FACULTY OF SCIENCE AND TECHNOLOGY

MASTER THESIS

Study programme / specialisation:
Engineering structures and materials /
Mechanical Engineering

The spring semester, 2022

Open

[illegible]
(signature author)

Author: Ali Awada

Course coordinator:

Supervisor(s):
Charlotte Obharai (UiS)
Mads Hansen (Kube Energy)

Thesis title:
Models for optimising power consumption to generation in captive solar projects

Credits (ECTS): 30

Keywords:
LSTM, Linear regression, simple linear regression, multiple linear regression, ARIMA, ETS, machine learning, recurrent neural networks, forecasting, time series analysis

Pages: 38

+ appendix: 22

Stavanger, 15-JUN-2022

1 Table of Contents

2 List of Figures

3 List of tables

4 Abstract

Kube Energy is a renewable energy services company that has most of its solar power plants planned around a single or multiple off-takers (end-users) connected in a small mini-grid. What is common for these off-takers is that a large proportion of their power consumption is to cover air conditioning and water heating for offices and residences. To date, Kube has designed their systems with larger battery banks or spinning capacity in diesel generators to provide stability in the grid during the day, as well as to provide electricity during the night.

This year, Kube Energy has started a project with an aim at optimizing consumption of electricity to better match generation. This mainly entails scheduling loads for when there is high production from the PV (e.g., reheating water in hot water tanks or charging electrical vehicles) or cutting loads if there is an unscheduled decrease in generation (clouds). In this effort, Kube Energy is looking at methods for forecasting consumption and generation. This thesis will explore multiple methodologies for consumption forecasting in the UNHCR offices in Kukuma, Kenya to help Kube build a power management system (PMS) that will improve the profitability of such system by reducing the gap between generation and consumption of electricity. An important feature of the PMS is to ensure adequate balance between power generation and consumption to avoid load shedding, in case power consumption is larger than power production, or other strategies when other type of load variations take place.

The thesis looks at parametric and non-parametric models and compares the models' prediction accuracy to choose the best performing or reliable one. The models used in this thesis are the Auto Regressive Integrated Moving Average (ARIMA), Error Trend and Seasonality (ETS), linear regression (simple and multiple), and Long-Short Term Memory (LSTM). The LSTM outperforms the others significantly and was selected as the most reliable one for forecasting consumption values at the site in question. The decision was based on the performance metrics of the models. The non-parametric model or LSTM had a better performance than the two remaining parametric models due to many reasons. One of the advantages of the LSTM is the ability to tune hyperparameters to obtain a well performing forecasting model, giving flexibility to the user if overparameterization is avoided.

5 Prediction methods

The prediction horizon, the time between two consecutive data points, and the purpose of the prediction played a role in determining prediction methods. The purpose of the prediction is to better handle and manage the power produced by the solar PVs and to reduce the use of the generator. Consequently, the prediction horizon will range from hours to days. Therefore, the prediction horizons are classified as short-term and medium short-term prediction [1]. Parametric models, including ARIMA, ETS, and linear regression models, and non-parametric model (LSTM) were chosen, model assumptions tuned, regression coefficients estimated, and results tested for robustness.

Most of the methods are considered univariate auto-regression because historical data of the consumption variable, regressor variable, are used to estimate future consumption value, response variable. The remaining methods include linear and multiple linear regression since the methods use variables or regressors such as weather and dummy variables to get a better fit.

The terms *predicted* and *forecasted* shall be distinguished in the following manner throughout the paper: The term *predicted* shall be used to refer to the data obtained from the training data and the *predicted* data will be compared with the actual data (testing data) to assess the performance of the model. The term *forecasted* shall be used to refer to the future data obtained after the model has been trained and validated.

6 Data source

Power consumption data is obtained from a UN office in Kakuma, Kenya. Credentials to access and download the data are provided by Kube Energy. The first ever recorded power consumption data is on January 18, 2022. There is a data blackout from 10:00 PM on 09 February 2022 until 4:00 PM on 22 February 2022. Then, there is a blackout from 05 April 2022 until the present day of editing/writing the thesis that is 31 May 2022. The sampling rate (data frequency) as obtained from the website is 1 hour. The unit of collected data is inconsistent because it is not consistently reported in the same unit. Data pre-processing and data treatment is explained in 7.2.

time	watt_hour
18-01-22 0:00	26.1 Wh
18-01-22 1:00	30.4 Wh
18-01-22 2:00	27.5 Wh
18-01-22 3:00	27.5 Wh
18-01-22 4:00	6.46 kWh
18-01-22 5:00	27.5 Wh
18-01-22 6:00	3.95 kWh

Table 1: Data snippet from the Excel file

As for the weather data, the values were collected from the website www.meteostat.net in which the location was set to be that of Nairobi because of the lack of data in the offices' location in Kukuma.

	count	mean	std	min	25%	50%	75%	max
Consumption(kWh)	1038	4.315012	4.931249	0.00	0.554250	2.200000	7.022500	29.10
Temperature(°C)	1038	21.552023	4.063941	0.00	18.00	21.00	25.00	30.00

Table 2: Summary of power consumption and temperature data

6.1 Tools and Programs

The following tools and programs were used in this paper:

R/RStudio: *R* is an open-source programming language used for statistical computing and graphics supported by the *R Core Team* and the *R Foundation for Statistical Computing*. RStudio is a third-party open-source graphical user interface. These tools are used in the parametric models of this study.

Python: is an open-source general purpose programming language. It is used to derive the non-parametric model (Machine learning part).

7 Data

7.1 Data types

Two of the most common statistical data types used are the time series and the cross sectional data [2]. While the cross-sectional data is recorded at one point in time, the time series data is one measurement over time [3]. The main difference between these two types is that in time series data, unlike in cross sectional data, the ordering of the data is crucial [2]. An example of cross sectional data is the gross annual income for a certain number of randomly chosen households in Stavanger for the year 2022. While an example of time series data is the wind speeds collected at a certain height over a period of time. The data analyzed in this study is labeled as a time series data since its values are recorded in a fixed frequency over a certain time interval.

7.2 Data pre-processing

As mentioned earlier, each power data cell is a string made up of the energy consumption value and the unit of consumption (Wh or kWh). Pre-processing is done in two steps. The first step is done directly on the raw data on Excel while the second step is done in Python. The first step is extracting the numerical part of the string and converting it to kWh. The second step handles zero and NULL values. In this study, zero or NULL values are replaced with the mean of the time series had it not have neither NULL nor zero values. This is known as mean imputation. For example, assuming that the data was the one in the left table in Figure 1 then the zero values would be deleted leading to the new data shape shown in the middle table. Here, the average of the non-zero values is calculated and then this value is replaced with the zeros in the left table to produce the final treated data in the right table of Figure 1.

time	watt_hour
01-02-22 3:00	0.976
01-02-22 4:00	0
01-02-22 5:00	5.72
01-02-22 6:00	0.878
01-02-22 7:00	0.819
01-02-22 8:00	0

$\rightarrow$

time	watt_hour
01-02-22 3:00	0.976
01-02-22 5:00	5.72
01-02-22 6:00	0.878
01-02-22 7:00	0.819

$\rightarrow$

time	watt_hour
01-02-22 3:00	0.976
01-02-22 4:00	2.09825
01-02-22 5:00	5.72
01-02-22 6:00	0.878
01-02-22 7:00	0.819
01-02-22 8:00	2.09825

Figure 1: Data imputation process. The average of the non-zero values of the middle table is 2.0985 and it replaces the zero values in the first table to get the treated data as shown in the right table

After collecting the temperature data, a Python code was written to align the collected weather data with the existing consumption data with respect the time column at which the consumption was collected using the $pandas$ library in Python. Then the data was saved in an Excel file in which two dummy variables were created. The first dummy variable is a binary variable called isWeekend that identifies whether the data and time fall on a weekend. The second dummy variable is also a binary variable called AMPM that identifies whether the time of the day is between 9:00 AM and 6:00 PM, inclusive. The time column format in the dataset has a character structure and must be converted to a date structure.

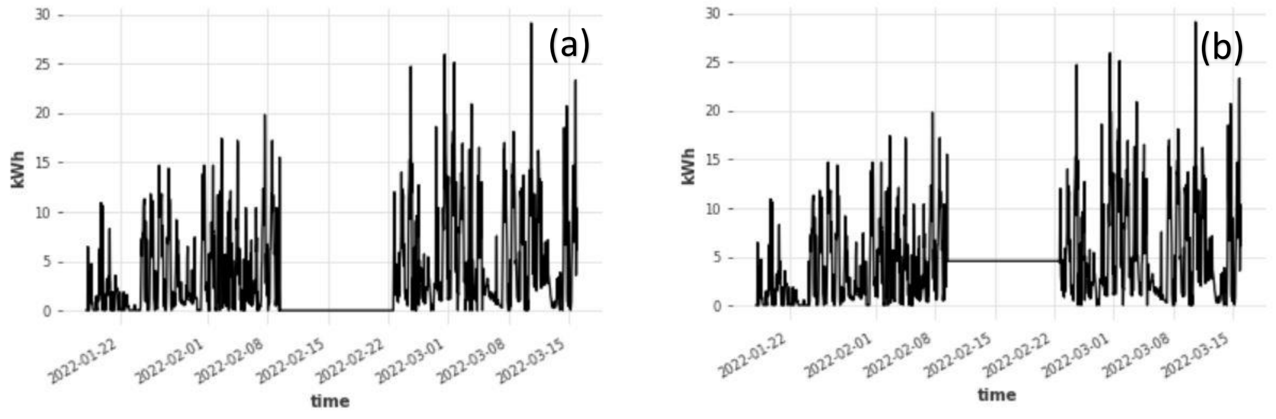


Figure 2 (a) Post processed Un-Imputed data (b) Post processed imputed data

Each model has two datasets: the original sub-daily (hourly) set and the daily set. The daily set is obtained by summing the original hourly set to a daily set. The forecasts are then obtained accordingly. The forecasts were based on the original hourly data and then compared to the forecasts obtained from the daily data.

time	watt_hour
18-01-22 0:00	26.1 Wh
18-01-22 1:00	30.4 Wh
18-01-22 2:00	27.5 Wh
18-01-22 3:00	27.5 Wh
18-01-22 4:00	6.46 kWh
18-01-22 5:00	27.5 Wh
18-01-22 6:00	3.95 kWh

$$\longrightarrow$$

time	watt_hour
18-01-22	10.549 Wh

Figure 3: Example showing transformation of the original hourly data into a daily data

7.3 Model fit metrics

For the models, the dataset is split into training and testing. Unlike in cross sectional data where partitioning is done randomly, in time series data partitioning is done sequentially where the data is trimmed into a training period from the start till period n and a testing period starting from $n + 1$ until the end.

Choosing the length of the testing or testing period is highly dependent on the forecasting horizon, data frequency, forecasting goal. The rule is to select a period that follows the forecast horizon so that the predictive performance of the model can be done [3]. In other words, if the forecast horizon is 10 hours and the data frequency is one hour then the testing period should at least have 10 entry points. In this study, the training and testing periods are set as 80% and 20% of the entire dataset, respectively.

In the parametric models, the coefficients of the regressions are estimated based on the training set. Once the coefficients are computed, they are used to predict the values from the testing. The predicted values are then compared to the observations (actual values) from the testing set.

While in the non-parametric model, there are two data subgroups, training and testing sets (like the training and testing sets in the parametric model, respectively). However, the training set in the non-parametric model is further split into a testing set as well. This testing process gives information that helps with the tuning of the model's hyperparameters and configurations accordingly. The testing set is a set of data not to be confused with the training set, that is used to validate our model performance during training.

To assess the predictive accuracy of a time series model and to avoid overfitting, the performance metrics of the testing period are calculated i.e., the predicted value generated by the model is compared to the actual value from the testing period. Also, a visualization of the predicted values using plots can prove useful [3]. The performance metrics used in this study to assess the performance of the models are the Pearson Correlation Coefficient (PCC), Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R squared (R^2).

$$PCC = \frac{\Sigma(y_i - \overline{y_i})(\widehat{y_i} - \bar{\hat{y}}_i)}{\Sigma(y_i - \overline{y_i})^2(\widehat{y_i} - \bar{\hat{y}}_i)^2} \qquad \text{Eq. 1}$$

where $\hat{y}_i$ is the predicted data, y_i is the actual data, $\bar{y}_i$ is the actual data's arithmetic mean (average), $\bar{\hat{y}}_i$ is the predicted data's arithmetic mean, and n is the number of data. PCC has a value between −1 to 1. An absolute value of 1 means that a linear equation describes the relationship between the two datasets perfectly. The sign of the coefficient is determined by the regression slope. If the regression slope is equal to +1 then when one dataset increases the other dataset will increase, vice versa when the slope is equal to -1. Whereas a PCC value of 0 means that there exists no linear relation between the two datasets [4]. The PCC is used in this study to assess how well the model predicted values correlate with the observed ones (in testing sets). A high correlation is a sign of good model prediction abilities.

