.7 presents the performance of the different models. Lastly, 4.8 presents the best model prediction using 21 degrees Celsius.

4.1 Time Interval

The achieved results are from section 3.2.2. The decision to utilize hours as the time interval was due to the complications imputation gave and the poor performance of models using other time intervals. The number of rows with a value zero in the total energy column made the models using one-minute intervals and ten-minute intervals perform poorly. Another reason to use hours was that the building's energy system changes per hour.

4.2 Correlation and Boxplot

The achieved results are from section 3.2.4. The seaborn heatmap in Figure 4.1 below shows the correlation between the variables.



	district_heating	heatpump_heating	Total_Energy	outdoor_temp	indoor_temp	Hour	Day	Month	Weekday	Quarter	Season
district_heating	1	0.24	0.95	-0.86	-0.59	-0.066	-0.037	-0.12	0.0048	-0.14	-0.61
heatpump_heating	0.24	1	0.54	-0.41	-0.43	-0.0044	0.012	-0.047	-0.18	-0.045	-0.086
Total_Energy	0.95	0.54	1	-0.88	-0.66	-0.058	-0.028	-0.12	-0.057	-0.14	-0.56
outdoor_temp	-0.86	-0.41	-0.88	1	0.75	0.078	0.023	0.14	-0.021	0.16	0.5
indoor_temp	-0.59	-0.43	-0.66	0.75	1	0.06	0.069	-0.052	0.03	-0.032	0.21
Hour	-0.066	-0.0044	-0.058	0.078	0.06	1	0.0016	-0.0031	0.00084	-0.0046	-0.00027
Day	-0.037	0.012	-0.028	0.023	0.069	0.0016	1	0.012	-0.017	0.015	0.004
Month	-0.12	-0.047	-0.12	0.14	-0.052	-0.0031	0.012	1	0.0011	0.97	0.57
Weekday	0.0048	-0.18	-0.057	-0.021	0.03	0.00084	-0.017	0.0011	1	0.0063	-0.0077
Quarter	-0.14	-0.045	-0.14	0.16	-0.032	-0.0046	0.015	0.97	0.0063	1	0.59
Season	-0.61	-0.086	-0.56	0.5	0.21	-0.00027	0.004	0.57	-0.0077	0.59	1

Figure 4.1: Correlation matrix, showing the correlation between target variable and independent variables.

Variables Day, Weekday, Hour, and Month had similar correlations with total energy, they all correlated close to zero. A correlation with value zero means there exists no relationship between the variables and therefore are unrelated, the Day attribute had the worst correlation (-0.028). district_heating and heatpump_heating had a strong positive correlation with total energy, 0.95 respectively 0.54. A positive correlation means that both variables change in the same direction. Variables outdoor_temp, indoor_temp, and season (in that order) had a strong negative correlation with total energy. This means that variables change in the opposite direction, when one variable increases the other decreases. The attributes were further analyzed with boxplotting. The boxplot in Figure 4.2 represents the average energy supplied every day. The y-axis is the Total energy (Wh), and the x-axis is the days in a month.

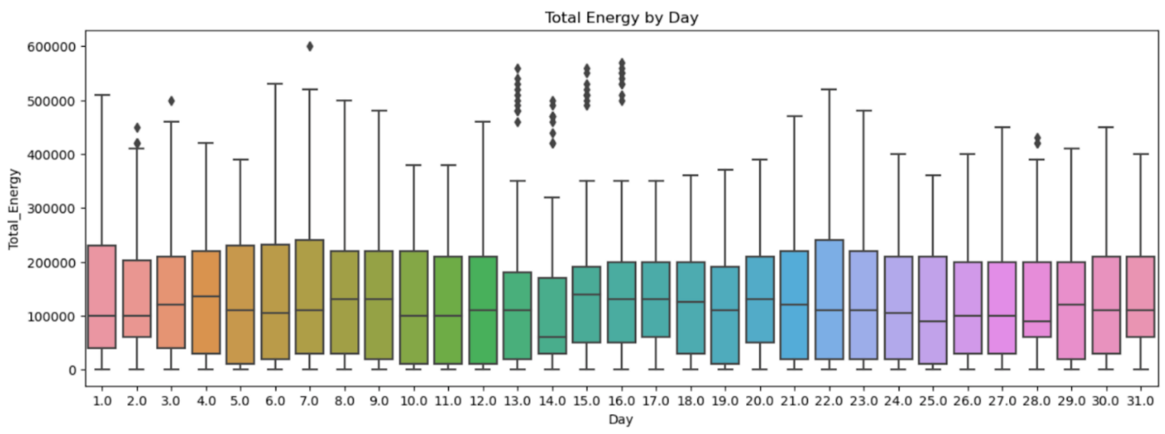


Figure 4.2: Boxplot showing energy supply per day.

This boxplot shows that the energy supplied per day follows no rhythm and is therefore of no use. Due to this factor, the Day variable is not included in the models inputs.

The final input features for all models were:

- Outdoor Temperature
- Indoor Temperature
- Hour
- Month
- Weekday
- Season

4.3 ARIMA

The Auto ARIMA method gave the p,d,q values of 2,1,4. This result means that the data is nonstationary and needs differentiation. That result opposes the result from the augmented Dickey-Fuller test which gave a p-value of 0.0019 which indicated stationarity. Plotting methods also indicated stationarity. An ARIMA model using p,d,q values 2,1,4 which was generated from the *auto_arima* was built and further some other values near 2,1,4 were also tested as a trial and error method. The figure below shows the performance of an ARIMA model using 2,1,4 as lags, predicted values (blue line) compared to the actual values (orange line).

