UNIVERSITY OF SOUTHAMPTON

Charging Load Forecasting of Electric Vehicles using Deep Learning

by

Constantinos Hadjigregoriou

Supervisor: Dr. Jie Zhang
Second Examiner: Dr. Enrico Marchioni

A thesis submitted in partial fulfillment for the
degree of Master of Science in Artificial Intelligence

in the
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

September 2022

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

MSc Artificial Intelligence

by Constantinos Hadjigregoriou

The increased penetration of Electric vehicles increases the probability of negatively affecting power networks. The power demand will be higher and variations of voltage levels will be increased. Accurately forecasting the energy demand ahead of time is critical. Previous work was done on short-term and medium-term time frame for energy prediction. This thesis explores ultra-short term (up to 1 hour ahead) energy prediction at electric vehicle stations using Deep Learning techniques. Extensive experiments are made investigating the performance of models using various input lengths of prior data and various output time step predictions into the future. Multi-variate models are also investigated using weather features, an investigation only previously performed on short-term prediction time frame. The data used originate from 13 stations in a Council of Scotland and span 3 years. The data are Event based charging events which were converted to Time series format, before neural network training is performed.

Contents

List of Figures

List of Tables

Nomenclature

ANN	Artificial Neural Network
EV	Electric Vehicle
CNN	Convolutional Neural Network
kWh	Kilowatt hour
$LSTM$	Long Short-Term Memory
NN	Neural Network
$ReLU$	Rectified Linear Unit
RNN	Recurrent Neural Network

Acknowledgements

I would like to thank my Supervisor Dr. Jie Zhang for his guidance, especially in the starting stages when I had to research and select a project.
I also want to thank my family and girlfriend for their continuous support support.

Chapter 1

Introduction

The shortage of fossil fuels and global warming caused by excessive carbon emissions in recent years has led to the adoption of renewable energy alternatives. One example are internal combustion engines, which run on petroleum and are slowly being replaced by electric vehicles (EVs). EVs operate with electricity, therefore petroleum consumption for transportation is reduced and also they don't not emit any polluting gases, improving the urban air quality [23].

Load forecasting can be divided into four categories. These are super short-term (1 second to 1 hour ahead), short-term (1 hour to 1 day or 1 week ahead), medium-term (1 month to 1 year ahead) and long-term (1 year to 10 years ahead), load forecasting. Ultra-short term load forecasting has a timeframe of prediction less than or equal to 1 hour ahead [23]. Super short-term and Short-term load forecasting is useful for utility optimal operations and scheduling. Long-term load forecasting is required in the grid system planning stage. Super-short term forecasting, such as minute level is used in real time power quality and security monitoring [24].

The ability to accurately predict the energy consumption at charging stations is beneficial for knowing when to start and stop internal generators, which are part of the grid the stations are connected, having therefore an economic benefit and also sustaining a stable grid operation [11].

Spyros Makridakis et.al [12] investigated the accuracy (of unseen by the models test data), of 8 popular machine learning methods 2 modern deep learning methods (RNNs and LSTMs) compared to eight traditional statistical methods on time series data used in the M3 competition. The M3 competition is comprised of 1045 time series datasets. For single-variate single step ahead forecasts it was found that 7 out of the 8 statistical methods outperform the machine learning methods. The best results were produced by ETS and ARIMA models. The statistical methods performed better with single-variate mulit-step predictions as well. Statistical techniques work very well with single-variate data and single-step predictions. Machine methods do offer promises when working with multivariate and multi-step forecasts, however. Deep learning methods specifically alleviate the problem of the scientist required to hand craft features. It is time consuming testing out different features and in order to come up with them one must have expert knowledge in the field of the problem. In addition, when the complexity of the problem is high there is a grater chance machine learning techniques produce better forecast results.

Previous works have largely looked at statistical methods and state of the art machine learning methods on the problem of load forecasting. Only three papers were found to use Deep learning techniques which will be examined in more detail in the next section.

This thesis aims at tackling the problem of load forecasting on the super-short term time frame. Several different deep learning models will be experimented, evaluated to find the best performers for the problem at hand. Mult-step forecast models and Multi-variate models will be tested. For the particular EV load forecasting problem only one thesis in the past attempts to use multiple variables, but their deep learning model is very basic. The complexity of the energy consumption data is expected to be high compared to the average time series data. This is because EV charging load is periodic, highly fluctuating and has large spikes at certain times of high demand [24]. It is therefore expected that Deep learning techniques to be used will be competitive with statistical methods. Statistical methods ARIMA and ETS, which were the best performing in study [12] will also be used to make predictions and compare their results to the deep learning models.

The three pillar questions to be answered in this thesis are:

- Are deep learning methods superior to statistical method for forecasting EV load?.
- Which deep learning models are the best performers?
- Are multi-variate DL models improving the performance of single-variate models?

Chapter 2

Literature Review

Pooya Zadeh et. al [22] surveyed the application of Deep Learning approaches in Electric vehicles, which are part of Micro grids. Their survey covered a variety of problems such as Energy Management, Pricing for PEV Charging, EV Charging Scheduling and Load Forecasting. Our project is on Load Forecasting, therefore the two out of twenty total papers presented were identified to be examined in greater detail. [7] is another survey by Yvenn Amara-Ouali et. al in which the authors examine strategies for overcoming the complications caused by having an increased number of electric vehicles. They reviewed EV models that make use of stochastic processes and machine learning approaches.

Juncheng Zhu et.al [11] use an LSTM model to perform super short-term load forecasting, which as they referred to, is load forecast within 1 hour ahead. They use an LSTM based network; however, they do not specify its exact architecture (the number of hidden layers and the number of LSTM units per hidden layer). Their dataset contains charging events provided by a company that operates charging stations in Sehenzhen, China. They split their data into a train and test sets with a ratio split of 7:3 and 10% of the train set is used as a validation set. Their data pre-processing is made in three steps. In the first step they create a box plot and the data above and below the upper and lower bounds of the box plot are considered as outliers. These outliers are replaced by the mean value of the energy consumption of the day. In the second step they do time interval processing and using one year of the dataset they create a 30 minute interval and a 15 minute interval dataset. In the last step, they perform data normalization and they use the Maximum Minimum normalization technique. For the input matrix to the model they use a sliding window of 1 and a step size of 1. To evaluate the models they use the RMSE and MAE metrics and as shown by their loss graphs they trained their models for about 30 epochs. The authors also experimented with various optimizers, which were Adam, SGD, RMSprop, Adadelta and Adagrad and found out that Adam optimizer achieved the best results in comparison with the other optimisers. They also used an ANN model for comparison, which was outperformed by their LSTM models in all time intervals and metrics by at least double the performance. On the test set their best results were 12.3 (15min- RMSE), 28.3(30min- RMSE), 5.5 (15min – MAE), 16.9 (30min-MAE).

Zhile Yang et.al [24] investigate single-variate super-short term load forecasting and use data at the minute level to make predictions. They use the same dataset as [11]. Data pre-processing include identifying outliers using an equation given in the paper [24] and with an interpolation method the outliers are replaced. After they scale the data using a minimum maximum scaler. Six models are trained and evaluated their performance. These are an ANN, RNN, LSTM, gated recurrent units (GRU), stacked auto encoders (SAE) and bi-directional LSTM (Bi-LSTM). For the LSTM, Bi-LSTM, RNN and GRU 2 hidden layers are used SAEs have four hidden layers. All the model's hidden layers have 16 nodes. The authors experimented with the learning rate between the range [1e-5, 0.01] and found the models performed the best with a learning rate of 1e-3. They used Dropout layers in their models and experimented with the dropout ratio in the range [0.3,08] and found that models with dropout set to 0.3 have better results. They trained all their models for 30 epochs and used a batch size of 512. Their optimizer of choice was RMSprop and for the loss function they choose MAE. They repeated their experiments for different input sequences of length, 1, 5 and 15 and the metrics they used to evaluate the performances were RMSE and MAE.