RMSE measures the accuracy of the model by comparing forecasting errors of different models on the same dataset [5].

$$\mathrm{RMSE} = \sqrt{\frac{\Sigma(\hat{y}_i - y_i)^2}{n}} \qquad \text{Eq. 2}$$

R^2 is also known as the coefficient of determination. It measures how well the prediction of the model fits the observed data based on the proportion of total variation of outcomes explained by the model [6]. For example, if the R2 value is 60% then the model was able to fit 60% of the data. So, a higher value is desirable.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \qquad \text{Eq. 3}$$

where,

$$SS_{res} = \sum (\hat{y}_i - y_i)^2 \qquad \text{Eq. 4}$$

is the residual sum of squares and

$$SS_{total} = \sum (\overline{y}_i - y_i)^2 \qquad \text{Eq. 5}$$

is the total sum of squares (proportional to the variance of the data), $\overline{y}_i$ is the arithmetic mean (average) of the observed data (y_i).

7.4 Overfitting

Overfitting occurs when a model fits exactly against its training data resulting in poor performance on unseen data and failure to generalize predict or forecast on new data [7]. The reasons of overfitting are too complex but can be categorized into noise learning, hypothesis complexity, and multiple comparison procedures [8]. Overfitting will lead to the model memorizing the training data rather than generalizing from a trend i.e., learning, meaning that the model will fail to predict accurately when presented with new data. The problem of overfitting can be resolved in several ways such as early stopping, regularization, network reduction, or expansion of the training data in machine learning [8]. Overfitting can be detected by partitioning the data and checking if the model has low error rates and a high variance [9]. In this study, a cross validation technique is used to check for overfitting. The technique chosen is the sliding window technique where previous time steps are used to predict the future time.

In the parametric models, increasing the range of the testing set can help identify overfitting and reduce its risk by splitting the data into training and testing set, building the model on the training set, and evaluating its performance on the testing set to see its ability for generalization. Doing so has many advantages such as including the testing period that is the most recent and closest to the forecasting period, including the training period thus providing more data, and possibly leading to better accuracy [3]. Whereas in the non-parametric model, in addition to splitting the data, tuning of the main hyper-parameters to get an optimal result in terms of R^2, RMSE, and PCC on the testing data were used to avoid the issue of overfitting. While in the non-parametric model, the risk of overfitting can be reduced not only by splitting the data but also by tuning the hyperparameters. However, over-tuning of the hyperparameters can lead to overfitting so the tuning was done using a limited grid search technique. Some of the hyperparameters' values were considered from previous studies to reduce the computational cost. Details are provided in section 7.5.2.1.1.

7.5 Models

7.5.1 Parametric models

7.5.1.1 Linear Regression models

Linear regression analysis forecasts values of a certain variable, using values of other variables. The variable that is to be forecasted is referred to as the dependent or response variable. The variables that are used to calculate the response variable are called independent variables or predictors [14] [15]. First, the coefficients of the linear equation are estimated then a straight line is fitted that minimizes the differences between predicted and actual output values. The best fit straight line can be found using many methods such as the least square method. Then the value of the response can be estimated from the value of the predictor [14]. The estimation is done, using two renown methods, one called ordinary least squats and the other maximum likelihood.

Simple linear regression (SLR) model and multiple linear regression (MLR) models are types of linear regression models. The MLR is characterized as having one response and multiple predictors whereas the SLR is a special case of the MLR having one response and only one nontrivial predictor [15]. Other linear regression techniques are available and the difference between them, is the assumptions regarding the data, the relationship between the dependent variables, and the error term.

7.5.1.1.1 Simple linear regression (SLR) model

The general form for a linear regression is

$$Y_{\mathrm{i}} = X_{\mathrm{i1}}\beta_1 + X_{\mathrm{i2}}\beta_2 + \cdots + X_{\mathrm{ip}}\beta_p + \varepsilon_{\mathrm{i}} \qquad \text{Eq. 6}$$

Where it is assumed that each observation in a sample $(y_i, x_{i1}, x_{i2}, \ldots, x_{ip}), i = 1, \ldots, n$ is generated by an underlying process described by Eq. 6, where Y_i is the dependent or explained variable and $X_{\mathrm{i1}}, \ldots, X_{ip}$ are the independent or explanatory variables.

The term ε_i is a white noise process, otherwise known as the error term, it arises primarily because we cannot capture all influence on the dependent variable in a model. The dependent variable is energy consumption, and the independent variables are as explained in each regression model section. The author acknowledges that the validity of the model depends crucially on the assumption of the stochastic process that has led to the observations of the data at hand. Assumptions such as linearity, full rank between the independent variables, exogeneity between the independent variables and the error term. The latter shall also respect the homoscedasticity, no-autocorrelation, and normality criteria.

The simple linear regression (SLR) model can be written as:

$$Y_i = \alpha + \beta_1 X_i + e_i$$

where α and β_1 are unknown regression coefficients to be estimated, e_i is the i_{th} error, Y_i is the response variable, and X_i is the predictor variable [15].

In this case, X_i is the $t-1$ observation of Y_t (also referred to as Y_i in the above equation). The equation can thus be written as follows:

$$Y_t = \alpha + \beta_1 Y_{t-1} + e_t \qquad \text{Eq. 7}$$

7.5.1.1.2 Multiple linear regression (MLR) model – three variables

$$Y_i = \alpha + \beta_1 X_{i,1} + \beta_2 X_{i,2} + \beta_3 X_{i,3} + e_i \qquad \text{Eq. 8}$$

Like the direct linear relationship between Y_i and $X_{i,1}$, and due to the conceptually and empirically proven relationship between energy consumption and weather, the author believes that adding a weather-related independent variable, $X_{i,2}$, will lead to an improvement in the fit. Thus, the temperature variable has been added, as well as, another binary variable, $X_{i,3}$ to distinguish data, that is observed in the weekends.

The MLR model with three variables has the consumption data, temperature, and isWeekend as predictors because the metrics between the MLR with temperature and isWeekend and the MLR with temperature and AMPM as predictors do not differ much from each other.

7.5.1.1.3 Multiple linear regression (MLR) model – four variables

The difference between the model in 7.5.1.1.2 and in this one is that the model of this section has four predictors. The added predictor here is the AMPM.

7.5.1.2 ARIMA Model

ARIMA or Auto Regressive Integrated Moving Average model assumes that the predicted or forecasted value of a variable is a linear function of several past observations (AR) and random errors (MA) [10]

Expanding equation 9, one can write the following general form of ARMA/ ARIMA models:

$$Y_i = \alpha + \sum_{i=1}^{p} \beta_i * X_{1t-i} + \sum_{i=1}^{q} \theta_i * \varepsilon_{\mathrm{t-i}}$$

Or simply use this form

$$Y_t = \alpha + \sum_{i=1}^{p} \beta_i * Y_{t-i} + \sum_{i=1}^{q} \theta_i * \varepsilon_{\mathrm{t-i}} \qquad \text{Eq. 9}$$

where,

Y_t and ϵ_t are the actual value and random error at time t, respectively

β and θ are model parameters

p and q are known as the order of the model.

Random errors, ϵ_t , are assumed to be independently and identically distributed with a mean of zero and a constant variance of σ^2 [10].

A crucial step in creating a reliable ARIMA model is to ensure that the data is stationary [11]. Stationarity in a time series means that statistical characteristics such as the mean and the autocorrelation structure are constant over time. Otherwise, if the time series displays a trend or seasonality, differencing is applied to the data to remove the trend and seasonality and stabilize the variance before an ARIMA model can be fitted [11]. The Dickey-Fuller test is often used to test for stationarity or tend stationarity in a time series [12]. In the Dickey-Fuller test, if the time series has a *p-value* lower than a certain threshold then it is stationary [12].

Determining the order (p,q) of the model is an essential task because it has a significant impact on the forecasted/predicted values [10] [11]. Box and Jenkins developed a methodology to find the order of the model [11]. In short, Box-Jenkins advocate the use of the autocorrelation function (ACF) and the partial autocorrelation function (PACF) of the sample data as the basic tools to identify the order of the ARIMA model [11]. It is common practice to enumerate the set of all possible candidate models (ARIMA (p, q)) by considering significant lags based on the auto-correlation and partial auto-correlation functions (ACF and PACF). A grid search technique is then applied to come up with p and q values that minimizes the Akaike Information Criterion (AIC).

$$AIC = 2(k + p + q) - 2\log(L) \qquad \text{Eq. 10}$$

where L is the likelihood of the data and k is the intercept of the ARIMA model.

The best model has the lowest AIC value. The grid search was developed in RStudio and used to determine the lowest AIC value.

7.5.1.3 ETS Model

The Error, Trend, and Seasonality (ETS) model is one of many methods of the exponential smoothing technique. The error, trend, and seasonality in this model can be either additive (A), multiplicative (M), or none (N) [13]. The ETS model has a built-in function in RStudio that can also automatically determine the best fit for error, trend, or seasonality. In this study, the error and seasonality are selected to be additive whereas the trend is none.

	Seasonality		
Trend	None	Additive	Multiplicative
None	ZNN	ZNA	ZNM
Additive	ZAN	ZAA	ZAM
Multiplicative	ZMN	ZMA	ZMM
Additive damped	ZAdN	ZAdA	ZAdM
Multiplicative damped	ZMdN	ZMdA	ZMdM

Table 3: Possible exponential smoothing models in R using the ETS function where Z can be either A or M [3].

Some of the combinations stated in Table 3 are restricted because numerical difficulties will result when applied to some time series. However, the restriction can be overwritten by turning the restriction option off in the ETS function `ets (..., restrict = TRUE,...).` Another option in the ETS function is the automated option for model selection. It is activated by `ets (..., model ="ZZZ",...).` It selects the best model among the several combinations from Table 3 except the restrictive and multiplicative models (unless overwritten) [3].

The best model is chosen based on the Akaike's Information Criterion (AIC)

$$AIC = 2z - \log(L) \qquad \text{Eq. 11}$$

where z is the number of smoothing parameters.

Models with few smoothing parameters can help avoid overfitting a model and thus lead to better predictions or forecasts in the testing period or the future, respectively [3].

7.5.2 Non-parametric models

7.5.2.1 LSTM Model

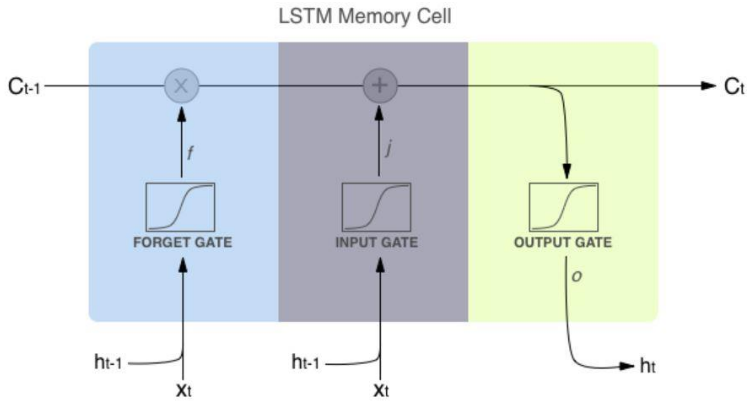


Figure 4: LSTM cell architecture [14]

Long Short Term Memory (LSTM) is one of many Recurrent Neural Networks (RNN) models that overpasses the issue of vanishing gradient problem and short term memory problem with gates and maintains information over long periods of time [16]. LSTM units have three different gates: input gate, output gate, and forget gate [9].

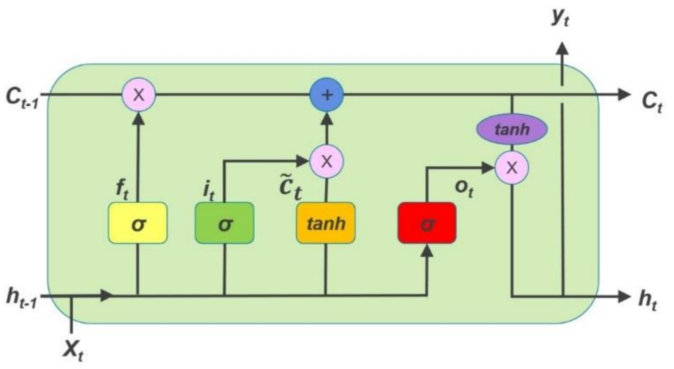


Figure 5: (a) LSTM memory cell (b) LSTM block at any timestamp (t) [14]

The mathematical equations associated with the LSTM gates at time t are as follows:

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i) \qquad \text{Eq. 12}$$

$$f_t = \sigma\left(w_f[h_{t-1}, x_t] + b_f\right) \quad \text{Eq. 13}$$

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \quad \text{Eq. 14}$$

$$\widetilde{c_t} = \tanh\,(w_c[h_{t-1}, x_t] + b_c \quad \text{Eq. 15}$$

$$c_t = f_t * c_{t-1} + i_t * \widetilde{c_t} \quad \text{Eq. 16}$$

$$h_t = o_t * \tanh\,(c_t) \quad \text{Eq. 17}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \text{Eq. 18}$$

where i_t is the input gate, f_t is the forget gate, o_t is the output gate, σ is the sigmoid function, w_x is the weight for the respective gate neurons, h_{t-1} is the output of the previous LSTM block at time $t-1$, x_t is the input at time t, b_x are the biases for the respective gates, c_t is the cell state at time t, and $\widetilde{c_t}$ represents the candidate for cell state at time t.