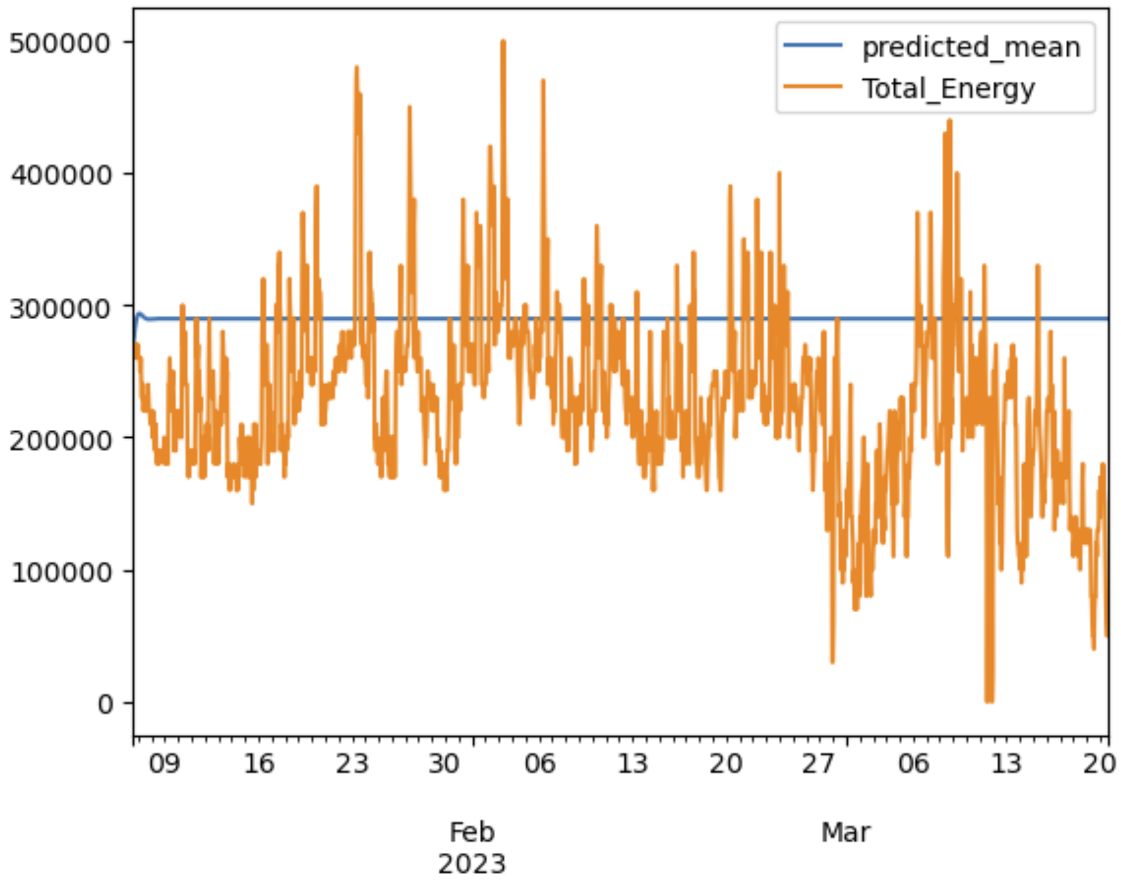


Figure 4.3: ARIMA performance compared with actual values. The y-axis is the Total Energy (Wh) and x-axis is the datetime.

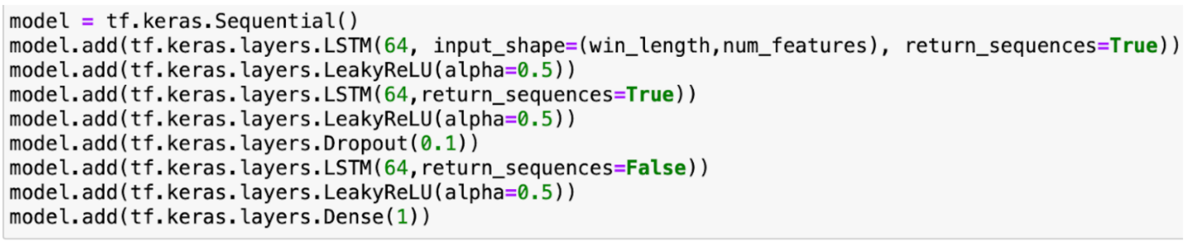
The mean absolute error and symmetric mean absolute error are presented in Table 4.1.

Table 4.1: MAE and SMAPE value for ARIMA.

MAE	SMAPE
74421.83	15.68 %

4.4 LSTM

The results from hyperparameter tuning using a set of chosen parameters shown in section 3.4.2 Figure 3.12 gave rise to a LSTM model with the structure shown in Figure 4.4.



```
model = tf.keras.Sequential()
model.add(tf.keras.layers.LSTM(64, input_shape=(win_length,num_features), return_sequences=True))
model.add(tf.keras.layers.LeakyReLU(alpha=0.5))
model.add(tf.keras.layers.LSTM(64,return_sequences=True))
model.add(tf.keras.layers.LeakyReLU(alpha=0.5))
model.add(tf.keras.layers.Dropout(0.1))
model.add(tf.keras.layers.LSTM(64,return_sequences=False))
model.add(tf.keras.layers.LeakyReLU(alpha=0.5))
model.add(tf.keras.layers.Dense(1))
```

Figure 4.4: The construction of the LSTM model using Tensorflow. The architecture consists of three LSTM layers with 64 neurons and following each layer there is a corresponding LeakyReLU layer with an alpha of 0.5. There is also a dropout layer with a drop rate of 0.1 and lastly the one output layer with one neuron. The best optimizer based on the results from hyperparameter tuning was Adam.

A train test split with a ratio of 90-10 using the TimeSeriesGenerator (TSG) with a length of three was the first LSTM model trained. Note that in this approach, due to lags, there is an extra input feature called next_outdoor_temp that has been added as mentioned in section 3.4.2. The result of the LSTM models performance using TSG is presented in Table 4.2.

Table 4.2: MAE and SMAPE value for LSTM using TSG.

MAE	SMAPE
29038.24	6.86 %

Figure 4.5 shows a plot of the model's prediction compared with the actual value for the whole test dataset.

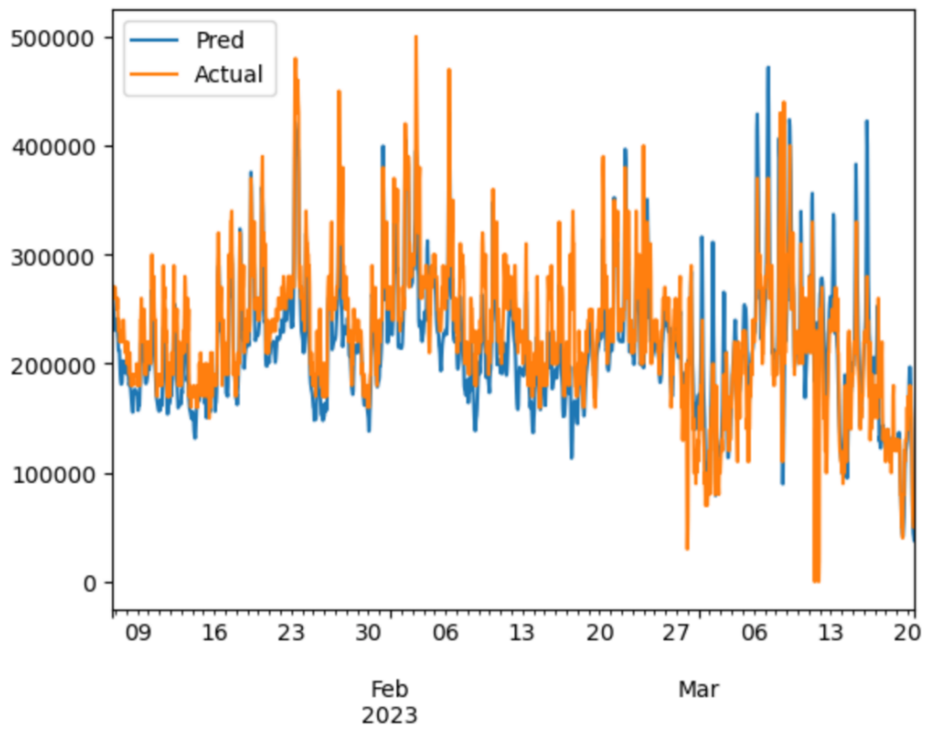


Figure 4.5: This is the plot result of the LSTM using TSG where the blue lines represent the prediction and orange lines represent the actual value.