Jiankang Zhang et. al [23] used the same dataset as the previous authors to perform load forecasting. The pre-processing steps of their work started with missing data detection. Missing data were filled with the average of the previous and next with respect to time charging event's load consumed. For outliers the previous data value was used. Next, they converted the data to hourly samples by dividing the charging events in 24 time points every day. Lastly, they used a min-max scaler to scale the data. Their model was multivariate. The additional features they used were, the real time electricity prices during peak and valley periods and a binary holiday mark. This feature was set to 0 for a weekday, 1 and 2 for the weekend days respectively and if the day was a special holiday it was set to 3. They used a GRU model. It consisted of an input, 2 hidden, 1 dropout and the output layer. The input layer had 6 units, while the hidden layers had 64 units each. They experimented with all the optimizers like the authors of [11] and concluded Adam achieved the best results. MSE was the loss function of choice and for the evaluation the adopted the metrics NRMSE and NMAE, which stand for normalized RMSE and normalized MAE. For their experiments the authors varied the number of hidden layers of their models, producing performance results for models with 2,3 and 4 hidden layers.

Alexander Orzechowski [17] investigated the same problem as in our thesis and used the same dataset. The work done was on short-term energy consumption prediction. It is also the first project were weather was investigated and found that these features improved the prediction results by approximately 9% on average over all error metrics.

The approach taken did not involve conversion of data to time series, instead the event based data are split using time stamps and fed directly to the ANN.

Chapter 3

Background

3.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) have the ability to retain information about previous events. With a traditional artificial neural network, we have inputs which are fed into the NN, which in turn outputs a prediction. The structure of an RNN is such, that the input contains information about the previous events that occurred. They do so, by containing loops, which is what allows for information to persist [9].

The structure of an RNN unit, A is depicted in Figure 3.1, where x_t is the input at time t and h is the output at time t. If we unroll the RNN structure on the left, then we can see that an RNN is equivalent to multiple copies of itself and the output of the network at timestep t is fed as an input together with the new input to a copy of the network at time step t+1. It is therefore deduced that RNNs are perfect when dealing with sequential data [9].

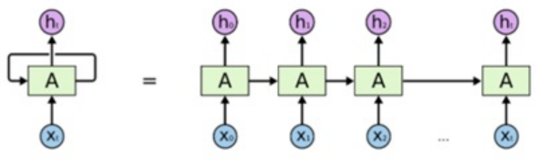


FIGURE 3.1: RNN unrolled

When the gap between the information from the past is needed at the current time step for the prediction is small, then RNNs can work very well. However, there are cases were the relevant information lie a substantial time step in the past, so the gap is large, which is where RNNs fail to perform well. As an example, a small gap can represent information at time step t_5, which are useful for making a good prediction at time step t_{10} and a large gap can represent information at time step t_5, which are useful for making a good prediction at time step t_{100} [9].

3.2 Long Short Term Memory Networks

Long Short-Term Memory networks is a variant of a RNN, which is competent at learning long-term dependencies. They have a chain-like structure just like the RNN, but their repeating unit differs. As shown in Figure 3.2, a standard RNN repeating module has a single neural network layer, for example a tanh layer. In an LSTM the repeating module has four layers which are connected to each other as depicted in the Figure[9].

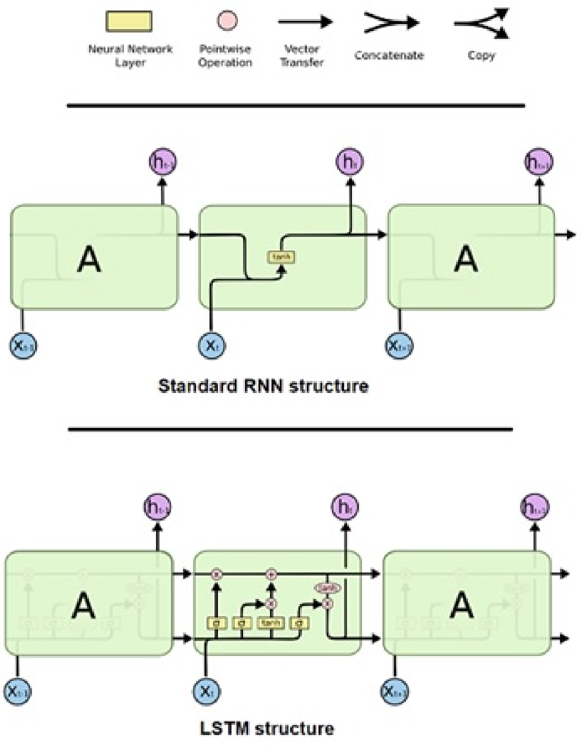


Figure 3.2: RNN vs LSTM structure

To better understand how LSTM operate we will go through its repeating module, starting from the input and ending at the output explaining each step. Figure 3.3 illustrate the steps and Figure 3.4 gives the mathematical equations that describe each step.

Step 1 In this step, information h_{t-1} and x_t are being looked at and fed into a sigmoid function. The sigmoid function outputs a number between 0 and 1. 0 represents discarding this information and 1 keeping the information content at 100%. This number is outputted for each number contained in the C_{t-1} cell state[9].

Step 2 In this step, it is decided which information to store in the cell state. First the values to be updated are decided using the input gate layer which is a sigmoid layer. Then a tanh layer creates the vector $C\ _t$. This vector represents the values which are possible to be added to the state[9].

Step 3 In this step, the decisions made in the previous two steps are applied, so that the old cell state C_{t-1} is updated to the new cell state C_t. To achieve this, the old state is multiplied by what it has been previously decided to be forgotten (by multiplying f_t with C_{t-1}) . $i_t * \tilde{C}_t$ is then added, which represents the new scaled candidate values[9].

Step 4 In this final step, the output is decided. The output is a filtered form of the contents of the cell state. First the cell state goes through a sigmoid layer, in which is decided which elements of the cell state to output. The contents of the cell state are input in a tanh layer, which makes their value between -1 and 1. These are multiplied by the output of the sigmoid layer to output only the desired elements[9].

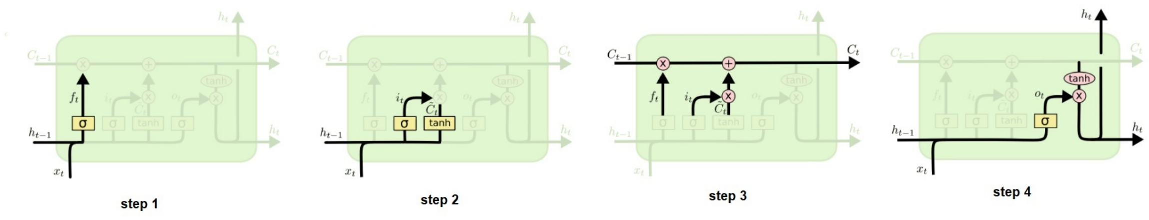


FIGURE 3.3: LSTM details (modified from [9])

step 1: $f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$

step 2: $i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$
$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$

step 3: $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

step 4: $o_t = \sigma\left(W_o \; [h_{t-1}, x_t] + b_o\right)$
$h_t = o_t * \tanh\left(C_t\right)$

FIGURE 3.4: LSTM equations (modified from [9])

3.3 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a NN with a structure designed to process input data in structured arrays. Its essence derives from a unique layer, the Convolutional layer which utilizes the convolution operation instead of matrix multiplication like a normal layer. Its architecture is a feed-forward neural network, which is created by having multiple hidden layers stacked in a sequence. The hidden layers are convolutional layers usually and following are activation layers followed by other convolutional layers etc. Sometimes following a convolution and activation layers are pooling layers [10].