The model used in this study is made of three layers: an input layer, one hidden layer, and an output layer. The model is of type `Sequential()` because the data is time dependent with LSTM as the first layer followed by two Dense layers each with an activation function.

7.5.2.1.1 Hyperparameter tuning

The hyperparameters are the inputs on which the neural networks model is built. Therefore, it is crucial for the model accuracy and performance to determine the best set of hyperparameters. This process is known as hyperparameter tuning. The parameters to be tuned are the number of layers, the activation function, EPOCHS, loss function, and the optimizer. Hyperparameter tuning can be done either manually or automatically.

In manual tuning, the model is run a first time and then values of the hyperparameters are tweaked manually until the model has an acceptable performance. In contrast, random search and grid search models are automated. In the random search method, random sets of hyperparameters are selected and tested. Whereas, in the grid search method, all possible combinations of hyperparameters are tested making it an exhaustive and computationally expensive way. In this study, an extensive grid search method is chosen.

7.5.2.1.1.1 Loss function

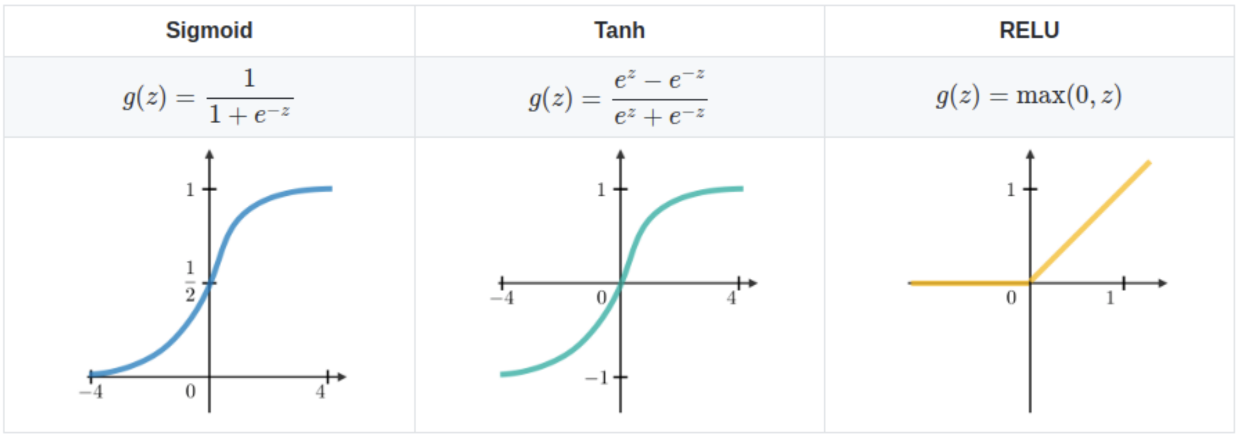
There are two types of loss functions: Classification and regression. The choice of loss function type depends on the desired outcome. The loss function type in this thesis is of the regression type since the data is of the sequential type. The loss function is necessary in the training of the model because it measures the deviation between the actual output and the output generated by the machine learning model. The result from the loss function is used to update the gradients that are in turn used to update the weights. There exist many loss functions each of which will have a different error for the same prediction. Therefore, the choice of the loss function will influence the performance of the model. In this model, the loss function is chosen to be the Mean Squared Error. The performance of the model is deemed good if the loss value is low enough meaning if the values predicted by the machine learning model are close to the actual values.

7.5.2.1.1.2 Optimizer and learning rate

The deep learning model must update the weights after each epoch while minimizing the loss function. The way the model does that is via an optimizer that changes the weights and learning rate of the model with respect to the gradient descent [17]. The learning rate determines how fast the model is trained [16]. Adam, derived from adaptive moment estimation, is a stochastic optimization method that needs first-order gradients with minimal memory requirement [18]. It computes "individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients" [18]. The optimizer used in this study is Adam not only because it has the advantages of two other stochastic optimization methods, AdaGrad and RMSProp [18] but also because the Adam optimizer is generally better, has less computational time, and requires fewer parameters for tuning [17].

7.5.2.1.1.3 Activation function

Activation functions, sometimes referred to as transfer functions, are used to calculate the weighted sum of input and biases and consequently determine whether a neuron is to be activated [19]. These functions can either be linear or non-linear. The choice of the proper activation function will lead to the best model performance [19]. In most case, the optimal activation function is generally determined by trials or tuning [20]. There are many activation functions available in deep machine learning such as the sigmoid, hyperbolic tan, and the rectified linear unit (ReLU).



Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z) = \max(0, z)$

Figure 6: Some of the activation functions available [25]

ReLU, used only in the hidden layers, is more efficient than other functions because it does not activate all the neurons simultaneously [21]. ReLU function is the most popular function and has a better performance than the other activation functions [21].

$$g(z) = \max\,(0, z) \qquad \text{Eq. 19}$$

where z is the input to a neuron.

7.5.2.1.1.4 EPOCH

The number of epochs is the number of times the model goes through the training data. Upon the completion of one epoch, the model would have seen each entry once. In other words, the dataset has been passed forward and backward through the network only once. In this study, the number of EPOCHs was determined manually by plotting different learning curves in which the model loss is plotted against the number of EPOCH. Since gradient descent is used for optimization then it is better to pass the entire dataset through a single network multiple times to update the weights and get a more accurate model [22]. However, there is no general rule to obtain the best epoch to find the optimal weights. Different datasets behave differently and therefore this parameter is data specific [22].

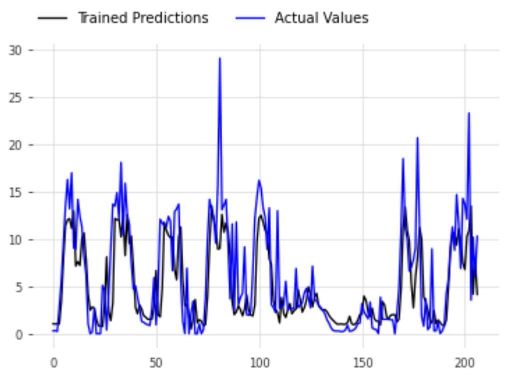
7.5.2.1.1.5 Batch Size

The batch size is the total number of training data used. For instance, when a dataset is divided into parts each part is referred to as a batch. An iteration, by contrast, is the number of batches required to use the entire dataset. Consequently, the number of batches is equal to number of iterations to complete one round of training using the entire set of data. For example, assume that we have 100 training points. If the dataset is divided into batches of 10 then 10 iterations would be needed to fulfill one round of training [22]. The batch size and the number of epochs determine how often the weights are updated and thus how quick the machine learning process is.

7.5.2.1.1.6 Number of neurons and layers

The number of layers and the number of neurons for each layer cannot be obtained from a certain formula. So, a trial and error approach will result in different performance curves upon which the best combination was chosen. The number of layers has been selected as 1 since results slightly increased resulted by adding an additional layer. The addition of layers might result as well in overfitting [23]. The number of neurons was determined using a manual and exhaustive grid search technique.

7.5.2.1.2 Hyperparameter tuning results

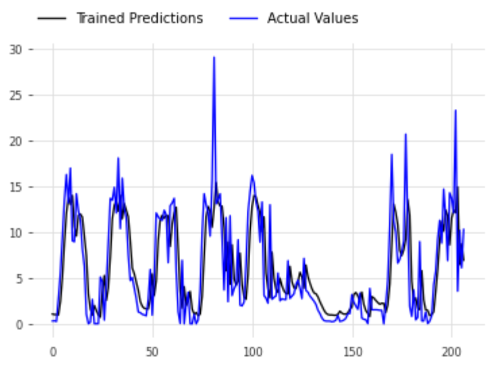


a) For learning_rate= 0.001, EPOCH= 30, win_size= 5, and batch_size= 5:

RMSE = 3.92

R^2 *= 0.51*

PCC = 0.73

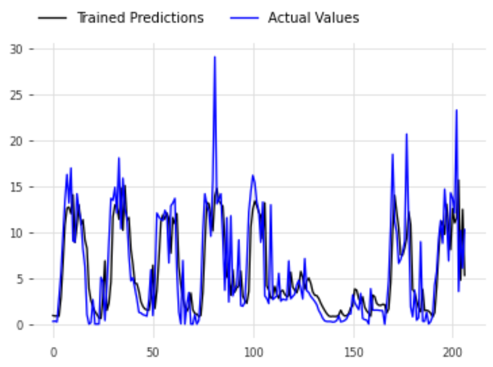


b) For learning_rate= 0.001, EPOCH= 150, win_size= 5, and batch_size= 5:

RMSE = 4.31

R^2 *= 0.40*

PCC = 0.69

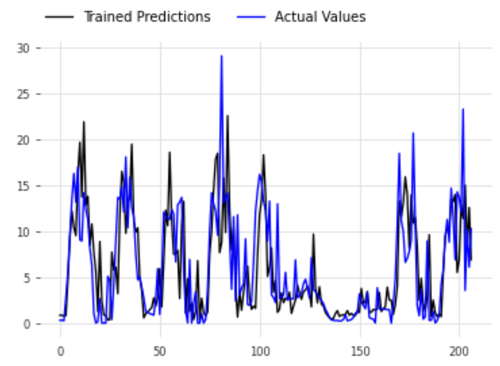


c) For learning_rate= 0.0001, EPOCH= 30, win_size= 5, and batch_size= 5:

RMSE = 3.72

R^2 *= 0.55*

PCC = 0.74



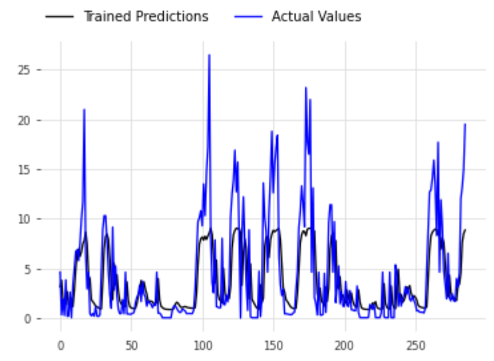
d) For learning_rate= 0.0001, EPOCH= 150, win_size= 5, and batch_size= 5:

RMSE = 3.70

R^2 *= 0.56*

PCC is 0.75

Figure 7: Comparison of LSTM models with different epochs and learning rates

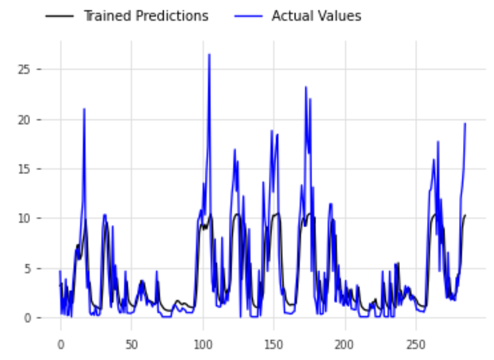


a) For LSTM neurons set at 8 and the Dense layer neurons at 2:

RMSE = 3.71

R^2 *= 0.509*

PCC = 0.76

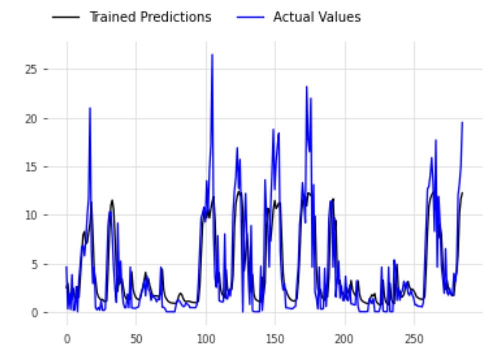


b) For LSTM neurons set at 8 and the Dense layer neurons at 8:

RMSE = 3.49

R^2 *= 0.565*

PCC = 0.771

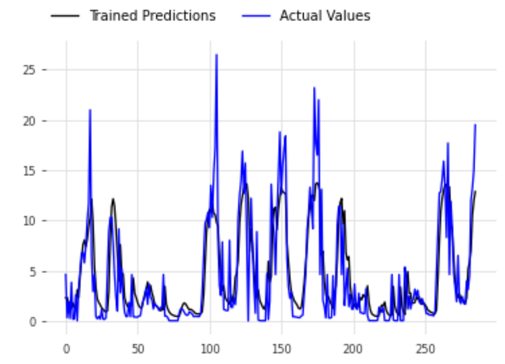


c) For LSTM neurons set at 16 and the Dense layer neurons at 16:

RMSE = 3.26

R^2 *= 0.6207*

PCC = 0.793

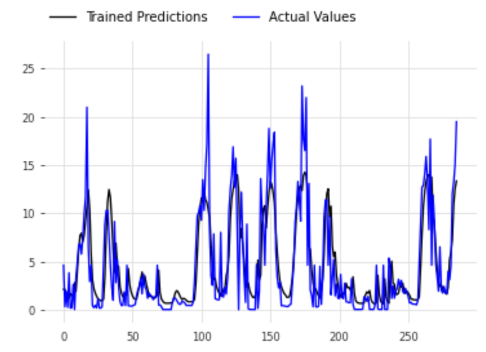


d) For LSTM neurons set at 32 and the Dense layer neurons at 32:

RMSE = 3.13

R^2 *= 0.651*

PCC = 0.808

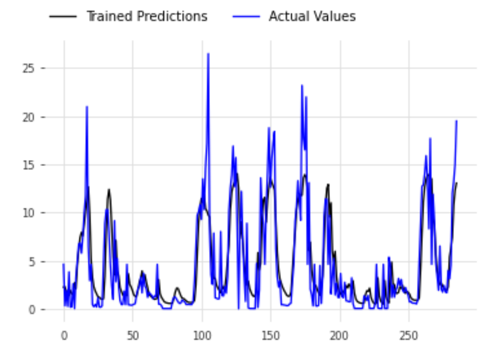


e) For LSTM neurons set at 32 and the Dense layer neurons at 64:

RMSE = 3.11

R^2 *= 0.655*

PCC = 0.81

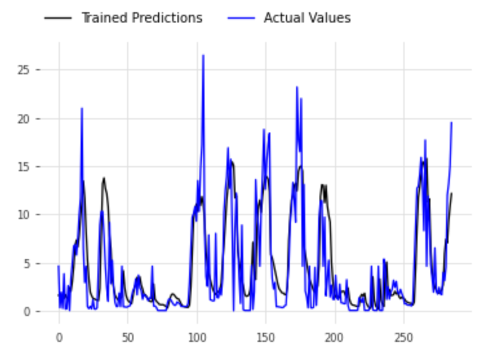


f) For LSTM neurons set at 32 and the Dense layer neurons at 128:

RMSE = 3.095

R^2 *= 0.659*

PCC = 0.813



g) For LSTM neurons set at 128 and the Dense layer neurons at 128:

RMSE = 3.38

R^2 = 0.593

PCC = 0.777

Figure 8: Models with different input neurons and Dense hidden layer neurons. Batch size = 15, Window size=20, Epoch=150, Learning rate=0.0001, drop out = 0.2

Among the machine learning models, the one with the relatively best metrics is selected. The model chosen and the metrics obtained can be summarized as follows:

Hyperparameter	**Value**
Batch size	15
Window size	20
Learning rate	0.0001
EPOCH	150
LSTM cell neurons	32
Number of hidden layers	1
Neurons per hidden layer	32
Activation function for hidden layer	ReLU
Activation function for output layer	Linear
Dropout coefficient	0.2
RMSE	3.12
$\boldsymbol{R^2}$	0.654
PCC	0.810

Table 4: Summary of best non-parametric model showing hyperparameters and metrics

8 Results

8.1.1 ARIMA Model

8.1.1.1 Hourly data

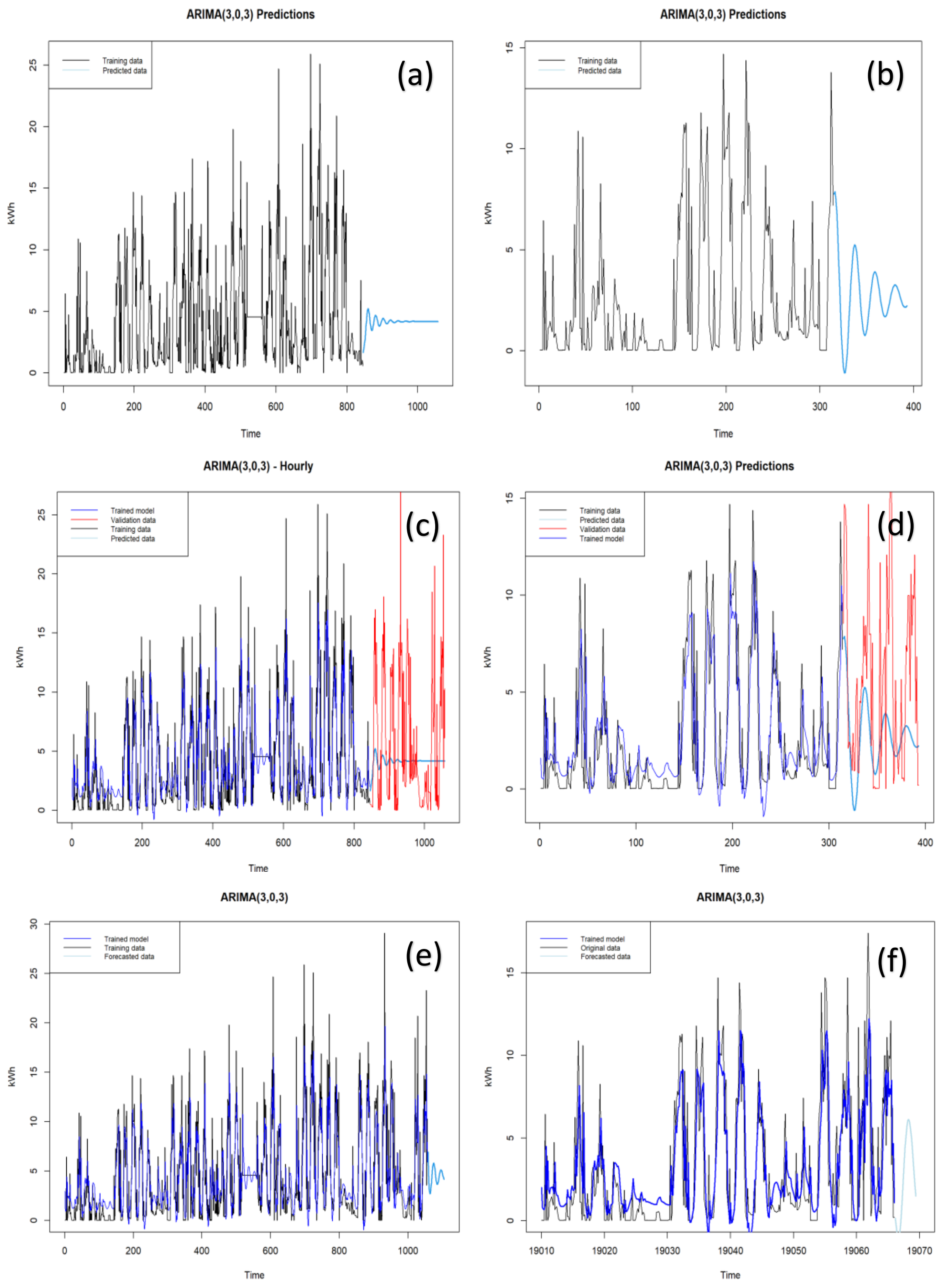


Figure 9: (a) Hourly data and predictions (b) Hourly data TS and predictions (c) validation of (a) (d) validation of (b) (e) forecast of (a) (f) forecast of (b). Here validation data means testing data

8.1.1.2 Daily data

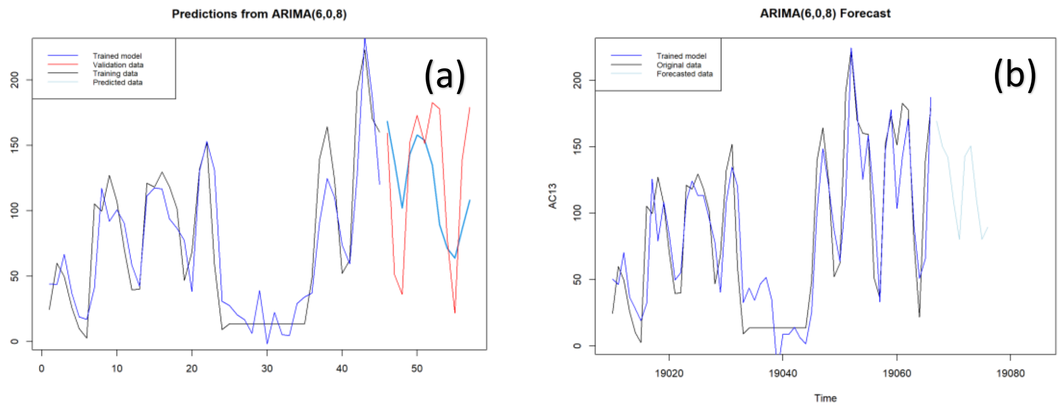
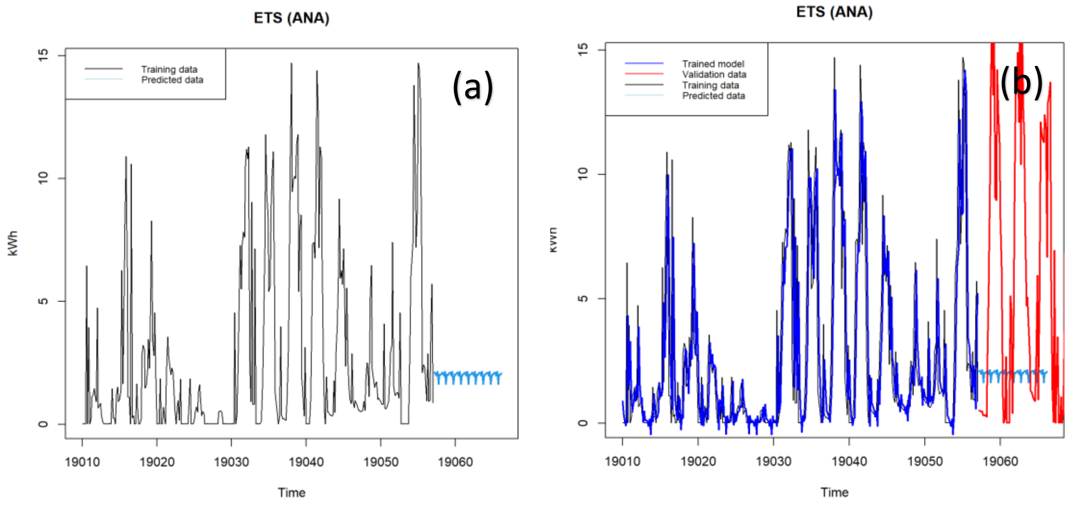


Figure 10: (a) Shows the training data in black, predicted values in light blue, trained model in blue, and validation(actual) values in red (b) Shows the forecasted data in light blue, trained model in blue, and original data in black. Here validation data means testing data

8.1.2 ETS Model

8.1.2.1 Hourly data



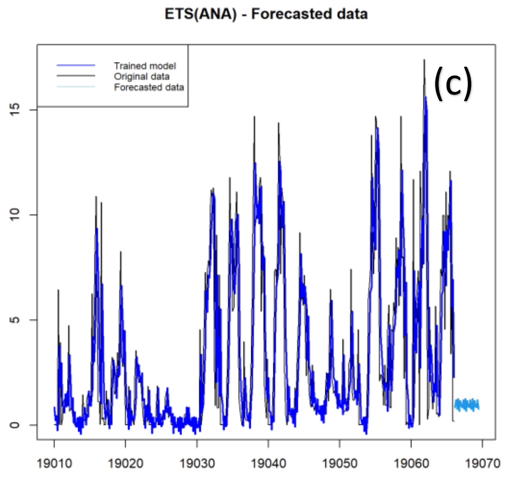


Figure 11: (a) Shows the training data in black and the predicted data in light blue (b) Shows the trained model in navy blue and the validation data in red over the original and predicted data (c) Shows the original data, trained model, and the forecasted values. Here validation data means testing data

8.1.2.2 Daily data

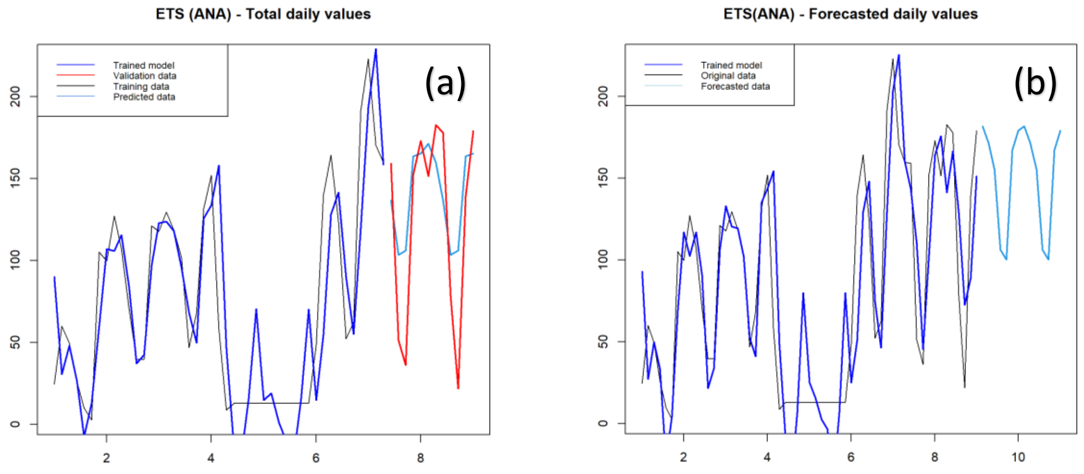
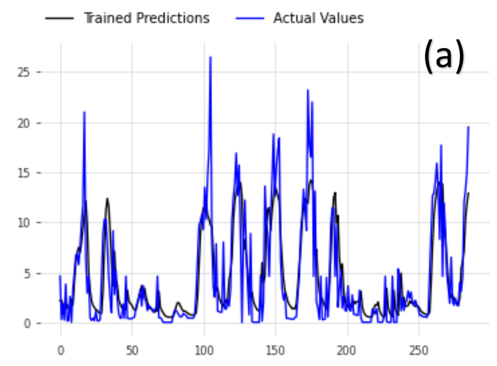