Without using lags, only using the corresponding independent feature to predict its corresponding target value was the second approach as described in section 3.4.2. The performance of this approach using the LSTM model is presented in Table 4.3

Table 4.3: MAE and SMAPE value for LSTM using simple split.

MAE	SMAPE
27858.59	6.73 %

Figure 4.6 shows a plot of the model's prediction compared with the actual value for the whole test dataset.

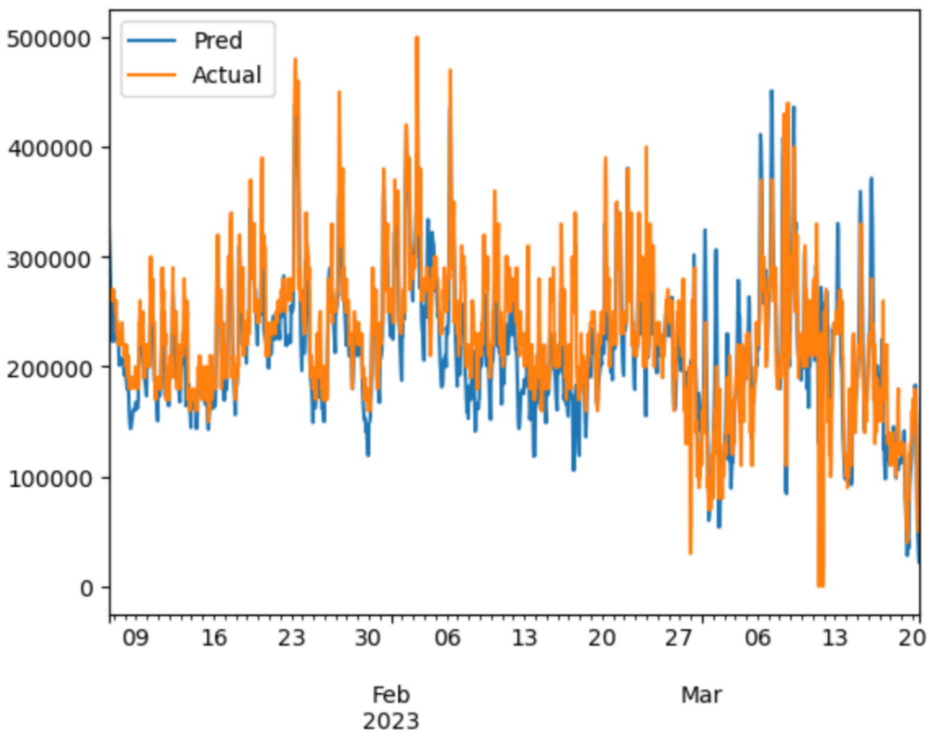


Figure 4.6: Simple split LSTM prediction compared with actual values.

The cross-validation method with five folds for the TSG approach yielded the result presented in Table 4.4

Table 4.4: MAE and SMAPE using TSG for each fold and the average is presented in the table.

	Cross-Validation - TSG					
	Fold:1	**Fold:2**	**Fold:3**	**Fold:4**	**Fold:5**	**Average**
MAE	18016.56	43206.72	13660.71	14269.83	25319.95	**22894.75**
SMAPE	25.94 %	9.37 %	25.95 %	26.79 %	5.37 %	**18.68 %**

The cross-validation method with five folds for the simple split approach yielded the result presented in Table 4.5

Table 4.5: MAE and SMAPE using the simple split approach for each fold and the average is presented in the table.

	Cross-Validation - Simple Split					
	Fold:1	**Fold:2**	**Fold:3**	**Fold:4**	**Fold:5**	**Average**
MAE	22378.76	37959.48	12907.53	11923.33	23686.25	**21771,07**
SMAPE	26.64 %	7.99 %	22.16 %	21.32 %	5.07 %	**16,63 %**

4.5 Ensemble LSTM

The result of Ensemble LSTM using the TSG is presented in Table 4.6.

Table 4.6: MAE and SMAPE value for ensemble LSTM using TSG.

MAE	SMAPE
23143.06	5.51 %

Its corresponding plot is presented in Figure 4.7.

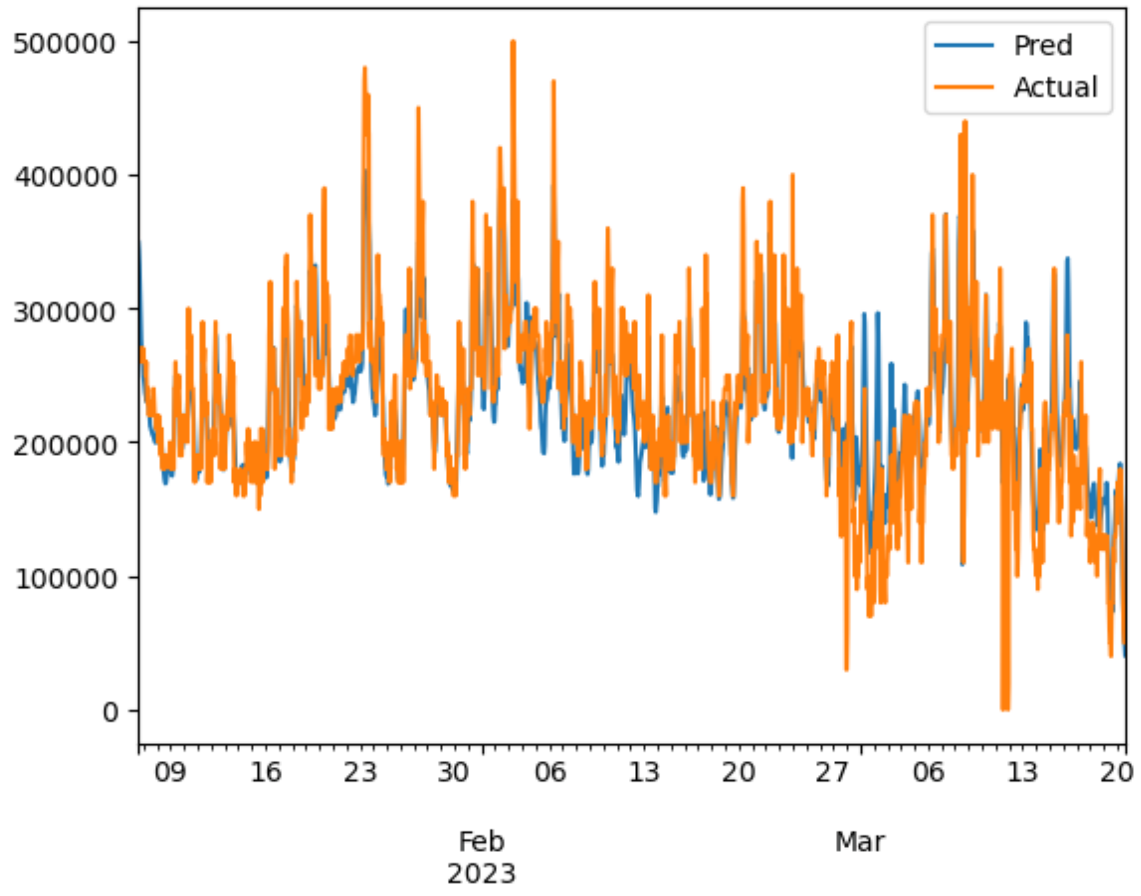


Figure 4.7: The prediction of Ensemble LSTM, with TSG, compared with the actual values.