A typical CNN architecture is shown in Figure 3.5. In this example the input are vehicle images, and the CNN is used to classify the input images into the different vehicle types, that is why the last activation layer is a SoftMax layer. Also, the activation layers throughout this network are ReLU activations[10].

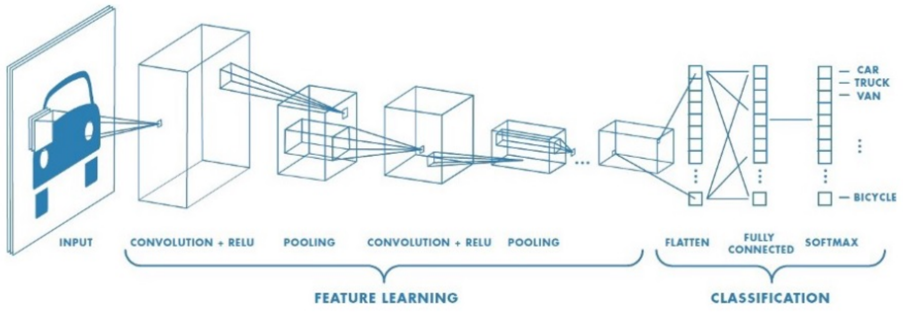


FIGURE 3.5: CNN example architecture (sourced from [14])

The purpose of the convolution operation in a convolutional layer is to extract features from the input. When there are multiple convolutional layers the first layers capture low level features of the input, and the last layers capture higher-level features. In the convolution operation the kernel, which is a small array of numbers is applied across the input array of numbers. The product between each element of the kernel and the input array is calculated and all the products are then summed up to give the output value in the appropriate position of the feature map, which is the output array. The convolution operation is demonstrated in Figure 3.6 [14].

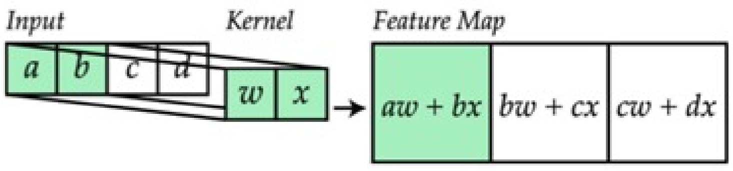


FIGURE 3.6: Convolution operation (sourced from [2])

The pooling layer function is to reduce the feature map's dimensions, so that the parameters that need to be learned by the NN are reduced. In this way overfitting is mitigated and the computing power required to train the network is also reduced. Max pooling is the most used pooling method. In max pooling, the maximum value of each patch of the feature map is calculated, and the other values are discarded [10].

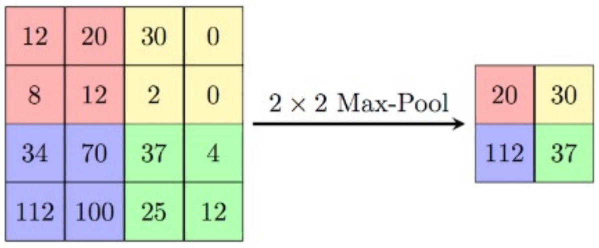


FIGURE 3.7: Max pooling example(sourced from [18])

Following the last convolutional and pooling layer the feature maps are converted from arrays into a vector. Therefore, they are flattened and are connected to one or more fully connected layers. A few fully connected layers constitute to the NN structure of a simple NN.

3.4 LSTM Encoder Decoder

Traditional DNNs use labeled data, where the inputs and target outputs are encoded into vectors with fixed dimensions. However, they cannot be used when a sequence to sequence (seq2seq) model is required [4]. A seq2seq model maps an input of fixed length to an output of fixed length, but the length of the input and output can vary. Seq2seq is used in machine translation and Google translate is powered by such model [15].

One effective model architecture for tackling seq2seq problems is the Encoder Decoder LSTM. Figure 3.8 represents the architecture of an encoder decoder model. As seen in the Figure the model is comprised of three main parts, the Encoder, the Encoder Vector and the Decoder.

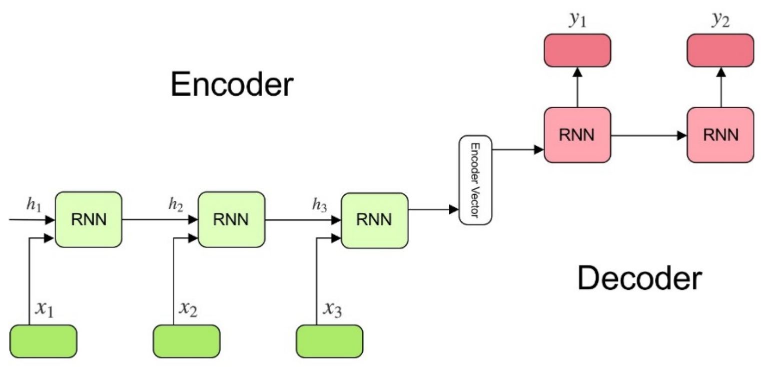


FIGURE 3.8: LSTM Encoder Decoder sequence to seqeunce model(sourced from [15])

The Encoder consists of LSTM units stacked. At each time step unit takes an element of the input sequence, learns its information and propagates it to the next unit for additional process [20]. The hidden states h_i are calculated using Equation X [15].

$$h_t = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$$
$$h_t = f(W^{(hh)}h_{t-1})$$
$$y_t = softmax(W^S h_t)$$

FIGURE 3.9: Max pooling example(sourced from [15])

The Encoded Vector is an in-between state which holds the sequential information of the input. The Decoder which like the Encoder is an LSTM model, takes the information stored in the Encoder vector and predicts the target sequence, element by element [4].

Encoder-Decoder models work very well with sequential data, such as Time series data and the reason is their use of RNN/LSTM layers, as discussed in Section 3.1.

3.5 CNN-LSTM Encoder Decoder

A CNN-LSTM Encoder Decoder model is a hybrid model where a CNN network is used for the Encoder and a LSTM network is used for the decoder. A CNN cannot read direct sequence input, but with a one-dimensional CNN the sequence can be read as a whole, and the network extract its features. The LSTM decoder network uses these features to make predictions.

3.6 SARIMA

SARIMA model stands for Seasonal Autoregressive Integrated Moving Average time series model. This model is used for single-variate time series records with a seasonal trend. A Time series model refers to a model that observes how variables vary over time. Autoregressive refers to a linear model which uses only past period data to make predictions for the future. Moving Average time series uses the moving average of the data as time progresses. It calculates the residual errors between the moving average and the actual data values and uses the error variable into the model. However, the above models show some limitations when it comes to seasonality fluctuations, meaning time series with a recurring phase. The SARIMA model is a combination of either or both above models which also uses differencing terms to eliminate the seasonality patterns in dataset and improve the predictions of the model, called integrating [16]. SARIMA model equation:

SARIMA(p, d, q) x $(P, D, Q)_{lag}$

where:

- p and seasonal P: number of AR variable (lags in the stationary series)
- d and seasonal D: differencing required to convert into a stationary series
- q and seasonal Q: number of MA variables (lags of the forecasting errors)
- lag: time period length of data[5]

Chapter 4

Data

4.1 Energy Consumption data

The data [19] used for this project originate from three different csv files, which when combined produce charging events spanning from 1st September 2016 to 31st August 2019. The charging events are from charging stations in the council area of Scotland, Perth and Kinross. The data consist of the features summarized in Table 4.1 for each charging event.