Figure 12: (a) Original training data in black, the trained model in navy blue, the validation data in red, and the predicted data in light blue (b) Original data, trained model, and forecasted data. Here validation data means testing data

8.1.3 LSTM

8.1.3.1 Hourly data



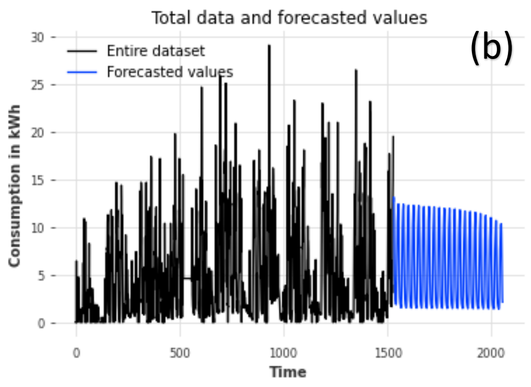
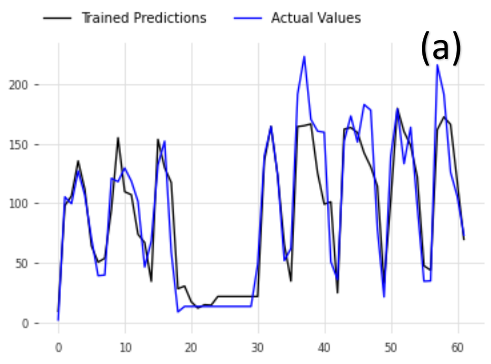


Figure 13: (a) Hourly model's predicted values in black versus model's actual values in blue (b) Hourly model forecasted values in blue

8.1.3.2 Daily data



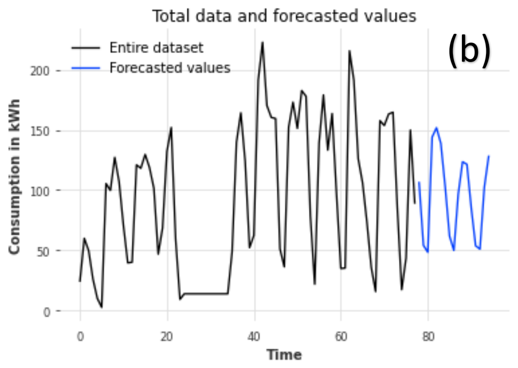
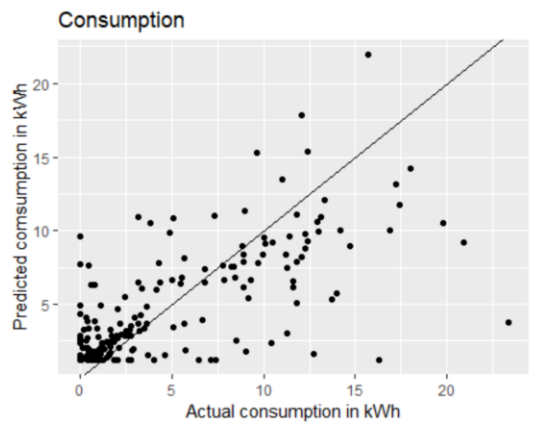


Figure 14: (a) Daily model's predicted values in black versus model's actual values in blue (b) Daily model forecasted values in blue

8.1.4 Regression models

8.1.4.1 Simple Linear Regression



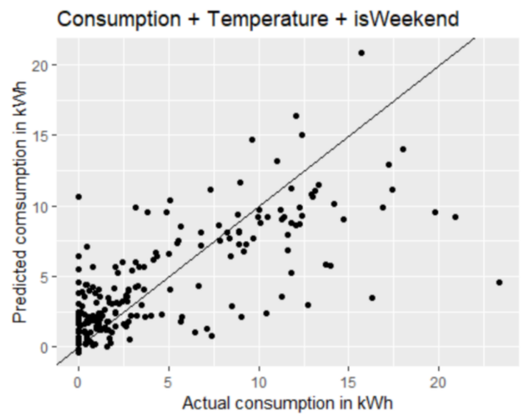
RMSE = 3.67

R^2 *= 0.495*

PCC = 0.496

Figure 15: Scatterplot for the SLR model showing predicted and actual values with the metrics shown on the side

8.1.4.2 Multiple linear regression (three variables)



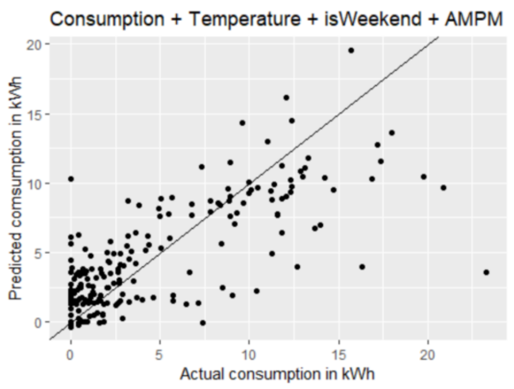
RMSE = 3.51

R^2 *= 0.537*

PCC = 0.537

Figure 16: Scatterplot for the MLR model with 3 variables showing predicted and actual values with the metrics shown on the side

8.1.4.3 Multiple linear regression (four variables)



RMSE = 3.384

R^2 = 0.57

PCC = 0.57

Figure 17: Scatterplot for the MLR model with 4 variables showing predicted and actual values with the metrics shown on the side

8.2 Summary

	ARIMA(3,0,3)-Hourly	ARIMA(6,0,8)-Daily	ETS – Hourly	ETS - Daily	LSTM – Hourly	LSTM - Daily
RMSE	4.807009	51.69176	3.960112	40.22078	3.095	43.814
PCC	0.2858419	0.2319191	0.0002081217	0.7388444	0.813	0.716
R^2	-0.09344764	0.2049215	-0.7021382	0.5184492	0.659	0.451

Table 5: Summary table of ARIMA, ETS, and LSTM models' performance metrics

	Simple Linear Regression	Multiple linear regression (two variables)	Multiple linear regression (three variables)
RMSE	3.665229	3.508423	3.383951
PCC	0.4955552	0.5373905	0.5696202
R^2	0.4948103	0.5371117	0.5693739

Table 6: Summary of linear regression models' performance metrics

9 Discussion

The models used in this thesis vary in terms of complexity, some are auto-driven and governed by specific assumptions, while others have some of the assumptions relaxed and can be manually tuned by the user. While the first type is easy to communicate and perform, the second adds complexity, but at the same time, can improve the forecasting and generalization capability of the models.

After making sure that the consumption variable is stationary, the simple linear regression is performed, and the coefficient of determination is computed. Additionally, the error terms have been tested for normality and autocorrelation.

Since including additional variables will improve the model, the weather and some control variables are added. It appears that the model slightly improves, and the coefficients for the added variables are statistically significant as well (with 95% confidence level).

All above mentioned regression models, respect the same set of rules and assumptions, mainly used on ordinary least square method. The errors are independent, non-autocorrelated and normal. However, when the error term is allowed to have a time series structure, it is possible to use other models, such as ARMA, SARIMA, or ETS, while adjusting the estimated regression coefficients and standard errors. This adds flexibility to the user with the potential to explore additional models.

The predictive power of the ARIMA and ETS models is minimal in the hourly data but gets significantly better when the daily data is used. Unfortunately, SARIMA model does not improve the results neither. The main difference between the ETS and ARIMA models is that the ETS deals with seasonality and trend in the data whereas ARIMA focuses on the autocorrelation in the data and does not detect neither trend nor seasonality. Also, it is worthy to mention that the data is relatively short giving an advantage for the ETS over the ARIMA [24]. Using a seasonal ARIMA model did not show improvements from the ARIMA model. When enough data is provided multiple seasonality might be present in the data and therefore another approach such as Seasonal-Trend decomposition using LOESS (STL) or Trigonometric seasonality, Box-Cox transformation, ARMA errors, Trend and Seasonal components (TBATS) must be selected [24].

Also, sub-daily data can sometimes have seasonal complexities and are better dealt with using the `msts` function [27]. When the data is not dealt with using any time series function such as `ts`, `xts`, or `msts` in the code, the resulting forecast looks like that of Figure 9(a), however, when such a function is used the forecast gets a better shape Figure 9(b).

ARIMA and ETS provide another approach to time series forecasting and are widely used approaches to time series forecasting. Conceptually, both models should give better fit, especially for data that contains trend/seasonality. This is the case for exponential smoothing models whereas the ARIMA models loosen the auto-correlation of errors assumption for regression models, as this adds flexibility.

To exploit all available options, machine learning techniques are also employed, and since we are constrained by the type of data, the LSTM model is used. The LSTM hourly model outperforms the other models. The parametric model with the performance closest to that of the LSTM is the MLR with four variables. One of the reasons the LSTM model outperforms the parametric models could be attributed to the relax-free environment in terms of loosening the assumptions, empirical tuning, and the black-box strategy where the machine learning methodology is responsible for finding the best relationship between inputs and outputs. Also, The “black-box” method has overcome some of the disadvantages in the parametric models. However, machine learning models require a large database to build a reliable model [23]. Therefore, the model can be retested when more data is available. Additionally, the results are case-sensitive to the data and its assumptions. Another reason the LSTM model outperforms the other models in predicting the consumption is mainly the optimization algorithm’s approach that finds the best result. The approach is known as iterative because the results are generated several times before the model with the lowest error is chosen [22].

As shown in Figure 7, increasing the epochs does not necessarily produce better results. As it can be concluded by comparing the metrics of Figure 7 (a) and (b) where the RMSE and the R^2 have decreased when the epoch increases while all other hyperparameters are held constant. Whereas, the same metrics increased, although slightly, in Figure 7 (c) and (d). So, there is a lack of evidence to conclude that training the dataset more than a certain number of times would improve the accuracy of the model.

In this study, it was evident that the more training data used the more accurate the model is. This finding is corroborated as well in the study [8]. However, acquiring more data such as weather data was difficult and can sometimes be costly. Also, data collection from Kube Energy’s website was not consistent because the sensors on site would not collect data properly (mention periods).

It is evident that the model performance metrics get better when there is more data while all other hyperparameter are held constant. This can also be seen by comparing the hourly and daily LSTM models. The daily model has fewer data and has lower performance metrics. Even though some of the models showed somewhat of an acceptable performance, the predictions are ought to be treated with care and are to be used for short term predictions.

10 Limitations

The main obstacle that was faced while writing the thesis was the limitation of data. Data is limited to one set of variables, namely the power consumption, which restricted the use of univariate parametric regression type methods, and non-parametric LSTM method. Additional variables, like the weather data used, as well as other data that could explain the consumption patterns, might produce better forecasting results. Additionally, the univariate data was sometimes unavailable as well, creating discontinuity in the sample. Given that the data provided started from the 18th of January 2022, any discontinuity could have significant effects on the study since all the models depend on data to perform well. Imputing methods were employed; however, the authors acknowledge that other types can be used, and produce different results. The use of continuous data set will avoid such issues in the future. The author believes that the machine learning model results can also be improved further, and overfitting avoided, if more data is available. In fact, more data, means that the training and testing data available will increase in range, thus the model will have better ability to learn, and to generalize. Additionally, rolling windows and cross validation techniques can be used more extensively to test for overfitting.

11 Conclusion

With the rapid development of advanced deep learning algorithms, machine learning models are becoming more popular. This thesis compares the prediction power of several parametric models, including ARIMA, ETS and linear regression models, and a non-parametric model, LSTM. These models were tuned and then used on a set of power consumption data to have valid consumption forecasting capability at hand. The results clearly showed an improvement of the prediction power when the LSTM model was used. This thesis advocates the use of deep learning models to forecast power consumption. The next logical step would be to compare the performance of the LSTM model with other emerging intelligent models such as BERT, Reformers, or T5. Also, when enough data is available the process of imputation can be improved, the results validated/tested, and the model generalizing capabilities enhanced. The author believes that these methods will help Kube, establish its PMS model. The consumption forecasting methods introduced in this document, can be added to the power generation model that is being built, and together can indeed improve the efficiency, and thus the profitability of their project will be enhanced.

12 References

[1] T. Talakoobi, "Solar Power Forecast Using Artificial Neural Network Techniques," 2020. [Online]. Available: https://webthesis.biblio.polito.it/15873/.

[2] A. Buteikis, "Practical Econometrics and Data Science," 2018. [Online]. Available: http://web.vu.lt/mif/a.buteikis/wp-content/uploads/PE_Book/index.html.

[3] G. Shmueli and K. C. Lichtendahl Jr, Practical Time Series Forecasting with R: A Hands-On Guide, 2016.

[4] P. S. E. C. o. Science, "2.6 - (Pearson) Correlation Coefficient r," [Online]. Available: https://online.stat.psu.edu/stat462/node/96/.

[5] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting,* vol. 22, no. 4, pp. 679-688, 2006.

[6] N. R. Draper and H. Smith, Applied Regression Analysis, Wiley Interscience, 1998.