The result for the simple 3d shaped train & test is presented in Table 4.7 and its corresponding plot in Figure 4.8.

Table 4.7: MAE and SMAPE value for ensemble LSTM using simple split.

MAE	SMAPE
22486.79	5.41 %

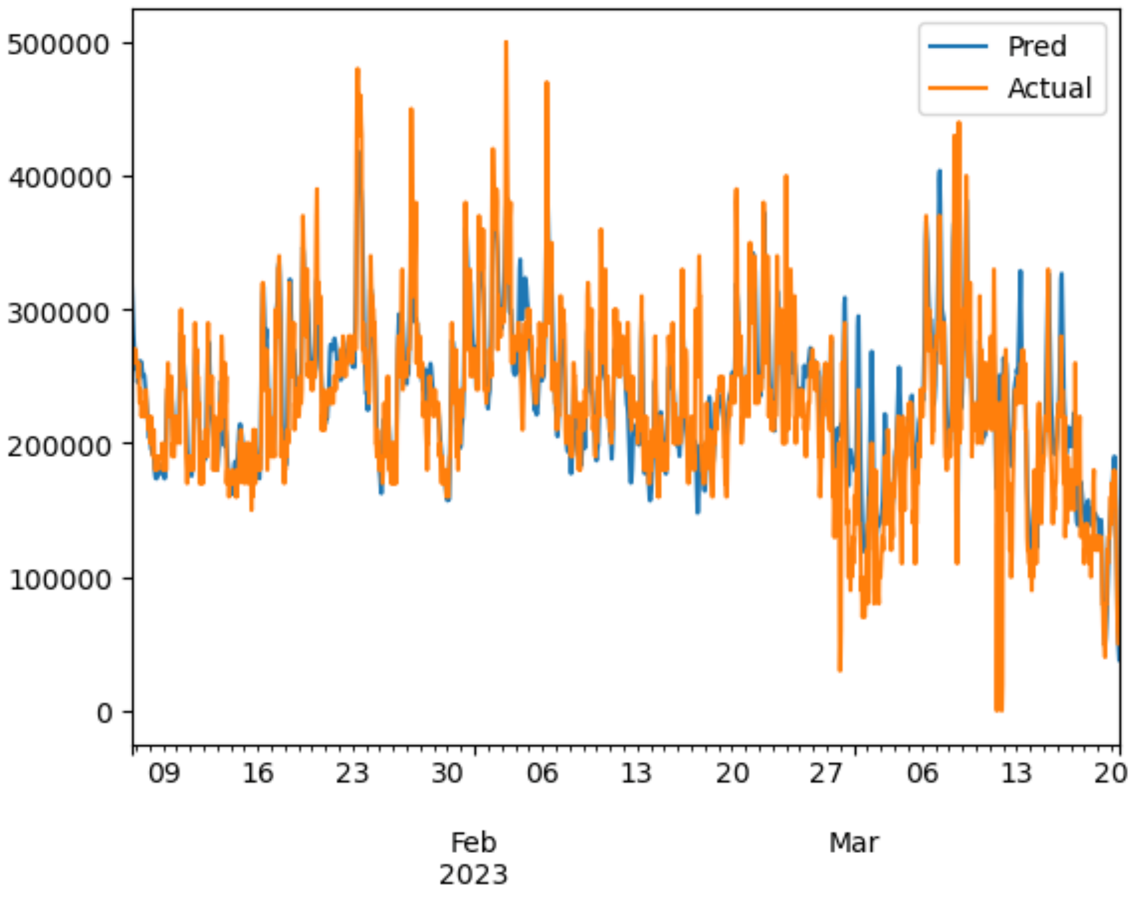


Figure 4.8: The prediction of Ensemble LSTM, with simple split, compared with the actual values.

Lastly the result for both approaches using cross-validation yielded the result presented in the Table 4.8 and Table 4.9.

Table 4.8: The cross-validation result for ensemble LSTM using TSG. MAE and SMAPE for each fold and the average is presented.

	Cross-Validation - TSG					
	Fold:1	**Fold:2**	**Fold:3**	**Fold:4**	**Fold:5**	**Average**
MAE	20658.13	43119.32	13085.60	12706.65	27051.20	**23324.18**
SMAPE	27.24 %	9.39 %	20.42 %	19.94 %	5,74 %	**16,54 %**

Table 4.9: The cross-validation result for ensemble LSTM using simple split. MAE and SMAPE for each fold and the average is presented.

	Cross-Validation - Simple Split					
	Fold:1	**Fold:2**	**Fold:3**	**Fold:4**	**Fold:5**	**Average**
MAE	19438.76	27737.01	13877.36	12191.99	25645.61	**19778,146**
SMAPE	26.40 %	8.16 %	21.46 %	21.20 %	5.47 %	**16.53 %**

4.6 AT-LSTM

The hyperparameter tuning results for the model AT-LSTM was using 64 neurons, 0.2 dropout rate, activation='relu', and optimizer='adam'. Attention-LSTM gave accurate predictions close to the actual values. The graph below shows the model's predictions compared with the actual values when using no lags.

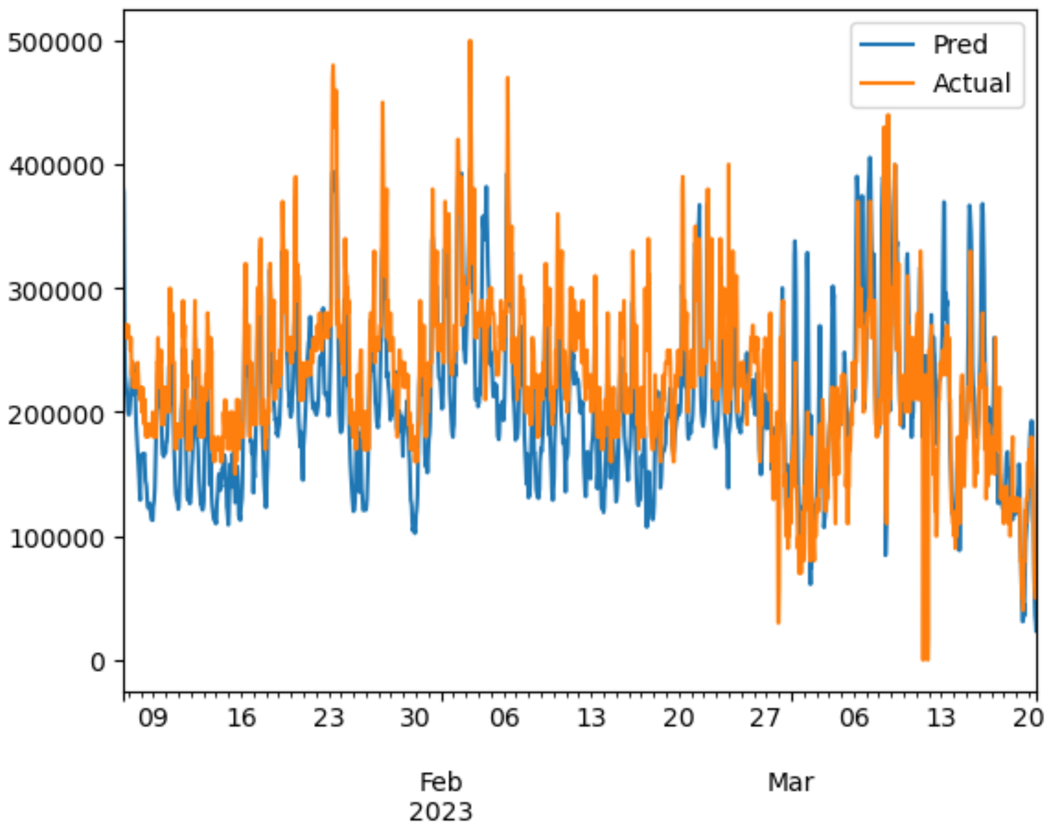


Figure 4.9: The blue line is the AT-LSTM prediction, and the orange line is the actual values.