Feature	**Datatype**
CP ID	Integer
Connector	Integer
Start Date	Integer
Start Time	Integer
End Date	Integer
End Time	Integer
Total kWh	Float
Site	String
Model	String

TABLE 4.1: Dataset features

From these 9 features of each charging instance, we are only concerned with 5 of them. These are the "Start Date", "Start Time", "End Date", "End Time" and "Total kwh".

The charging station in which each charging event has occurred is specified in the "Site" feature. All "sites" names were located using [13], which contains a map of all the EV charging stations in the UK. After all the sites were located, the corresponding feature in the data was renamed with the correct name of the charging station. Table 4.2, portrays all the charging stations names, addresses and exact coordinates. Figure 4.1, shows a maps with all the stations marked.

Station Name	**Coordinates**	**Charging Instances**
Broxden Park and Ride	0.221132144, 56.38654958	14696
Kinross Park and Ride	0.18735141, 56.20688103	12451
South Inch Car Park	0.097791086, 56.39174796	6499
Rie-Achan Road Car Park	0.084805441, 56.70361787	5636
Crieff Hospital	0.068900659,56.369995	4579
Crown Inn Wynd Car Park	0.06775708, 56.295725	4503
Leslie Street Car Park	0.058984622, 56.591155	3920
Canal Street Car Park	0.056215956, 56.394366	3736
Mill Street	0.047458545, 56.397219	3154
Aberfeldy Moness Terrace Car Park	0.039333113, 56.62015799	2614
Atholl Street Car Park	0.035315538, 56.5677509	2347
Friarton Depot	0.033524933, 56.39172899	2228
Market Square Alyth	0.001429474, 56.62260461	95
TEST SITE Charge Your Car HQ	?, ?	21
Total instances		**66458**

Table 4.2: Charging stations locations and number of charging events

Using the "Site" feature and the unique charging station names it was deduced that each charging event occurred at a unique of 14 charging stations. The “***TEST SITE*** Charge Your Car HQ” could not be identified as a charging station and since it only accounts for 21 charging events, it was decided to ignore the events from that station. Therefore, the data are from 13 charging stations as shown in Table 4.3. Table 4.3 shows also that each town/city has 1 station from which we have data from, except Perth which has data from 5 stations.

Station Location (city/town)	**Number of stations**
Perth	5
Kinross	1
Pitlochry	1
Crief	1
Auchterarder	1
Blairgowrie	1
Aberfeldy	1
Dunkeld	1
Alyth	1
Total stations	

Table 4.3: Number of stations per Town/ city

4.2 Weather data

The weather data are collected using the World Weather Online API [21]. The Perth and Kinross council state, in which the charging stations of our data are located covers

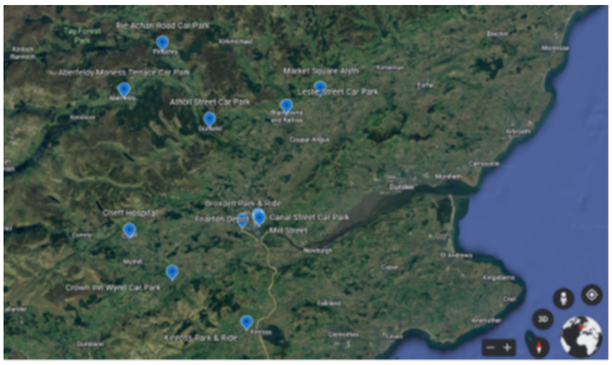


Figure 4.1: Electric vehicle stations mapped

an area of 5286 km^2. The weather in each city/town varies, therefore is not sensible to receive the weather data from the average location of the 13 stations. For this reason it was decided for the multivariate models (they include the weather features) to use a subset of the dataset. Specifically only the charging events that occurred in any one of the five stations in Perth, which account for 45.65% of the whole dataset, where used. Consequently, using the weather API weather information was downloaded for the city of Perth. The greatest frequency that could be accessed was the weather data for each hour. For this reason, the multivariate models will use the hourly energy consumption data to match with the hourly weather data.

Table 4.4 shows the weather information that will be used as features in the multivariate models.

Feature	**Description**	**Type**
tempC	Temperature in degrees Celsius	Integer
windspeedKmph	Wind speed in kilometers per hour	Integer
precipMM	Precipitation in millimeters	Float
humidity	Humidity in percentage (%)	Integer
visibility	Visibility in kilometers	Integer
cloudcover	Cloud cover amount in percentage (%)	Integer

Table 4.4: Weather data features and description

Chapter 5

Time series Forecasting Framework

In order to come up with a framework for the problem at hand, we need to analyze the problem and identify its characteristics. This requires answering the following questions, inspired by [8].

What are the inputs and what are the outputs for one forecast?
The input data for one forecast are the last x minutes or y hours of Energy consumption (kWh) data to forecast the next z minutes Energy consumption, which is the output data. The values of x, y and z will depend on the model.

What type of variables do we have?
There are two types of variables, Endogenous and Exogenous. Endogenous are input variables which are affected by some other variables in the system and the output variable depends on them. Exogenous are input variables which are not affected by any other variables in the system and the output variable depends on them. The Energy consumption variable is endogenous since it is affected by the weather data (at least this is the assumption for using the weather data as features in the first place). Weather data are exogenous since the weather is not affected by the Energy consumption.

Is our problem a Classification or a Regression problem?
In a classification problem the predictions are classified as one of two or as one of many labels. In a regression problem the predictions are numerical quantities. Our problem is a regression problem since we want the prediction to be an Energy consumption quantity.

The time series variables, are they structured or unstructured?
Unstructured variable time series data have no detectable systematic time-dependent pattern, while structured they do. Structured data have a trend and/or a seasonality.

Is our problem a Single-variate or Multivariate time series problem?
single-variate means there is only a single variable measured over time. Multivariate means there are two or more variables measured over time. We also need to distinguish whether our outputs are single-variate or multivariate. In this thesis two problems will be considered, one will be with single-variate input, which is the problem where we forecast the energy consumption base on the past energy consumption only. The other problem will be with Multivariate input, which is the problem where we predict the energy consumption based on the past energy consumption and weather data. The output in both problems is single-variate since we are predicting a single variable, the energy consumption.

Do we want a single-step or multi-step forecast?
A single-step forecasting problem is one that requires prediction of the next single time-step. In a multi-step forecasting problem, we me predictions for x amount of steps into the future. We will experiment with both, starting with single-step forecasting. The framework, we will develop will be a universal multi-step however were the number of time steps to be predicted into the future will be an input and to turn the our problem to single-step forecasting, the input will be set to 1.

Are observations contiguous or discontiguous?
Contiguous observations are ones that are uniform over time, while discontiguous observations have gaps in them. Our data are discontiguous, not only because of missing or incorrect values that need to be removed, which is solved by a data cleaning process, but because they are based on charging events and not all charging events are adjacent. In fact, some are overlapping and some have time period gaps between them. Therefore, a procedure is required after the data cleaning, in which we will convert our discontigious data into contiguous time series data.

Table 5.1 summarises the characteristics of the models to be created. Choices for the characteristics will be explained in Section 6.

Variables	Output type	Sites	Resolution	Input steps	Output steps
Single-variate	Multi-step	All	Minutely	[15,30,60,120,180]	[15,30,60]
Single-variate	Single-step	Perth	Hourly	[4,6,12,24,48,144]	1
Multivariate	Single-step	Perth	Hourly	[4,6,12,24,48,144]	1

Table 5.1: Forecast models characteristics

Chapter 6

Methodology

Section 6 is organised in order of execution of the steps. Some background information is given at each when necessary subsection, if the information was not given in Section 3.