[7] IBM, "Overfitting," [Online]. Available: https://www.ibm.com/cloud/learn/overfitting.

[8] X. Ying, "An Overview of Overfitting and its Solutions by 2019 J. Phys.: Conf. Ser. 1168 022022," *Journal of Physics: Conference Series,* vol. 1168, 2019.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation,* vol. 9, p. 1735–1780, 1997.

[10] IBM, "Linear Regression," [Online]. Available: https://www.ibm.com/topics/linear-regression.

[11] D. J. Olive, Linear Regression, Springer, 2017.

[12] G. Zhang, "Time series forecasting using a hybrid ARIMA and neural network model," 2001.

[13] G. J. G.E.P. Box, "Time Series Analysis, Forecasting and Control," 1970.

[14] D. A. Dickey and W. A. Fuller, "Distribution of the Estimators for Autoregressive Time Series with a Unit Root," *Journal of the American Statistical Association,* 1979.

[15] R. J. Hyndman, A. B. Koehler, R. D. Snyder and S. Grose, "A state space framework for automatic forecasting using exponential smoothing methods," *International Journal of Forecasting,* pp. 439-454, 2002.

[16] R. S. a. H. N. M. Sundermeyer, "LSTM neural networks for language modeling," in *Thirteenth annual conference of the international speech communication association*, 2012.

[17] A. Gupta, "A Comprehensive Guide on Deep Learning Optimizers," Analytics Vidhya, 7 October 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/#:~:text=An%20optimizer%20is%20a%20function,loss%20and%20improve%20the%20accuracy.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations,* 2014.

[19] C. E. Nwankpa, W. Ijomah, A. Gachagan and S. Marshall, "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning," 2020.

[20] Ö. F. Ertugrul, "A novel type of activation function in artificial neural networks: Trained activation function," *Neural Networks,* vol. 99, pp. 148-157, 2018.

[21] S. Sharma, S. Sharma and A. Athaiya, "ACTIVATION FUNCTIONS IN NEURAL NETWORKS," *International Journal of Engineering Applied Sciences and Technology,* vol. 4, no. 12, pp. 310-316, 2020.

[22] S. Siami-Namini, N. Tavakoli and A. S. Namin, "A Comparison of ARIMA and LSTM in Forecasting Time Series," in *17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018.

[23] M. Uzair and N. Jamil, "Effects of Hidden Layers on the Efficiency of Neural networks," in *IEEE 23rd International Multitopic Conference (INMIC)*, 2020.

[24] R. J. Hyndman and G. Athanasopoulos, Forecasting: Principles and Practice, Otexts, 2018.

[25] L. Gigoni, A. Betti, E. Crisostomi, A. Franco, M. Tucci, F. Bizzarri and D. Mucci, "Day-Ahead Hourly Forecasting of Power Generation From Photovoltaic Plants," *EEE Transactions on Sustainable Energy,* vol. 9, no. 2, pp. 831-842, 2018.

[26] B. Kanani, "Activation Functions in Neural Network," 2019. [Online]. Available: https://studymachinelearning.com/activation-functions-in-neural-network/.

Appendix A: ARIMA(3,0,3) - Hourly data

```
# Clear console and envirronment
cat("\014")
rm(list=ls())

# Set Working directory
setwd("C:/Users/ali_a/Desktop/Master Thesis/Code")

# Libraries required
library(dplyr)
library(tseries)
library(lubridate)
library(ggplot2)
library(forecast)

# Code start
power_use <- read.csv("Energy consumption per hour-data-2022-03-16
17_50_56_Edited.csv")
time_vector <- dmy_hm(power_use$ï..time)
ac13 = power_use$watt_hour

# Fill missing data using "Impute"

ac13[ac13==0]=NA
# Replace NA values with the median value computed in the previous step
  power_use_replaced <- power_use %>%
    mutate(ac13_mean_replaced  = ifelse(is.na(ac13), mean(ac13,na.rm=TRUE),
ac13))
  sum(is.na(power_use_replaced$watt_hour))
  sum(is.na(power_use_replaced$ac13_mean_replaced))
  head(power_use_replaced)

# de-trending and de-seasonalizing
AC13 <- power_use_replaced$ac13_mean_replaced
plot(time_vector,AC13,type= "l")
noSnoT = AC13
noSnoT
plot(time_vector[1:length(time_vector)],noSnoT,type = "l", xlab = "Time",
ylab = "kWh")

# Dickey-Fuller Test
Stationarity = adf.test(noSnoT, alternative = "stationary")
if (Stationarity$p.value <= 0.01) {
  print("AC13 is a stationary data set")
}

# ACF and PACF(Partial Auto Correlation Factor)
acf(AC13)
pacf(AC13)
acf(noSnoT)
pacf(noSnoT)

# Splitting data into training and validation period
```

```
train_per <- 0.8
limit <- floor(train_per * length(noSnoT))
train_noSnoT <- noSnoT[1:limit]
train_noSnoT
validation_noSnoT <- noSnoT[(limit+1):length(noSnoT)]
validation_noSnoT

#Building the model
arima_noSnoT = arima(train_noSnoT, order=c(3,0,3))
arima_noSnoT
checkresiduals(arima_noSnoT)
x_range = seq(1,length(noSnoT))
predict_noSnoT = forecast(arima_noSnoT,h=length(validation_noSnoT), level=0)
predict_noSnoT

#plotting the model
plot(predict_noSnoT,main="ARIMA(3,0,3) - Hourly",xlab="Time",ylab="kWh")
lines(x_range[(limit+1):length(noSnoT)],validation_noSnoT,col="red")
lines(predict_noSnoT$fitted,col="blue")
legend("topleft",c("Trained model","Validation data","Training
data","Predicted data"),
       lty=c(1,1,1,1),col=c("blue","red","black","light blue"),y.intersp =
0.35,
       cex = 0.75,text.width = 200)
#calculating metrics
rmse = sqrt(mean((validation_noSnoT - predict_noSnoT$mean)^2))
rmse

rsq1 <- function (x, y) cor(x, y) ^ 2
pcc = rsq1(predict_noSnoT$mean,validation_noSnoT)
pcc

rss <- sum((predict_noSnoT$mean - validation_noSnoT)^2) ## residual sum of
squares
tss <- sum( ( validation_noSnoT - mean(validation_noSnoT) ) ^ 2)  ## total
sum of squares
rsq <- 1 - rss/tss
rsq

# Forecasting
predict_future = 50
arima_all = arima(noSnoT,order=c(3,0,3))
forecast_noSnoT = forecast(arima_all,h=predict_future, level=c(0))
forecast_noSnoT
plot(forecast_noSnoT,main="ARIMA(3,0,3)",xlab="Time",ylab="kWh",xlim =
c(0,1150))

lines(forecast_noSnoT$fitted,col="blue")

legend("topleft",c("Trained model","Training data","Forecasted data"),
       lty=c(1,1,1,1),col=c("blue","black","light blue"),y.intersp = 0.35,
       cex = 0.75,text.width = 200)
```

Appendix B: ARIMA(3,0,3) with ts – Hourly data

```r
# Clear console and environment
cat("\014")
rm(list=ls())

# Set Working directory
setwd("C:/Users/ali_a/Desktop/Master Thesis/Code")

# Libraries required
library(dplyr)
library(tseries)
library(lubridate)
library(ggplot2)
library(forecast)
library(xts)

# Code start
power_use <- read.csv("Energy consumption per hour-data-2022-03-16
17_50_56_Edited.csv")
dates <- power_use$ï..time
ac13 = as.numeric(power_use$watt_hour)
plot(ac13,type='l')

ac13[ac13==0] = NA
# Fill missing data using "Impute"
avg_ac13 = mean(na.omit(ac13))

power_use_replaced <- power_use %>%
   mutate(ac13_mean_replaced  = ifelse(is.na(ac13), avg_ac13, ac13))
 sum(is.na(power_use_replaced$watt_hour))
 sum(is.na(power_use_replaced$ac13_mean_replaced))
 head(power_use_replaced)


power_use_replaced %>%
   mutate(ï..time=as.POSIXct(dates,format="%d-%m-%y %H:%M",tz=""))

Ac13=power_use_replaced$ac13_mean_replaced
dates13=power_use_replaced$ï..time
AC13 = ts(Ac13, frequency = 7, start = as.Date(dates13[1],format="%d-%m-%y"),
          end = as.Date(dates13[length(Ac13)],format="%d-%m-%y"))

noSnoT = AC13
noSnoT


# Dickey-Fuller Test
Stationarity = adf.test(noSnoT, alternative = "s")
if (Stationarity$p.value <= 0.01)
{
  print("AC13 is a stationary data set")
}

# ACF and PACF(Partial Auto Correlation Factor)
```

```
acf(AC13)
pacf(AC13)

acf(noSnoT)
pacf(noSnoT)

# Splitting data into training and validation period
train_per <- 0.8
limit <- floor(train_per * length(noSnoT))
train_noSnoT <- noSnoT[1:limit]
train_noSnoT
validation_noSnoT <- noSnoT[(limit+1):length(noSnoT)]
validation_noSnoT


# Determining the values of p, d, and q based on lowest AIC:
pvalue_manual = c(0:2)
qvalue_manual = c(0:1)
manual_order = 0
min_aic = 1000000
 for(p in 1:length(pvalue_manual)){
  for(q in 1:length(qvalue_manual)){
    man_arima = arima(train_noSnoT, order=
c(pvalue_manual[p],0,qvalue_manual[q]))
    print(c(pvalue_manual[p],0,qvalue_manual[q]))
    print(":")
    print(man_arima$aic)
    if(man_arima$aic < min_aic) {
      min_aic = man_arima$aic
      manual_order = c(pvalue_manual[p],0,qvalue_manual[q])
    }
  }
}
print(paste("AC13 Lowest AIC: ",min_aic))
print(paste("AC13 Corresponding ARIMA: ",toString(manual_order)))

x_range = seq(1,length(noSnoT))


p = 3#manual_order[1] #1 #3
d = 0#manual_order[2]
q = 3#manual_order[3] #5 #3

# Using of SARIMA
P = 0 #2
D = 0
Q = 0 #2

#Building the model
arima_noSnoT = arima(train_noSnoT, order=c(p,d,q),seasonal = c(P,D,Q))
arima_noSnoT
checkresiduals(arima_noSnoT)
predict_noSnoT = forecast(arima_noSnoT,h=length(validation_noSnoT),
level=c(0))
predict_noSnoT

# Plotting test data vs predictions
```

```
plot(predict_noSnoT, main="ARIMA(3,0,3) Predictions",xlab =
"Time",ylab="kWh")
lines(predict_noSnoT$fitted,col="blue")
lines(x_range[(limit+1):length(noSnoT)],validation_noSnoT,col="red")
legend("topleft",c("Training data","Predicted data","Validation
data","Trained model"),
       lty=c(1,1),col=c("black","light blue","red","blue"),y.intersp = 0.35,
       cex = 0.75,text.width = 60)

#Calculating the metrics
rmse = sqrt(sum((validation_noSnoT -
predict_noSnoT$mean)^2)/length(validation_noSnoT))

rsq1 <- function (x, y) cor(x, y) ^ 2
pcc = rsq1(predict_noSnoT$mean,validation_noSnoT)
pcc

rss <- sum((predict_noSnoT$mean - validation_noSnoT)^2) ## residual sum of
squares
tss <- sum( ( validation_noSnoT - mean(validation_noSnoT) ) ^ 2)  ## total
sum of squares
rsq <- 1 - rss/tss
rsq

# Residuals
x = seq(1,length(validation_noSnoT))
max_y = max(predict_noSnoT$upper,validation_noSnoT)
min_y = min(predict_noSnoT$lower,validation_noSnoT)

plot(x, as.numeric(predict_noSnoT$mean),type="l",col="blue",
     ylim=c(min_y,max_y + 0.05))
lines(x,as.numeric(predict_noSnoT$upper), col="green")
lines(x,as.numeric(predict_noSnoT$lower), col="red")
points(x,validation_noSnoT,col="black", type="l")
legend(1,max_y + 0.05,legend=c("upper","mean","lower","actual"),
       col=c("green","blue","red","black"),
       pch=c("","","","*"),lty=c(1,1,1,0), ncol=4,cex = 0.5)

# Forecasting using all data
predict_num = 24 #forecasting horizon
arima_all_noSnoT = arima(noSnoT, order=c(p,d,q),seasonal = c(P,D,Q))
lt_noSnoT = predict(arima_all_noSnoT, n.ahead = predict_num)
ltFore_noSnoT = forecast(arima_all_noSnoT, h=predict_num,level=0)

print(lt_noSnoT$pred)
x = seq(1,predict_num)
max_y = max(lt_noSnoT$pred)
min_y = min(lt_noSnoT$pred)

plot(x, as.numeric(lt_noSnoT$pred),type="l",col="blue",
     ylim=c(min_y,max_y), main="Predictions")

legend(15,max_y,legend=c("noSnoT"),
       col=c("blue"),
       pch=c(""),lty=c(1), ncol=1)
```

```r
x = seq(1,predict_num)
max_y = max(lt_noSnoT$se)
min_y = min(lt_noSnoT$se)

plot(x, as.numeric(lt_noSnoT$se),type="l",col="blue",
     ylim=c(min_y,max_y), main="standard errors on predictions")

legend(25,max_y,legend=c("AC13"),
       col=c("blue"),
       pch=c(""),lty=c(1), ncol=1)

# Plotting Forecasts
plot(AC13,main="ARIMA(3,0,3)",xlab="Time",ylab="kWh",xlim=c(19010,19070))
lines(lt_noSnoT$pred, col="light blue",lwd=2)
lines(ltFore_noSnoT$fitted, col="blue",lwd=2)

legend("topleft",c("Trained model","Original data","Forecasted data"),
       lty=c(1,1,1,1),col=c("blue","black","light blue"),y.intersp = 0.35,
       cex = 0.75,text.width = 10)
```