The MAE and SMAPE for this model can be viewed in the Table 4.10 down below.

Table 4.10: MAE and SMAPE value for AT- LSTM using simple split.

MAE	SMAPE
44748.95	11.07 %

The outcome of the cross-validation method is presented in Table 4.11. The average of all folds is calculated for a better understanding of the overall performance.

Table 4.11: The cross-validation result for AT-LSTM using simple split. MAE and SMAPE for each fold and the average is presented.

	Cross-Validation					
	Fold1	Fold2	Fold3	Fold4	Fold5	Average all folds
MAE	38720.35	71004.91	19421.79	11241.84	59541.89	39986.16
SMAPE	39.00 %	17.26 %	26.38 %	19.09 %	14.54 %	24.16 %

4.7 XGBoost

Hypertuning of the XGBoost model gave the result:

```
{'alpha': 0.05,
 'eta': 0.05,
 'gamma': 0,
 'max_depth': 1,
 'min_child_weight': 0,
 'reg_lambda': 0.05}
```

These parameters were used to train the model. The graph below shows the result of the model's predictions compared with the actual values.

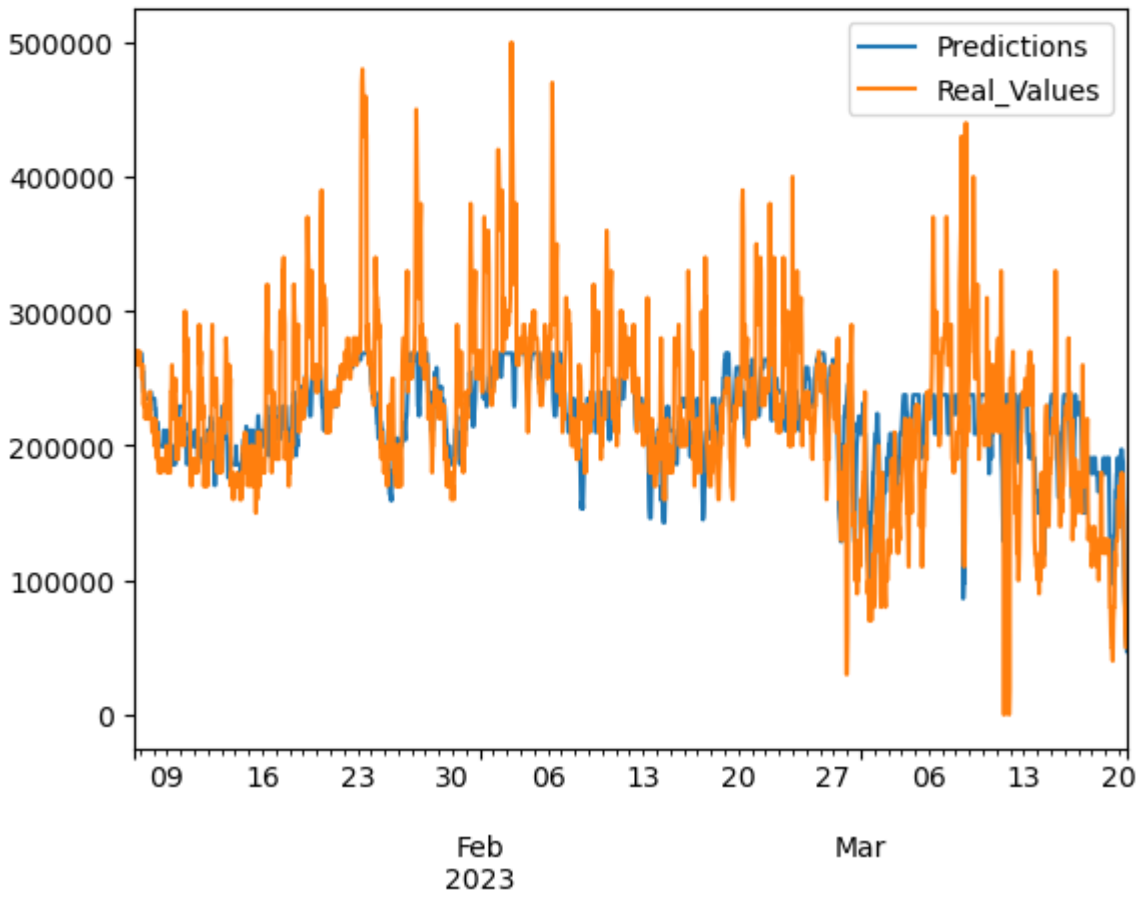


Figure 4.10: The XGBoost performance. The blue line is the XGBoost performance, and the orange line is the actual values.

Table 4.12 down below demonstrates the MAE and SMAPE for the model.

Table 4.12: MAE and SMAPE value for Xgboost.

MAE	SMAPE
36932.71	8.53 %

The MAE and SMAPE for the different folds in cross-validation are presented in Table 4.13. The average of all folds is calculated for a better understanding of the overall performance.

Table 4.13: The cross-validation result for XGboost using simple split. MAE and SMAPE for each fold and the average is presented.

	Cross-Validation					
	Fold1	Fold2	Fold3	Fold4	Fold5	Average all folds
MAE	36750.13	57630.93	62330.95	32778.71	22592.78	42416.70
SMAPE	47.88 %	25.11 %	43.31 %	48.27 %	51.04 %	43.12 %

4.8 Prediction with 21 degrees C

The Figure 4.11 presents the ensemble model prediction compared with the original data obtained from the company. The data used for the prediction is the original data with the modification of the indoor temperature to 21 degrees Celsius.