6.1 Data Preparation

The data are comprised of charging events. The first step was to combine the "Start Date" with "Start Time" and "End Date" with "End Time". Two new features were created named "Start Datetime" and "End Datetime" of type pandas.Timestamp. The time units (year, month, day, hour, and minute) for each charging event were assigned for the start and end datetime correspondingly. This datatype will be useful because it is easy to access each unit of time and if necessary, replace it. The smallest time unit for the Timestamps are minutes, since the "Start Time" and "End Time" features have minutes as their smallest time unit.

6.1.1 Data Cleaning

The useful features of the data which are "Start Datetime", "End Datetime" and "kWh" need to be assessed to identify missing values, incorrect values and remove or correct them, which ever appropriate for the occasion. According to the "kWh" feature all data were searched and identified the charging events with NaN values and zero values and negative values. The charging events with NaN values were removed. The charging events with zero values means either they are anomalies or that the vehicle was connected to the port but was not charging. Since we are concerned with predicting Energy consumption and not Charging events, removing these instances or not does not make any difference due to their zero contribution to the total energy consumption. They were removed, to reduce the data. Charging events with negative charging values are definitely an anomaly and they were removed from the data.

“Start datetime” and “End Datetime” were assessed together to verify the datetimes make logical sense. By inspecting the data, it was identified that in the CSV files some dates were in DD/MM/YY format, while others in MM/DD/YY. There were instances even were the “Start Date” was in the first format and the “End Date” in the latter, and the reverse also. Due to the large number of charging events it was too time consuming to go through the dates of each charging events manually. Functions were written to identify the majority of the incorrect dates and correct them. For example, if the “Start Date” was “11/11/2017” and the “End Date” was “11/12/2017” it was assumed that it is very unlikely for the car to have been charging for 30 days and that the “Start Date” is in DD/MM/YY format while the “End Date” is in MM/DD/YY format. A rule was written to identify such cases. In the CSV files the charging events were ordered by datetime descending order. Another rule written was to compare each charging event “Start date” with the previous instance’s “Start date”. If for example the current charging instance is “08/02/2019”, but the previous charging event was “01/08/2019”, it means that the current charging event “Start date” being checked is in MM/DD/YY format, therefore it was corrected. Other rules were to identify if a “Start Date” was chronologically greater than “End Date” and check the Time Difference between the “Start Date” and “End Date”. If the first rule is True and in the second rule if the Time Difference is greater than 1 day, then either is a data anomaly or the dates are not in the desired format. In all cases were the “Start Date” was chronologically greater than “End Date” the two dates had different formats, so they were corrected. In the events with too large “Time Difference” the instances were checked manually and where the date formats were not matching, they were corrected. By the end of this procedure all the dates were adjusted in DD/MM/YY format.

A charging event has an arbitrary duration which is calculated by subtracting the “Start Datetime” from the “End Datetime” and a new feature named “Time duration” was created in which the time duration in minutes for each charging even is saved. Figure 6.1 is a Histogram of the “Time duration” in minutes of the charging events with the y axis showing the total count number of charging events with a particular time duration in Logarithmic scale. Figure 6.2 is the Box Plot of the “Time duration” with a mean of **88.3788** minutes.

6.2 Conversion of Data

After the data are cleaned, they need to be converted into Time series data. Time series are data points in a sequence with their index being their time order. Time series is data are collected at regular intervals in time. These data points are made by taking

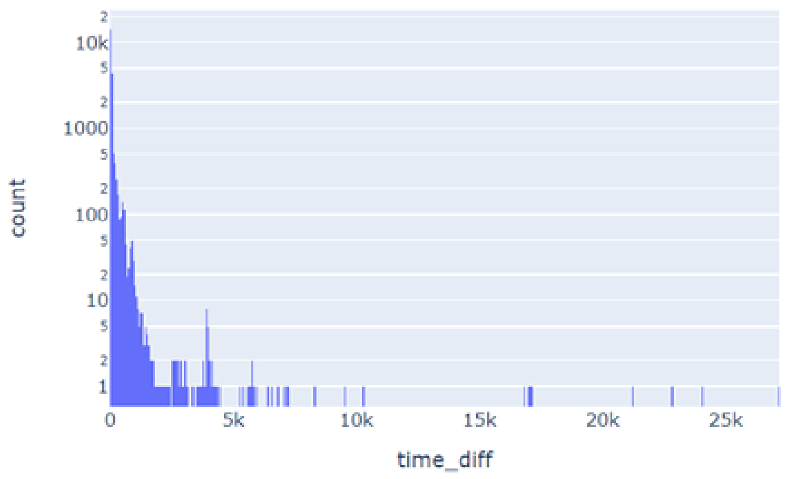


FIGURE 6.1: Time duration of charging events Histogram

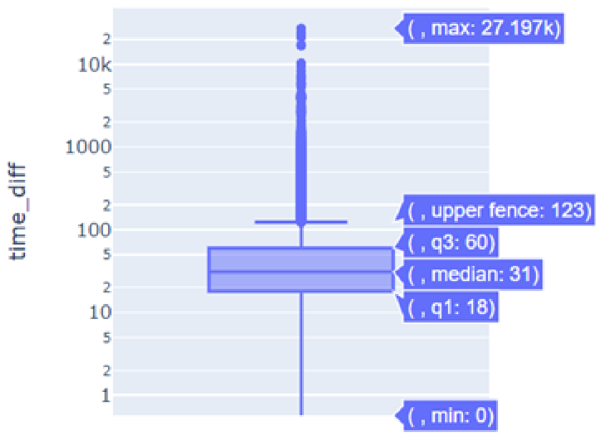
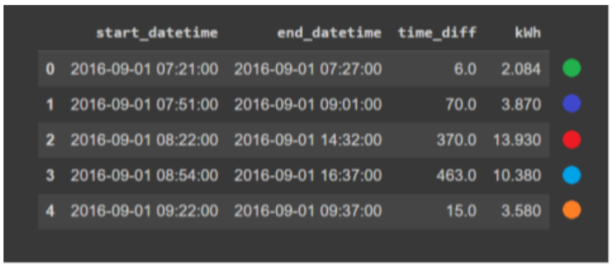