Appendix C: ETS(A,N,A) – Hourly data

```
# Clear console and envirronment
cat("\014")
rm(list=ls())

setwd("C:/Users/ali_a/Desktop/Master Thesis/Code")

library(expsmooth)
library(forecast)
library(lubridate)
library(tseries)
library(dplyr)
library(tidyverse)

# TRY 1 out 2: Results included in the study

power_use <- read.csv("Energy consumption per hour-data-2022-03-16
17_50_56_Edited.csv")

dates <- power_use$ï..time
# Plot the data
plot(power_use$watt_hour, type="l")
watt_hour = power_use$watt_hour

# Fill missing data using "Impute" and find daily averages
power_use[power_use == 0] <- NA
power_use[is.na(power_use)] <- mean(watt_hour,na.rm=TRUE)
newdata = power_use %>%
  mutate(dates=as.POSIXct(dates,format="%d-%m-%y %H:%M",tz="")) %>%
  group_by(date(dates)) %>%
  summarise(watt_hour=sum(watt_hour,na.rm=TRUE))

AC13 = newdata$watt_hour
noSnoT = (AC13)
noSnoT

# Splitting data into training and validation period
train_per <- 0.8
limit <- floor(train_per * length(noSnoT))
train_noSnoT <- noSnoT[1:limit]
train_noSnoT
validation_noSnoT <- noSnoT[(limit+1):length(noSnoT)]
validation_noSnoT

train_noSnoT_ts = ts(train_noSnoT, frequency = 7)
# Building the model
x_range =  seq(1,by=1/7,length.out=length(noSnoT))
ses <- ets(train_noSnoT_ts, model = "ANA")
ses.pred <- predict(ses, h = length(validation_noSnoT),level = 0)
ses.pred
```

```r
#Plotting predictions vs actual values
plot(ses.pred,main="ETS (ANA) - Total daily values")
lines(ses.pred$fitted, lwd = 2, col = "blue")
lines(x_range[(limit+1):length(noSnoT)],validation_noSnoT,lwd = 2, col =
"red")
legend("topleft",c("Trained model","Validation data","Training
data","Predicted data"),
       lty=c(1,1,1,1),col=c("blue","red","black","cornflowerblue"),y.intersp
= 0.35,
       cex = 0.75,text.width = 2)

# Calculate metrics
rmse = sqrt(sum((validation_noSnoT -
ses.pred$mean)^2)/length(validation_noSnoT))
rmse

rsq1 <- function (x, y) cor(x, y) ^ 2
pcc = rsq1(validation_noSnoT,ses.pred$mean)
pcc

rss <- sum((ses.pred$mean - validation_noSnoT)^2) ## residual sum of squares
tss <- sum( ( validation_noSnoT - mean(validation_noSnoT) ) ^ 2)  ## total
sum of squares
rsq <- 1 - rss/tss
rsq

#Forecast
horizon = 14
ses_all <- ets(ts(noSnoT, frequency = 7), model = "ANA")
ses.pred.all <- forecast(ses_all, h = horizon,level = 0)
ses.pred.all
plot(ses.pred.all,main="ETS(ANA) - Forecasted daily values")
lines(ses.pred.all$fitted, lwd = 2, col = "blue")

legend("topleft",c("Trained model","Original data","Forecasted data"),
       lty=c(1,1,1),col=c("blue","black","light blue"),y.intersp = 0.35,
       cex = 0.75,text.width = 2)
```

Appendix D: ETS(A,N,A) – Daily data

```r
# Clear console and envirronment
cat("\014")
rm(list=ls())

setwd("C:/Users/ali_a/Desktop/Master Thesis/Code")

library(expsmooth)
library(forecast)
library(lubridate)
library(tseries)
library(dplyr)
library(tidyverse)

# TRY 1 out 2: Results included in the study

power_use <- read.csv("Energy consumption per hour-data-2022-03-16
17_50_56_Edited.csv")

dates <- power_use$ï..time
# Plot the data
plot(power_use$watt_hour, type="l")
watt_hour = power_use$watt_hour

# Fill missing data using "Impute" and find daily averages
power_use[power_use == 0] <- NA
power_use[is.na(power_use)] <- mean(watt_hour,na.rm=TRUE)
newdata = power_use %>%
  mutate(dates=as.POSIXct(dates,format="%d-%m-%y %H:%M",tz="")) %>%
  group_by(date(dates)) %>%
  summarise(watt_hour=sum(watt_hour,na.rm=TRUE))

AC13 = newdata$watt_hour
noSnoT = (AC13)
noSnoT

# Splitting data into training and validation period
train_per <- 0.8
limit <- floor(train_per * length(noSnoT))
train_noSnoT <- noSnoT[1:limit]
train_noSnoT
validation_noSnoT <- noSnoT[(limit+1):length(noSnoT)]
validation_noSnoT

train_noSnoT_ts = ts(train_noSnoT, frequency = 7)
# Building the model
x_range =  seq(1,by=1/7,length.out=length(noSnoT))
ses <- ets(train_noSnoT_ts, model = "ANA")
ses.pred <- predict(ses, h = length(validation_noSnoT),level = 0)
ses.pred

#Plotting predictions vs actual values
```

```
plot(ses.pred,main="ETS (ANA) - Total daily values")
lines(ses.pred$fitted, lwd = 2, col = "blue")
lines(x_range[(limit+1):length(noSnoT)],validation_noSnoT,lwd = 2, col =
"red")
legend("topleft",c("Trained model","Validation data","Training
data","Predicted data"),
       lty=c(1,1,1,1),col=c("blue","red","black","cornflowerblue"),y.intersp
= 0.35,
       cex = 0.75,text.width = 2)

# Calculate metrics
rmse = sqrt(sum((validation_noSnoT -
ses.pred$mean)^2)/length(validation_noSnoT))
rmse

rsq1 <- function (x, y) cor(x, y) ^ 2
pcc = rsq1(validation_noSnoT,ses.pred$mean)
pcc

rss <- sum((ses.pred$mean - validation_noSnoT)^2) ## residual sum of squares
tss <- sum( ( validation_noSnoT - mean(validation_noSnoT) ) ^ 2)  ## total
sum of squares
rsq <- 1 - rss/tss
rsq

#Forecast
horizon = 14
ses_all <- ets(ts(noSnoT, frequency = 7), model = "ANA")
ses.pred.all <- forecast(ses_all, h = horizon,level = 0)
ses.pred.all
plot(ses.pred.all,main="ETS(ANA) - Forecasted daily values")
lines(ses.pred.all$fitted, lwd = 2, col = "blue")

legend("topleft",c("Trained model","Original data","Forecasted data"),
       lty=c(1,1,1),col=c("blue","black","light blue"),y.intersp = 0.35,
       cex = 0.75,text.width = 2)
```

Appendix E: Linear Regression models – Hourly data only

```
library(readxl)
library(ggplot2)
library(forecast)

# Clear console and envirronment
cat("\014")
rm(list=ls())

data1=as.data.frame(read_excel("C:/Users/ali_a/Desktop/Master
Thesis/Code/cleaned_data1_1.xls"))


# Split data into two sets: training and testing
set.seed(1234)
index = sample(1:nrow(data1),round(0.80*nrow(data1)))
train = data1[index,]
test = data1[-index,]
nValid = length(test)

################ 1 - Simple linear Regression: Consumption
##################################################################

#Build the model

model1=lm(watt_hour ~ watt_hour_LR , data=train)
summary(model1)

#Calculate R^2
rss1 <- sum(na.omit(predict(model1,test) - test$watt_hour)^2) ## residual sum
of squares
tss1 <- sum(na.omit((test$watt_hour - mean(na.omit(test$watt_hour)))) ^ 2)
## total sum of squares
rsq12 <- 1 - rss1/tss1

watt_hour_predicted1 = predict(model1,test)
#Calculate RMSE
rmse1 = sqrt(sum((watt_hour_predicted1 -
test$watt_hour)^2)/length(watt_hour_predicted1))
rmse1

#Calculate Pearson's coefficient
PCC <- function (x, y) cor(x, y) ^ 2
pcc1 = PCC(test$watt_hour,watt_hour_predicted1)
pcc1

# Scatter plot
ggplot(data=test,aes(x=watt_hour, y=watt_hour_predicted1))+
  geom_point()+
  geom_abline(intercept=0, slope=1)+
  ggtitle("Consumption") + # and Binary")
  xlab("Actual consumption in kWh") + ylab("Predicted comsumption in kWh")

################ 2 - Multiple linear Regression: Consumption + Temp
#################################################################

#Build model

model2=lm(watt_hour ~ watt_hour_LR + Temp, data=train)
summary(model2)

#Calculate R^2
rss2 <- sum(na.omit(predict(model2,test) - test$watt_hour)^2) ## residual sum
of squares
tss2 <- sum(na.omit((test$watt_hour - mean(na.omit(test$watt_hour)))) ^ 2)
## total sum of squares
rsq2 <- 1 - rss2/tss2

watt_hour_predicted2 = predict(model2,test)

#Calculate RMSE
rmse2 = sqrt(sum((watt_hour_predicted2 -
test$watt_hour)^2)/length(watt_hour_predicted2))
rmse2

#Calculate Pearson's coefficient
pcc2 = PCC(test$watt_hour,watt_hour_predicted2)
pcc2

#Scatter plot
ggplot(data=test,aes(x=watt_hour, y=watt_hour_predicted2))+
  geom_point()+
  geom_abline(intercept=0, slope=1)+
  ggtitle("Consumption + Temperature ") + # and Binary")
  xlab("Actual consumption in kWh") + ylab("Predicted comsumption in kWh")

################ 3 - Multiple linear Regression: Consumption + Temp +
isWeekend #################################################################

#Build Model

model3=lm(watt_hour ~ watt_hour_LR + Temp + isWeekend, data=train)
summary(model3)

##Calculate R^2
rss3 <- sum(na.omit(predict(model3,test) - test$watt_hour)^2) ## residual sum
of squares
tss3 <- sum(na.omit((test$watt_hour - mean(na.omit(test$watt_hour)))) ^ 2)
## total sum of squares
rsq3 <- 1 - rss3/tss3

watt_hour_predicted3 = predict(model3,test)

#Calculate RMSE
rmse3 = sqrt(sum((watt_hour_predicted3 -
test$watt_hour)^2)/length(watt_hour_predicted3))
```

```
rmse3

#Calculate Pearson's coefficient
pcc3 = PCC(test$watt_hour,watt_hour_predicted3)
pcc3

#Scatter plot
ggplot(data=test,aes(x=watt_hour, y=watt_hour_predicted3))+
  geom_point()+
  geom_abline(intercept=0, slope=1)+
  ggtitle("Consumption + Temperature + isWeekend") + # and Binary")
  xlab("Actual consumption in kWh") + ylab("Predicted comsumption in kWh")

################ 4 - Multiple linear Regression: Consumption vs Other
Variables ##################################################################

#Build Model

model4=lm(watt_hour ~ watt_hour_LR + Temp + isWeekend +AMPM , data=train)
summary(model4)

#Calculate R^2
rss4 <- sum(na.omit(predict(model4,test) - test$watt_hour)^2) ## residual sum
of squares
tss4 <- sum(na.omit((test$watt_hour - mean(na.omit(test$watt_hour)))) ^ 2)
## total sum of squares
rsq4 <- 1 - rss4/tss4

watt_hour_predicted4 = predict(model4,test)

#Calculate RMSE
rmse4 = sqrt(sum((watt_hour_predicted4 -
test$watt_hour)^2)/length(watt_hour_predicted4))
rmse4

#Calculate Pearson's coefficient
pcc4 = PCC(test$watt_hour,watt_hour_predicted4)
pcc4

#Scatter plot
ggplot(data=test,aes(x=watt_hour, y=watt_hour_predicted4))+
  geom_point()+
  geom_abline(intercept=0, slope=1)+
  ggtitle("Consumption + Temperature + isWeekend + AMPM") + # and Binary")
  xlab("Actual consumption in kWh") + ylab("Predicted comsumption in kWh")
```