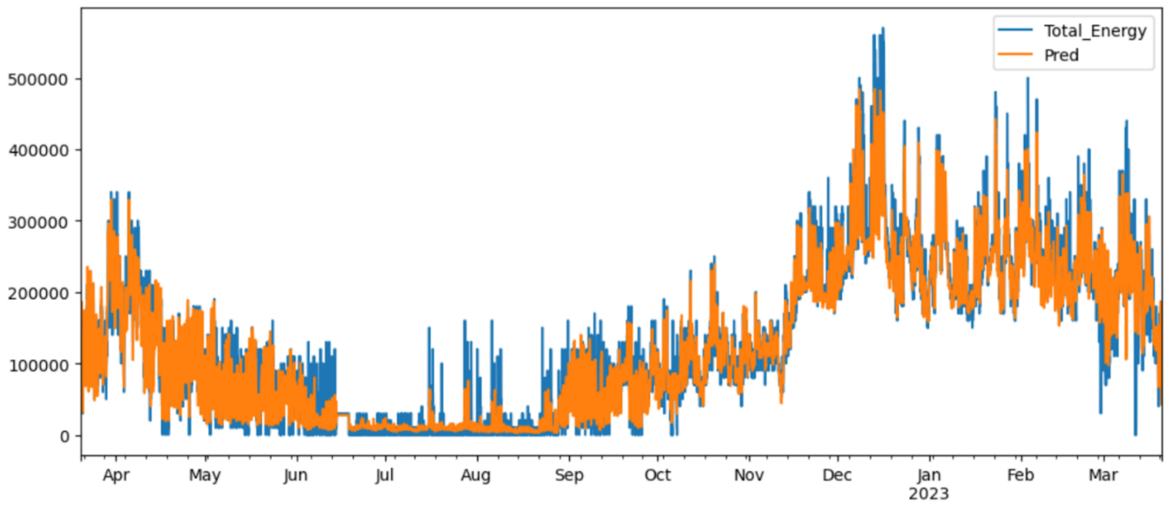


Figure 4.11: The prediction of the ensemble model using an inside temperature of 21 degrees compared with actual values. This figure shows data from a full year.

5 Analysis and Discussion

This chapter presents the analysis of the results achieved in chapter 4. The analysis will begin with interpreting the variables correlation and then move forward to analyzing the ML models performances and evaluation techniques. Lastly, alternative methodologies will be presented.

5.1 Interpreting variable correlation

By plotting and using heat maps it was possible to get a better understanding of the data. It was no surprise that variables district_heating and heatpump_heating had such a high correlation with total_energy. Outdoor_temp, indoor_temp, and season had a very high negative correlation with total_energy, this result was expected. When temperature decreases the energy supply grows to accelerate the heating of the building and when temperature increases less energy is needed for heating.

The variables Day, Weekday, and Hour gave results very close to zero, and after plotting these variables with respect to the energy supplied the result made more sense. The energy supplied does not have major changes for these intervals. Figure 4.2 in section 4.2 presents the energy supplied for each day. The daily change is not immense and results in low importance for the models predictions. For that reason, the Day attribute was removed from the models training data.

The Weekday attribute was chosen because we analyzed an office building. The mindset was to find patterns in the data due to no employees working on weekends. Figure 5.1 presents the energy supplied for each weekday.

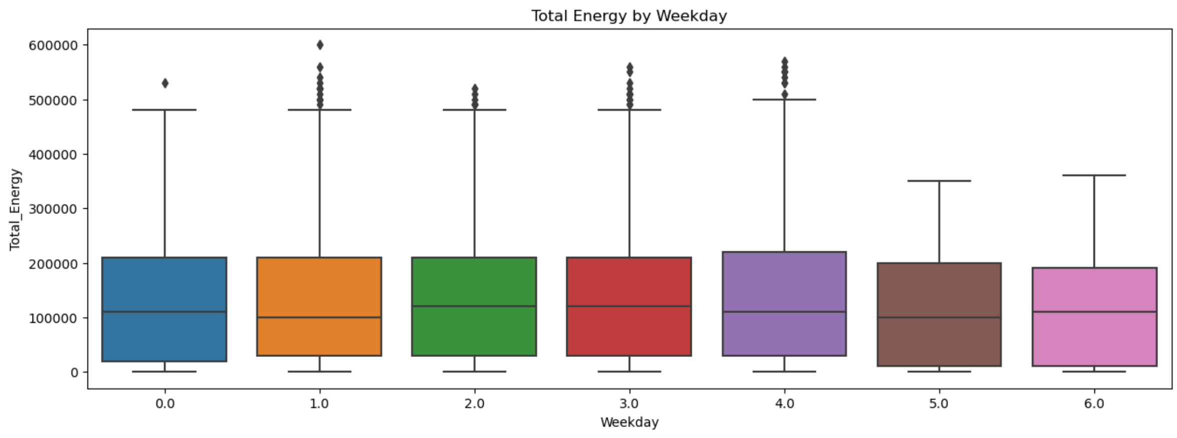


Figure 5.1: The energy supply per weekday.

As shown above, the energy supplied is usually higher on weekdays compared to weekends. The 50 percent which is the colored box and the median which is indicated with a line are very similar for both workdays and weekdays. This can be due to people on workdays contributing to heating the building and can also be data from the summer. Whereas for the rest 25 percent of the data which is above the colored box a clear difference can be seen between weekdays and workdays. Therefore, the Weekday attribute was included in the training data.

The variable Hour was intended to find patterns of energy supplied during non-working hours. Figure 5.2 presents the energy supplied for each hour.

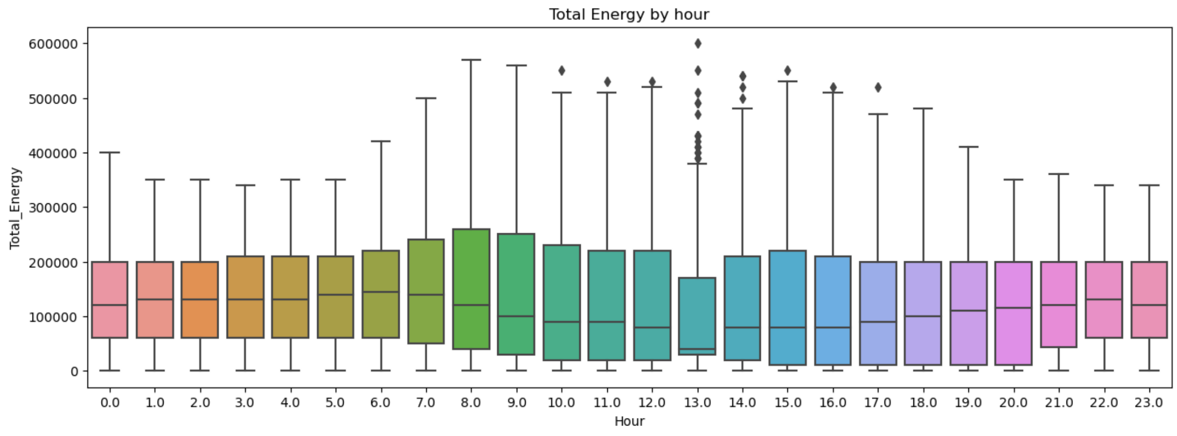


Figure 5.2: The energy supply per hour.

The plot shows that the energy supply goes slightly up around 6 o'clock and then down again around 18 - 19. This was the pattern we were searching for using the variable Hour and that is why we chose to include it in the training data even though it has a low correlation with total energy.