FIGURE 6.2: Time duration of charging events Box plot

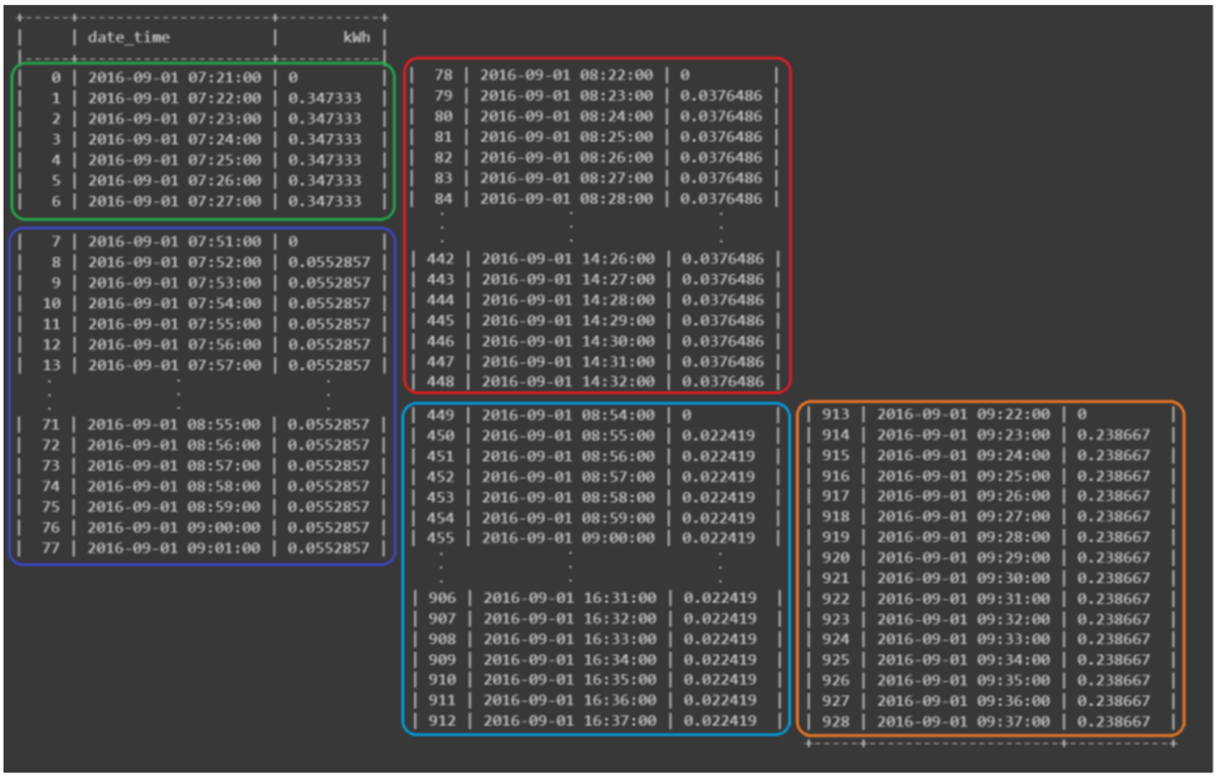
consecutive measurements using the same source over a time interval and represent the change over time of a quantity. In our case the source is the collection of all charging stations and the time interval between each observation is 1 minute. Therefore, we need to calculate the Total Energy consumption (kWh) of all charging stations for each minute in time between the time period 1st September 2016 to 31st August 2019. The following Figure shows the first 5 charging events of our data. The conversion to Time series will be demonstrated by using these charging events as examples.

In this small sample from the data the 5 charging events resulted in 928 time stamps when converted to time series. Each charging event from Figure 6.3 results in the time series data marked with the same colour box in Figure 6.4. For the first charging event the "kWh" is divided by the "time_diff". 2.084/ 6 results in 0.34733 kWh consumed at each minute between 07:21 and 07:27. The first minute in the period is set to 0 and the



	start_datetime	end_datetime	time_diff	kWh	
0	2016-09-01 07:21:00	2016-09-01 07:27:00	6.0	2.084	🟢
1	2016-09-01 07:51:00	2016-09-01 09:01:00	70.0	3.870	🔵
2	2016-09-01 08:22:00	2016-09-01 14:32:00	370.0	13.930	🔴
3	2016-09-01 08:54:00	2016-09-01 16:37:00	463.0	10.380	🔵
4	2016-09-01 09:22:00	2016-09-01 09:37:00	15.0	3.580	🟠

FIGURE 6.3: Dataframe before conversion of data to timeseries



```
+-----+---------------------+-----------+
|     | date_time           |       kWh |
|-----+---------------------+-----------|
|   0 | 2016-09-01 07:21:00 | 0         |
|   1 | 2016-09-01 07:22:00 | 0.347333  |
|   2 | 2016-09-01 07:23:00 | 0.347333  |
|   3 | 2016-09-01 07:24:00 | 0.347333  |
|   4 | 2016-09-01 07:25:00 | 0.347333  |
|   5 | 2016-09-01 07:26:00 | 0.347333  |
|   6 | 2016-09-01 07:27:00 | 0.347333  |
|   7 | 2016-09-01 07:51:00 | 0         |
|   8 | 2016-09-01 07:52:00 | 0.0552857 |
|   9 | 2016-09-01 07:53:00 | 0.0552857 |
|  10 | 2016-09-01 07:54:00 | 0.0552857 |
|  11 | 2016-09-01 07:55:00 | 0.0552857 |
|  12 | 2016-09-01 07:56:00 | 0.0552857 |
|  13 | 2016-09-01 07:57:00 | 0.0552857 |
    .           .                 .
    .           .                 .
    .           .                 .
|  71 | 2016-09-01 08:55:00 | 0.0552857 |
|  72 | 2016-09-01 08:56:00 | 0.0552857 |
|  73 | 2016-09-01 08:57:00 | 0.0552857 |
|  74 | 2016-09-01 08:58:00 | 0.0552857 |
|  75 | 2016-09-01 08:59:00 | 0.0552857 |
|  76 | 2016-09-01 09:00:00 | 0.0552857 |
|  77 | 2016-09-01 09:01:00 | 0.0552857 |

|  78 | 2016-09-01 08:22:00 | 0         |
|  79 | 2016-09-01 08:23:00 | 0.0376486 |
|  80 | 2016-09-01 08:24:00 | 0.0376486 |
|  81 | 2016-09-01 08:25:00 | 0.0376486 |
|  82 | 2016-09-01 08:26:00 | 0.0376486 |
|  83 | 2016-09-01 08:27:00 | 0.0376486 |
|  84 | 2016-09-01 08:28:00 | 0.0376486 |
    .           .                 .
    .           .                 .
    .           .                 .
| 442 | 2016-09-01 14:26:00 | 0.0376486 |
| 443 | 2016-09-01 14:27:00 | 0.0376486 |
| 444 | 2016-09-01 14:28:00 | 0.0376486 |
| 445 | 2016-09-01 14:29:00 | 0.0376486 |
| 446 | 2016-09-01 14:30:00 | 0.0376486 |
| 447 | 2016-09-01 14:31:00 | 0.0376486 |
| 448 | 2016-09-01 14:32:00 | 0.0376486 |
| 449 | 2016-09-01 08:54:00 | 0         |
| 450 | 2016-09-01 08:55:00 | 0.022419  |
| 451 | 2016-09-01 08:56:00 | 0.022419  |
| 452 | 2016-09-01 08:57:00 | 0.022419  |
| 453 | 2016-09-01 08:58:00 | 0.022419  |
| 454 | 2016-09-01 08:59:00 | 0.022419  |
| 455 | 2016-09-01 09:00:00 | 0.022419  |
    .           .                 .
    .           .                 .
    .           .                 .
| 906 | 2016-09-01 16:31:00 | 0.022419  |
| 907 | 2016-09-01 16:32:00 | 0.022419  |
| 908 | 2016-09-01 16:33:00 | 0.022419  |
| 909 | 2016-09-01 16:34:00 | 0.022419  |
| 910 | 2016-09-01 16:35:00 | 0.022419  |
| 911 | 2016-09-01 16:36:00 | 0.022419  |
| 912 | 2016-09-01 16:37:00 | 0.022419  |

| 913 | 2016-09-01 09:22:00 | 0         |
| 914 | 2016-09-01 09:23:00 | 0.238667  |
| 915 | 2016-09-01 09:24:00 | 0.238667  |
| 916 | 2016-09-01 09:25:00 | 0.238667  |
| 917 | 2016-09-01 09:26:00 | 0.238667  |
| 918 | 2016-09-01 09:27:00 | 0.238667  |
| 919 | 2016-09-01 09:28:00 | 0.238667  |
| 920 | 2016-09-01 09:29:00 | 0.238667  |
| 921 | 2016-09-01 09:30:00 | 0.238667  |
| 922 | 2016-09-01 09:31:00 | 0.238667  |
| 923 | 2016-09-01 09:32:00 | 0.238667  |
| 924 | 2016-09-01 09:33:00 | 0.238667  |
| 925 | 2016-09-01 09:34:00 | 0.238667  |
| 926 | 2016-09-01 09:35:00 | 0.238667  |
| 927 | 2016-09-01 09:36:00 | 0.238667  |
| 928 | 2016-09-01 09:37:00 | 0.238667  |
+-----+---------------------+-----------+
```

FIGURE 6.4: Dataframe after conversion of data to timeseries

rest with the 0.34733 kWh value. This procedure is repeated for all charging events, and this is how the rest of the time stamps shown in Figure y were produced as well. In this type of conversion to time series it is assumed that the car was uniformly charging from the time it was connected to the port to the time it was disconnected. This is a necessary assumption to make with our data in order to make the conversion, because it is unknown whether a car was not charging the whole time it was connected to the port.