Appendix F: LSTM – Hourly data

```
%reset
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import torch.optim as optim
import datetime
import seaborn as sns
import tensorflow as tf

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.optimizers import Adam

import sys
import time
import warnings
warnings.filterwarnings("ignore")
import logging
logging.disable(logging.CRITICAL)


## load data (empty cells are treated as 0)
file = (r'C:\\Users\ali_a\Desktop\Master Thesis\Code\Energy consumption per
hour-data-2022-04-24 13_58_34_APRIL.csv')

df = pd.read_csv (file)
df

df.index = pd.to_datetime(df['time'], format = '%d-%m-%y %H:%M')
df[:26]

power_ac13 = df['watt_hour']
power_ac13.plot(ylabel='kWh');

# Cleaning data

power_ac13_replaced = power_ac13[power_ac13 != 0]
print(f"{len(power_ac13)-len(power_ac13_replaced)} empty values have been
omitted")

#Calculate the mean for the series with the omitted empty values
power_ac13_replaced_mean = np.mean(power_ac13_replaced)
power_ac13_replaced_mean


#Impute: replace empty values with mean
```

```
power_ac13[power_ac13 == 0] = power_ac13_replaced_mean
print(f"length of Imputed series is {len(power_ac13)}")

print(power_ac13[power_ac13 == power_ac13_replaced_mean])

# # Box-plot to see the outliers
sns.boxplot(power_ac13)

AC13 = power_ac13
AC13.plot(ylabel='kWh')

#comparing before and after data cleaning

plt.figure(figsize=(16,8))
plt.subplot(2,2,1)
sns.distplot(power_ac13)
plt.subplot(2,2,2)
sns.boxplot(power_ac13)
plt.subplot(2,2,3)
sns.distplot(AC13)
plt.subplot(2,2,4)
sns.boxplot(AC13)
plt.show()


Ac13 = AC13

# Creating a function to assemble the data and make it appropriate for the
machine learning model

def df_to_new(df,window_size): #window size the value of used data to
forecast the future data at window_size+1
    df_as_np = df.to_numpy()
    X = []
    y = []
    for i in range(len(df_as_np)-window_size):
        row = [[a] for a in df_as_np[i:i+window_size]]
        X.append(row)
        label = df_as_np[i+window_size]
        y.append(label)

    return np.array(X),np.array(y)

win_size = 20 #This is the sliding window
num_features = 1 #since we only have one variable
X,y = df_to_new(Ac13,win_size)

X.shape,y.shape

pd.DataFrame(y)

#Split the data into training and testing
perc_train = 0.8
limit_train = int(np.floor(len(Ac13)*perc_train))
xtrain,ytrain = X[:limit_train],y[:limit_train]
xval,yval = X[limit_train:],y[limit_train:]
```

```
#creating the model
dropout_rate = 0.2
model1 = Sequential()
model1.add(InputLayer((win_size,1))) # the inputlayer should be equal to the
# of variables and number of inputs at a time
#                                       # (which is equal to win_size), 1 is
the number of features
model1.add(LSTM(32))
model1.add(Dropout(dropout_rate))
model1.add(Dense(128,'relu')) # Dense feeds the outputs from the previous
cell to all its neurons
model1.add(Dropout(dropout_rate))
model1.add(Dense(1,'linear'))
model1.summary()

#creating a location to save the model
import os
checkpointpath = 'C:\\Users\\ali_a\\training/cp.ckt'
cp = ModelCheckpoint(checkpointpath, save_best_only=True,verbose=1) # the one
with the lowest validation loss:
                                                              # what we care about the
model predicting well on unseen
learningRate = 0.0001
model1.compile(loss=MeanSquaredError(),optimizer =
               Adam(learning_rate=learningRate),
               metrics=[RootMeanSquaredError()]) # The higher the learning
rate the faster the
                                                          # model tries to decrease
its loss 0.0001


#fitting the model
import time
EPOCH = 150
batchSize = 15
t0 = time.time()
model1.fit(xtrain,ytrain,validation_data=(xval,yval),batch_size=batchSize,epo
chs=EPOCH,callbacks=[cp],shuffle=False)
t1 = time.time()

print(f'total time for simulation was {t1-t0} seconds, {(t1-t0)/60} minutes')
from tensorflow.keras.models import load_model
model1.save("my_model")
model1 = load_model("my_model")

train_predictions = model1.predict(xtrain).flatten() # flatten() removes the
brackets inside the data
train_results = pd.DataFrame(data={'Train
Predictions':train_predictions,'Actual values':ytrain})
train_results

scale = len(train_predictions)

plt.plot(train_results['Train Predictions'][:scale],label='Trained
Predictions')
plt.plot(train_results['Actual values'][:scale],'b',label='Actual Values')
plt.legend(bbox_to_anchor =(0.75, 1.15), ncol = 2)
plt.show()

val_predictions = model1.predict(xval).flatten() # flatten() removes the
brackets inside the data
val_results = pd.DataFrame(data={'Validate
Predictions':val_predictions,'Validation values':yval}) #yval are the actual
values
val_results

scale2 = len(val_predictions)

plt.plot(val_results['Validate Predictions'][:scale2],label='Trained
Predictions')
plt.plot(val_results['Validation values'][:scale2],'b',label='Actual Values')
plt.legend(bbox_to_anchor =(0.75, 1.15), ncol = 2)
plt.show()

#RMSE
rmse = np.sqrt(((val_results['Validate Predictions']-val_results['Validation
values'])**2).sum()/len(val_results['Validate Predictions']))

#Rsquared

RSS = ((val_results['Validate Predictions']-val_results['Validation
values'])**2).sum()
TSS = ((val_results['Validation values']-np.mean(val_results['Validation
values']))**2).sum()
rsq = 1 - RSS/TSS

#Pearson's correlation coefficient
def pearsonr(x, y):
  # Assume len(x) == len(y)
  n = len(x)
  sum_x = float(sum(x))
  sum_y = float(sum(y))
  sum_x_sq = sum(xi*xi for xi in x)
  sum_y_sq = sum(yi*yi for yi in y)
  psum = sum(xi*yi for xi, yi in zip(x, y))
  num = psum - (sum_x * sum_y/n)
  den = pow((sum_x_sq - pow(sum_x, 2) / n) * (sum_y_sq - pow(sum_y, 2) / n),
0.5)
  if den == 0: return 0
  return num / den
PCC = pearsonr(val_results['Validation values'],val_results['Validate
Predictions'])

print(f'rmse is {rmse}')
print(f'R**2 is {rsq}')
print(f'PCC is {PCC}')


# Assembling forecasts in an array
prediction = [] #Empty list to populate later with predictions
current_batch = xval[-win_size] #Final data points in train
current_batch = current_batch.reshape(1, win_size,num_features) #Reshape
f_horizon = 240 # Number of hours ahead
for i in range(len(xval) + f_horizon):
    current_pred = model1.predict(current_batch)[0]
    prediction.append(current_pred)
    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],axis=1)

current_pred = model1.predict(current_batch)
# Plotting forecasts

plt.plot(prediction)
plt.title(f'EPOCH = {EPOCH}, learning rate = {learningRate}');
plt.xlabel("Time");
plt.ylabel("kWh");
xnew = np.arange(0, len(AC13))
yrange = np.arange(xnew[-1]+1,xnew[-1]+len(prediction)+1)
plt.plot(xnew,AC13)
plt.plot(yrange,prediction);
plt.title("Total data and forecasted values");
plt.xlabel("Time")
plt.ylabel("Consumption in kWh")
plt.legend(['Entire dataset',"Forecasted values"],loc='upper left');
```

Appendix G: LSTM – Daily data

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import torch.optim as optim
import datetime
import seaborn as sns
import tensorflow as tf


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.optimizers import Adam


import sys
import time
import warnings
warnings.filterwarnings("ignore")
import logging
logging.disable(logging.CRITICAL)

## load data (empty values are treated as 0)
file = (r'C:\\Users\ali_a\Desktop\Master Thesis\Code\Energy consumption per
hour-data-2022-03-16 17_50_56_Edited.csv')

df = pd.read_csv (file)
df

df.index = pd.to_datetime(df['time'], format = '%d-%m-%y %H:%M')
df[:24]

power_ac13 = df['watt_hour']
power_ac13.plot();

#Remove null entries
power_ac13[power_ac13 != 0]
power_ac13_replaced = power_ac13[power_ac13 != 0]
print(f"{len(power_ac13)-len(power_ac13_replaced)} empty values have been
omitted")

#Calculate the mean for the series with the omitted empty values
power_ac13_replaced_mean = np.mean(power_ac13_replaced)
power_ac13[power_ac13==0]=np.mean(power_ac13_replaced)

print(f"length of Imputed series is {len(power_ac13)}")

print(power_ac13[power_ac13 == power_ac13_replaced_mean])
# Transofrming the values from hourly to daily
```

```
newdata13 = power_ac13.resample('D').sum()
newdata13
plt.plot(newdata13)

type(newdata13)
len(newdata13)
print(newdata13)

# Creating ta function to assemble data for the machine learning model

def df_to_new(df,window_size): #window size the value of used data to
forecast the future data at window_size+1
    df_as_np = df.to_numpy()
    X = []
    y = []
    for i in range(len(df_as_np)-window_size):
        row = [[a] for a in df_as_np[i:i+window_size]]
        X.append(row)
        label = df_as_np[i+window_size]
        y.append(label)

    return np.array(X),np.array(y)

# Pearson correlation
def pearsonr(x, y):
  # Assume len(x) == len(y)
  n = len(x)
  sum_x = float(sum(x))
  sum_y = float(sum(y))
  sum_x_sq = sum(xi*xi for xi in x)
  sum_y_sq = sum(yi*yi for yi in y)
  psum = sum(xi*yi for xi, yi in zip(x, y))
  num = psum - (sum_x * sum_y/n)
  den = pow((sum_x_sq - pow(sum_x, 2) / n) * (sum_y_sq - pow(sum_y, 2) / n),
0.5)
  if den == 0: return 0
  return num / den

win_size = 5 #Sliding window
num_features = 1 #since we only have one variable
X,y = df_to_new(newdata13,win_size)

#Split the data
perc_train = 0.8
limit_train = int(np.floor(len(newdata13)*perc_train))

xtrain,ytrain = X[:limit_train],y[:limit_train]
xval,yval = X[limit_train:],y[limit_train:]

#creating the model
dropout_rate = 0.2
model1 = Sequential()
model1.add(InputLayer((win_size,1))) # the inputlayer should be equal to the
# of variables and number of inputs at a time
#                                      # (which is equal to win_size), 1 is
equal to the number of features
model1.add(LSTM(128)) #check how the input of LSTM can affect the forecast
```

```
model1.add(Dense(256,'relu')) # Dense feeds the outputs from the previous
cell to all its neurons
model1.add(Dropout(dropout_rate))
model1.add(Dense(256,'relu')) # Dense feeds the outputs from the previous
cell to all its neurons
model1.add(Dropout(dropout_rate))
model1.add(Dense(1,'linear'))
model1.summary()

#creating a directory to save the model
import os
checkpointpath = 'C:\\Users\\ali_a\\training_daily/cp.ckt'
cp = ModelCheckpoint(checkpointpath, save_best_only=True,verbose=1) # the one
with the lowest validation loss:
                                                     # what we care about the
model predicting well on unseen
model1.compile(loss=MeanSquaredError(),optimizer =
               Adam(learning_rate=0.001),
               metrics=[RootMeanSquaredError()]) # The higher the learning
rate the faster the
                                                 # model tries to decrease
its loss

#fitting the model
EPOCH = 150
model1.fit(xtrain,ytrain,validation_data=(xval,yval),batch_size=3,epochs=EPOC
H,callbacks=[cp],shuffle=False)

from tensorflow.keras.models import load_model
model1.save("my_model")
model1 = load_model("my_model")

train_predictions = model1.predict(xtrain).flatten() # flatten() removes the
brackets inside the data
train_results = pd.DataFrame(data={'Train
Predictions':train_predictions,'Actual values':ytrain})
train_results

scale = len(train_predictions)

plt.plot(train_results['Train Predictions'][:scale],label='Trained
Predictions')
plt.plot(train_results['Actual values'][:scale],'b',label='Actual Values')
plt.legend(bbox_to_anchor =(0.75, 1.15), ncol = 2)
plt.show()

val_predictions = model1.predict(xval).flatten() # flatten() removes the
brackets inside the data
val_results = pd.DataFrame(data={'Validate
Predictions':val_predictions,'Validation values':yval})
val_results

#RMSE
rmse = np.sqrt(((val_results['Validate Predictions']-val_results['Validation
values'])**2).sum()/len(val_results['Validate Predictions']))
print(f'rmse is {rmse}')
```

```
#Rsquared

RSS = ((val_results['Validate Predictions']-val_results['Validation
values'])**2).sum()
TSS = ((val_results['Validation values']-np.mean(val_results['Validation
values']))**2).sum()
rsq = 1 - RSS/TSS
print(f'R**2 is {rsq}')

pcc = pearsonr(val_results['Validation values'],val_results['Validate
Predictions'])
print(f'PCC is {pcc}')

# Function to store future values
prediction = [] #Empty list to populate later with predictions
current_batch = xval[-win_size] #Final data points in train
current_batch = current_batch.reshape(1, win_size,num_features) #Reshape
f_horizon = 6 # Number of 5 minutes ahead
for i in range(len(xval) + f_horizon):
    current_pred = model1.predict(current_batch)[0]
    prediction.append(current_pred)
    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],axis=1)

current_pred = model1.predict(current_batch)

# Forecast curve shape

plt.plot(prediction)
```