5.2 Performance of the machine learning models

First of all, for the ARIMA model, the metrics and the plot convey that the model did not perform well. There can be several reasons for this, firstly the dataset, two years of data probably is not enough for an ARIMA model to perform well. The second reason is that using only the previous y value, maybe is not enough for solving this problem. There are a lot of other factors which are not taken into consideration such as the outdoor temperature. It is important to note that the value of energy supply has those influencing factors such as outdoor temperature baked into itself but having them as inputs will give them a higher focus which is missing with the ARIMA model. In conclusion the ARIMA model was not suitable for the dataset in hand. For that reason, there was no additional hyperparameter tuning or cross-validation executed for this model.

The XGBoost was somewhat unique, it performed better when the train data was less. The result of XGBoost presented in the result section is a 90-10 split and this was done so that the models can be compared with the same test data. But the XGBoost performs best for a split of 50-50. A positive aspect is that the XGBoost can perform better than the rest of the models using less data. The Same 50-50 approach was performed with the rest of the models and they were not able to perform equally well.

For the LSTM and ensemble model two approaches were performed. Both performed equally well but it was possible to see that the simple way of splitting by not using any lags performed slightly better for both the models. This can be seen by comparing the result between the models from both train test splits of 90-10 and the average result from cross-validation. For this case using a simple way of splitting was more efficient than using TSG.

The main reason for choosing TSG apart from that it is mainly designed for time series, was the advantage of overlapping sequences which was believed to be helpful since the data in hand was very small. Even though the TSG approach didn't perform that poorly, the simple way of splitting gave a slightly better result for this problem. But it doesn't mean that the TSG will be ignored, if future research is performed in this field, for example, if more dataset is collected in the future and new training is executed, both approaches will be tested. TSG has its advantages when it comes to finding temporal dependencies and when having a large dataset since it may not be practical to use the entire sequence as input.

Moving forward, of the three LSTM models when comparing the result regardless of the approach chosen, there is a clear indication both from the MAE, SMAPE, and from the plotting that the ensemble LSTM is the model that performs best among them.

The most anticipated model AT-LSTM was the least-performing model among these three. As mentioned in section 2.6 the main property of the attention is to give higher weights to the variables with higher influence on the target and it is very clear from the metrics and the plot that the model failed to achieve that. Comparing the result from AT-LSTM with simple LSTM it is evident that the LSTM using equal weights to all input values overperformed the AT-LSTM. The most probable reason for this is that the attention focused on the wrong variables and gave them higher weights, this hypothesis is considered the most probable case since it performed worse than the simple LSTM. Why the attention could have focused on the wrong variables could have been that it was fed with a very small amount of data and due to that it failed to recognize the actual contributing variables and maybe saw some short-term factors of some variables and decided to give such variables higher weights.

Both the LSTM and ensemble LSTM performed well but the ensemble model was one step ahead, overall, it was off with 1% less and has an MAE difference of 2000-3000 on average (this includes both train test split train and cross-validation). The strength of the ensemble LSTM is that it is built upon the LSTM construction that performed decently well and when having multiple of them with different initial conditions they can capture different perspectives and sources of variation in the data. This leads to improved generalization, and robustness and also reduces the risk of overfitting to specific patterns in the training data. It is noticeable from the result that the ensemble managed to succeed with that.

In summary, the result shows that the ensemble LSTM performs best. The plot shows how well the ensemble model predicts compared to the actual and according to SMAPE the model is off with 5.41% (simple split approach) and for cross-validation it is 16.53%. This is the best score achieved among all the models.

5.3 Ensemble-LSTM for 21 degrees C

Ensemble LSTM proved to be the most efficient model and was, therefore, further evaluated to analyze the generalization of the model and how suitable it is to achieve the goal. The model was trained on a dataset where the indoor temperature was in the range of 19-26, but the model will mainly be used to predict the energy supply needed to get 21 degrees. The result of the estimation of how well the model has generalized can be seen in Figure 4.11, section 4.8. It can be noted that the energy prediction is slightly lower than the original which seems logical since more than 30 percent of the original dataset contains an indoor temperature over 22 degrees. This is a good indication that the model considers the indoor temperature when predicting, but a much deeper analysis must be performed. Further investigation with the help of plotting could have been done, such as comparing the temperature movements with the predicted energy supply to see if the model predicts lower when it should.

A more precise way of evaluating the predictions is by either making a simulation or using the model in practice. The problem is that knowing with certainty that the predicted amount will lead to 21 degrees is impossible if not tested.

5.4 Interpreting the Performance

MAE is a very good metric to get a value-based representation of how much the average error is. For example, for the ensemble LSTM with a simple split, the MAE was around 22 000, which indicates that the model would predict with an average error of that value. This gives a good understanding of how well the model performs when taking into consideration that the values in most cases exceed 200 000. Being off with 22 000 is very decent and can of course be improved. The SMAPE is similar but is used to give a percentual representation and this can give a quicker understanding of the performance since it directly says in percentage how off the model's prediction is whereas for the MAE some self-analysis has to be made. But interpreting the metrics is not as straightforward and at the same time they might not be the best way to evaluate in this case.

Throughout the previous section the word “best”, and ”poor” has been mentioned for performance but it is a very vague way of expressing the models performances. Continuing with the previous example, having an ensemble model with MAE 22 000, it is important to note that when comparing each predicted value with the actual there will be instances where the error exceeds or subceeds the MAE. The point that is tried to be conveyed is that those hours where the prediction is way off are not visible in the metrics and can, if deployed in the real world, be very problematic. It was for this reason a simple plotting method was chosen, the figures presented in the result section are of the whole test dataset but when the plot was analyzed multiple plots, each with 24 intervals, were made. But that was a very tedious approach, plotting test data split into 24-hour intervals is a lot of plots and at the same time hard to manually analyze.

Lastly, cross-validation as mentioned in section 3.4.4 was to test the model on different datasets and specifically how the model performs during different seasons. It was possible to get a good overview of how the model performed during different seasons both using the metrics and plots. Using the cross-validation it was possible to note that all the LSTM models performed worst for the folds two and five, and these two folds were the winter season. There were two restrictions faced with the cross-validation, the first one was the size of the dataset and the second one was the metric SMAPE.

When predicting on some test folds the test data was unseen, it is understandable that the test data should be previously unseen but, in this case, it was trained on data during the winter season and predicted for the spring. This applies only for the two first folds because from fold three and onwards the model had seen a full year. The SMAPE was a good metric to get a percentage representation, but it had one drawback, it was sensitive to zero values.

Our dataset had zero values mainly during summer seasons, in most hours no energy is supplied for heating the building. The SMAPE was contradicting the MAE in folds three and four (summer season) and it was mainly for the reason of zero values.

Overall, by using two metrics, cross-validation and plotting for evaluation, it was possible to be as diversified as possible so that the evaluation for the model is not dependent on one method and can be as good as possible from different perspectives.

5.5 Alternate Methodology

The alternative methodologies mentioned below lead to further investigation and discussion.

5.5.1 Building Architecture

As mentioned in Chapter 3 data about the room sizes was not available. If this data was provided a method called clustering would be a good approach for dealing with this problem. By putting rooms, with similar sizes, into clusters the rooms would not be treated equally. A room with 80 m^2 needs more energy supply than a room of 20 m^2. Rooms often have different sizes, and the project would have been more realistic if the model would have taken this into account.