The procedure described above was performed once with the data containing charging events from all the sites and once with the data with charging events that happened in Perth. Therefore, two Time series minutely datasets are produced.

The next step is to collect all unique "Datetime" values and sum up the Energy consumption for each, due to some charging events having overlapping charging periods.

The final step is to fill the dataframe with the missing times, because except from some charging events overlapping there are some others which are not adjacent to each other, meaning there is a time period in between. These minutes are set with zero kWh values since no charging occurred during these times. The result of this procedure is the complete conversion of Event based charging data to Time series data with minutely intervals.

Depending on the time interval we want our time series data to be we can use the pandas method dataframe.resample(freq) where freq is the code for the resampling frequency. Table 6.1 shows some frequency codes and the corresponding resampling frequency.

”freq” code	**Resampling frequency**
D	calendar day
H	Hourly
T	Minutely
15T	Every 15 minutes

Table 6.1: Frequency codes and their resampling frequency

A function was written to convert the Time series minute interval to hourly minute interval and was checked against the dataframe resample class to check whether the resampling using the class method was indeed performed as intended, which it was.

The resampling method is used once the time series data are loaded as a dataframe, if the model resolution needs to be Hourly. It is also used later for the plots in Section 7...

6.3 Data Insights

Box plots of the energy consumption of weekdays vs weekends were plotted for all months for all three years the data range from, shown in Figure 6.5 and Figure 6.6 . In 2016 all months, September to December have higher energy consumption on weekdays compared to weekends. In 2017 all months, January to December except February have higher energy consumption on weekdays compared to weekends. In 2019 all months, January to August have higher energy consumption on weekdays compared to weekends. It is evident that in weekdays there is more energy load consumed by the EVs charging at the stations. Box plots were created to compare the energy consumption for each season during the 4 years. In years 2017,2018 and 2019 were data were available for seasons winter, spring, summer, it is found that the energy consumption was highest during the summer and lowest during the winter. In 2016 and 2018 the consumption in autumn was higher compared to winter and in 2018 the autumn had the highest consumption compared to the other seasons in the same year. In 2017 however the consumption in autumn was the lowest. Overall it is deduced that during the dry seasons (spring and summer) the consumption is higher compared to the rainy season (autumn and winter).

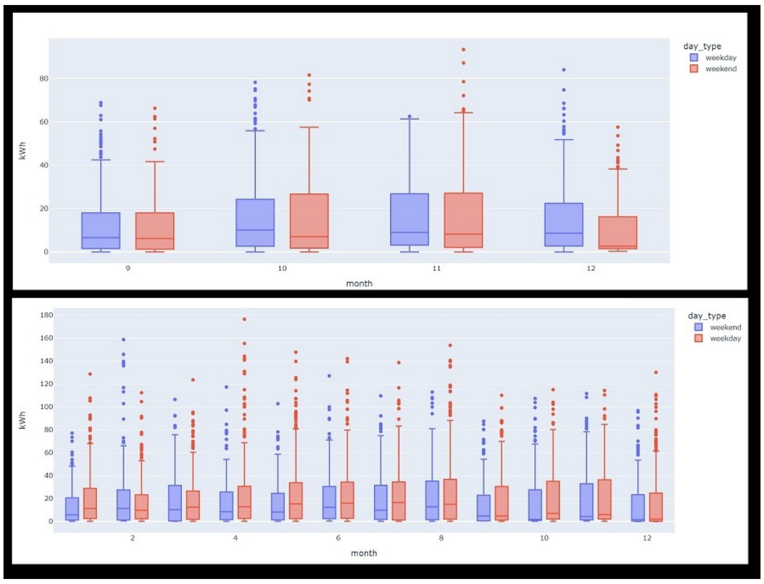


FIGURE 6.5: Week days vs Weekends consumption (2016-top, 2017-bottom)

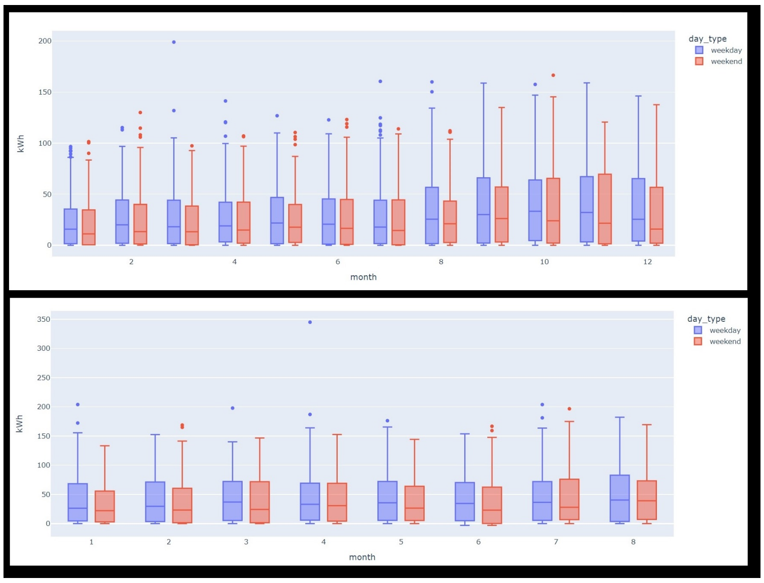


FIGURE 6.6: Week days vs Weekends consumption (2018-top, 2019-bottom)

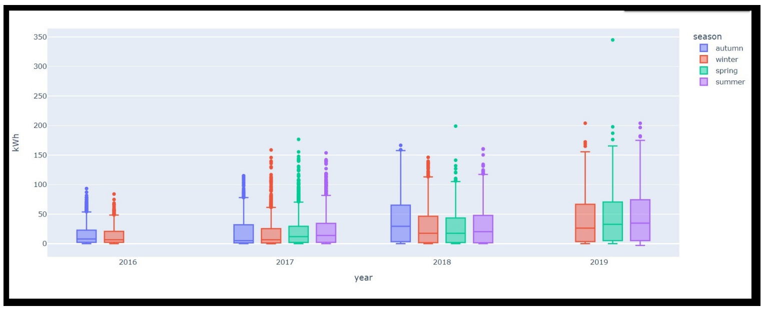


FIGURE 6.7: Energy consumption during each season

6.4 Data Transformation

Part of the general preparation of data in machine learning projects is data transformations. Data transformations include scaling and or standardization. The reason this step is performed is because ML models have better performance when data distributions are close to normal distributions and when the data comprise of several features, the features scales are similar. Scaling is when change the range of values of the data. In our 10-minute interval data, energy consumption has a range [0, 308.17] kWh and we can scale the range down to [0, 1] for example. Standardizing the data means we change the values so that the standard deviation equals 1. Usually together with standardization we perform mean centering which means changing the data values so that the mean of the data is equal to zero.