5.5.2 Metric for Evaluation

The metrics chosen were mean absolute error and symmetric mean absolute percentage error, these were the most suitable for our five models. It happens though that these metrics sometimes contradict each other and give us an uncertain result. A third metric would be a good solution, but it wasn't found.

5.5.3 Alternate ensemble model

As mentioned above in section 5.2 the ensemble model was the most accurate model. A sixth model that consists of both XGBoost and LSTM could be tried. The XGBoost predictions could be used as extra input for the LSTM models and in that way help the LSTM to predict more accurately. Alternatively using an ensemble approach where each model is responsible for predicting the energy supply for one season could be tried. Using such a model can improve the accuracy since the model is focused on only one season.

5.6 Projects impact on economy, social, ethics, and environment

By using a ML model that make accurate energy supply predictions instead of hiring consultants to regulate it manually will make an economic impact on the company in charge for the energy supply. The company will no longer need to pay unnecessary salaries and can focus on further improvements. Looking from an environmental point of view a well working ML model will make more accurate predictions than a human and by that contributing to less spill of energy. The change of energy regulation will have no impact on a social aspect as no person will get in hand with it.

From an ethical aspect using ML models can be both good and bad. Some employees may lose their job and can be seen as bad but on the other hand less energy will be wasted, and less damage will be made on the greenhouse effect. If those two consequences get compared to each other, the use of a ML model makes more good than bad.

6 Conclusions

This thesis has evaluated five machine learning models that predict the energy supply needed to regulate the inside temperature of smart buildings. The first goal was to analyze the time series data, and this was successfully done using feature engineering and TSA, but there's room for improvement. Of those models, ARIMA was not suitable for this problem. Both the ensemble LSTM model and the XGBoost model gave promising results. The comparison of the models showed that the ensemble LSTM, using the simple way of splitting, was the most precise one with an MAE of 22486.79 and SMAPE of 5.41 %. A limited comparison between the current system and the ensemble LSTM was performed.

Three out of four goals were accomplished as expected. Comparing the ensemble LSTM with the current system needs further analysis, this could not be done due to it being too time intensive for the scope of the project.

In conclusion, machine learning can be useful for predicting the energy supply in smart buildings but needs a proper evaluation before deployment.

6.1 Future work

In the process of analyzing the data, there were data points that was suspected to be outliers but could not with certainty remove them. With an expert in the field, these suspicions could be confirmed and removed. By removing the outliers, the model will be more generalized.

One problem constantly repeated throughout the report was that the dataset was too small. That could have been an affecting factor as to why models such as ARIMA and AT-LSTM performed as poorly as they did. In the future when more data is available the models performances can change.

A better evaluation technique is needed to determine if the trained model can actually replace the current system. For that a simulation could be done, the important aspect of the simulation is to be able to confirm, with high precision, that the model's prediction leads to an indoor temperature of 21 degrees Celsius.

References

[1] Liza LY. EnergiLäget [Internet]. 2023 [updated 2023-03-27, cited 2023-04-07]. Available from:
https://www.energimyndigheten.se/statistik/energilaget/

[2] Auffarth B. Machine learning for Time-Series with Python [Internet]. Edition 1. 2021. [cited 2023 Mar 25]. Available from:
https://learning.oreilly.com/library/view/machine-learning-for/9781801819626/Text/Chapter_2.xhtml#_idParaDest-32

[3] J Hyndman R, Athanasopoulos G. Forecasting: Principle and Practice [Internet]. Edition 2. 2018. [cited 2023 Mar 25]. Available from:
https://otexts.com/fpp2/tspatterns.html

[4] Peixeiro M. Time Series Forecasting in Python [Internet]. Edition 1. 2022. [cited 2023 Mars 27]. Available from:
https://learning.oreilly.com/library/view/time-series-forecasting/9781617299889/Text/01.htm#heading_id_5

[5] Kostadinov S. Recurrent Neural Networks with Quick Start Guide [Internet]. Edition 1. 2018. [cited 2023 Apr 6]. Available from:
https://learning.oreilly.com/library/view/recurrent-neural-networks/9781789132335/117e47a3-eaac-4a6a-90d8-b14ee0f1ab16.xhtml

[6] Ninagawa C. AI Time Series Control System Modeling [Internet]. Edition 1. 2022. [cited 2023 Apr 5]. Available from:
https://link-springer-com.focus.lib.kth.se/chapter/10.1007/978-981-19-4594-6_4

[7] ScienceDirect [Internet]. 2019. [cited 2023 Apr 6]. Available from:
https://www-sciencedirect-com.focus.lib.kth.se/science/article/pii/S0950705119302400

[8] IEEE [Internet]. 2017. [cited 2023 Apr 3]. Available from:
https://ieeexplore-ieee-org.focus.lib.kth.se/document/7508408

[9] Manu. A simple overview of RNN, LSTM, and Attention Mechanism [Internet]. 2023. [cited 2023 May 18] Available from:
https://medium.com/swlh/a-simple-overview-of-rnn-lstm-and-attention-mechanism-9e844763d07bg/

[10] Google. Attention is all you need [Internet]. 2017. [cited 2023 Apr 7]. Available from: https://arxiv.org/pdf/1706.03762.pdf

[11] Information school, Renmin University of China, Beijing. AT-LSTM: An Attention-based LSTM Model for Financial Time Series Prediction [Internet]. 2019. [cited 2023 Apr 7]. Available from: https://iopscience.iop.org/article/10.1088/1757-899X/569/5/052037/pdf

[12] Kyriakides G, Margaritis K. Hands-On Ensemble Learning with Python. 2019. [cited 2023 Apr 8]. Available from: https://learning.oreilly.com/library/view/hands-on-ensemble-learning/9781789612851/9757ba44-2449-4ae3-ac3b-1730dae271e3.xhtml

[13] Kumar A, Jain M. Ensemble Learning for AI Developers: Learn Bagging, Stacking, and Boosting Methods with Use Cases. 2020. [cited 2023 Apr 8]. Available from: https://learning.oreilly.com/library/view/ensemble-learning-for/9781484259405/html/489264_1_En_4_Chapter.xhtml#Sec1

[14] University of Washington. XGBoost: A Scalable Tree Boosting System [Internet]. 2016. [cited 2023 Apr 20]. Available from: https://dl.acm.org/doi/pdf/10.1145/2939672.2939785

[15] Shumway R, Stoffer D. Time Series Analysis and Its Application. 2017. [cited 2023 Apr 7]. Available from: https://link.springer.com/book/10.1007/978-3-319-52452-8

[16] Brockwell P, Davis R. Introduction to Time Series Forecasting. 2016. [cited 2023 Apr 7]. Available from: https://link.springer.com/book/10.1007/978-3-319-29854-2

[17] ScienceDirect [Internet]. 2018. [cited 2023 May 24]. Available from: https://www.sciencedirect.com/science/article/pii/S0360544218319145

[18] Journal of Physics.Forecast Energy Consumption Time-Series Dataset using Multistep LSTM Models [Internet]. 2021 [cited 2023 May 24]. Available from: https://iopscience.iop.org/article/10.1088/1742-6596/1933/1/012054/pdf

TRITA – CBH-GRU-2023:097