- Min – Max Scaler: Data are rescaled in a predefined range, usually between 0 and 1 using the following formula, $x_{scaled} = (x - x_{min})/(x_{max} - x_{min})$
- Max Absolute scaler: Data are rescaled to the range [-1, 1].
- Standard scaler: The standard scaler scales the data so that they have mean=0 and standard deviation=1. There is no pre-defined range in which the data are scaled to. $x_{scaled} = (x - \tilde{x})/\sigma$
- Robust Scaler: The robust scaler subtracts the median and scales the data according to the Interquartile range (IQR). The IQR ranges range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).
  $x_{scaled} = (x - x_{median})/(x_{75} - x_{25})$
- Power Transformer: The Scikit learn library contains the PowerTransformer() which corrects the skewness of the data distribution. The two options for the power transformer are the ‘Box-Cox’ and ‘Yeo-Johnson’ transforms.

The log-transformation requires non-negative and non-zero data. Our data are all non-negative since we are dealing with Energy consumption and the weather data which are non-negative as well. However they do have zero values. The Box-Cox transformation also requires non-zero data. All the data transformation techniques described above were tested and the data distribution for the Energy consumption was plotted for each one. As expected, the Standard Scaler, Min-Max Scaler, Max Absolute Scaler and Robust Scaler do not change the skewness of our data. The Log, Box-Cox and Yeo-Johnson do transform the data to multimodal distributions. However, in order to plot the Log and Box-Cox transformations the zero data were changed to the small positive value of 0.0001. It was decided it is not appropriate to alter any data to be able to transform them and feed them to our models. Hence, for the single-variate models, which use only the Energy consumption features the data are transformed using the Yeo-Johnson method. For the Multivariate models, a two step data transformation method was applied. The energy consumption data are transformed using the Yeo-Johnson method and then all the features (energy consumption + weather features) are transformed using the Min-Max scaler.

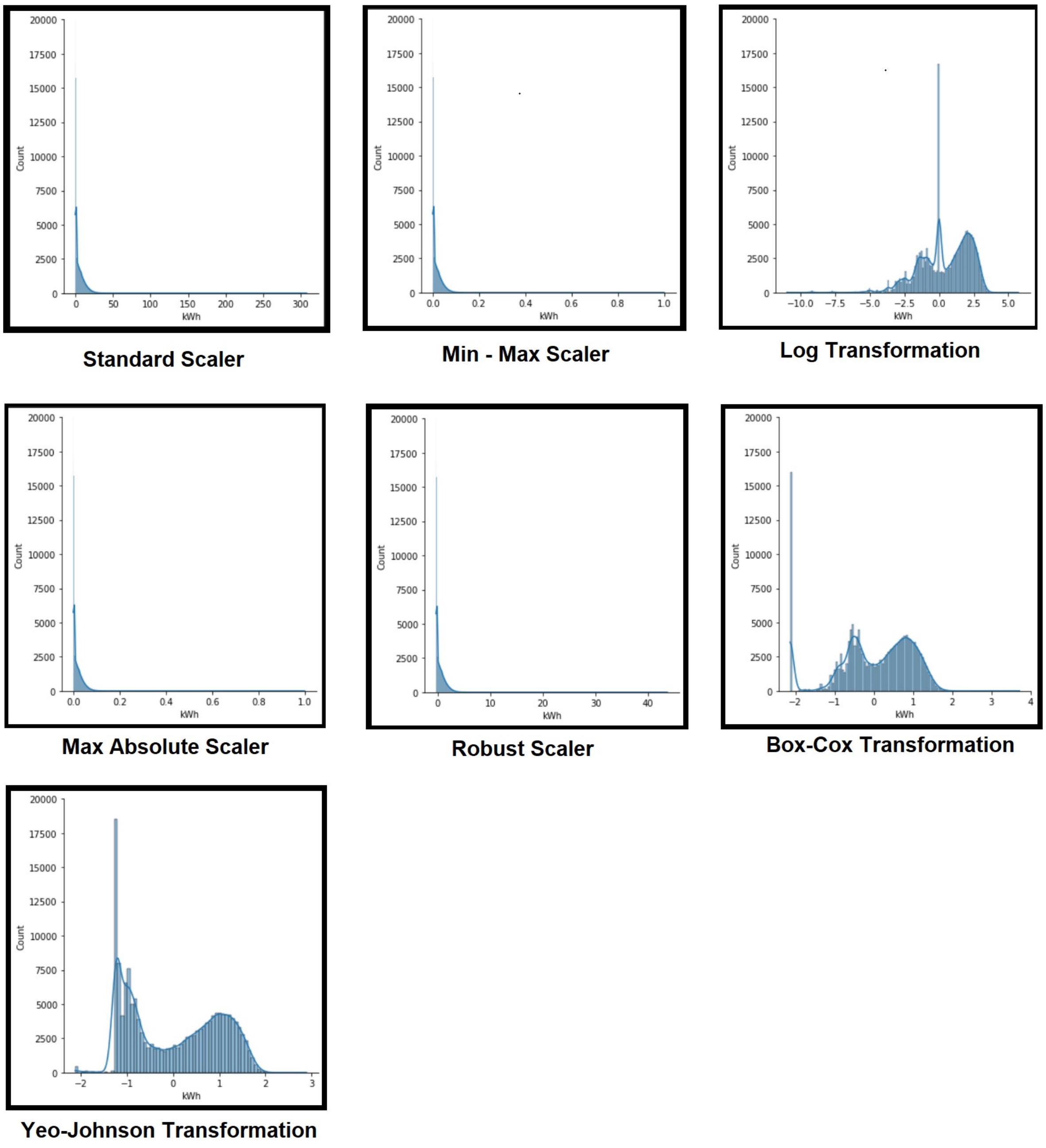


Figure 6.8: Data transformation techniques

6.5 SARIMA

For the SARIMA implementation the SARIMA model contained in the Statsmodel library is used. It has the following hyperparameters

- trend: A single parameter for specifying the type of trend. The trend can be constant, 'c', linear, 't', constant linear, 'ct' or can be no trend, 'n'.
- order: 3 parameters, p, q and t, which make up a tuple. The order is for modelling the trend.
- seasonal_order: 4 parameters, P, D, Q and m, which make up a tuple and it models the seasonality

A Grid search approach is taken where all the possible combinations of the parameters are trained and tested on models. The values used for the AR, MA and I components for the trend are all low value [0,1,2], while for the seasonality the values [0,12,24,48,168] are used. The values represent no seasonality, 12 hours, 1 day, 2 days and 7 days seasonality correspondingly. These values result in a total of 3140 models to be trained and tested. Some of the configurations are not valid in practice, and so are skipped during the initialisation of the model if they result in an error.

6.6 Transform Time series data to Supervised

In supervised learning there is an input variable X and an output variable Y and we try to find the function mapping X to Y using an algorithm.

$$Y = f(x)$$

Table 6.2 shows an example of supervised learning data, which contains 5 observations (each row is one observation) and there is an input variable X and an output variable Y, which is the variable that we want to predict

X	**Y**
0	-0.3
6	0.8
2	-0.5
1	0.1
4	-0.7

Table 6.2: Supervised learning data example

Table 6.3 shows an example of time series data. We want to represent the time series data into the supervised learning format, so we can use supervised learning techniques to devise an algorithm to predict the quantity at the next time step. Table 6.4 shows the data after being altered to be in supervised learning format. For each observation variable X is the quantity at time step t, and variable Y the observation at time step t+1.

Time	**Quantity**
1	10
2	5
3	25
4	20
5	70

Table 6.3: Timeseries data example

X	**Y**
?	10
10	5
5	25
25	20
20	70

Table 6.4: Timeseries data converted to supervised example

Table 6.5 shows an example of Multivariate time series data. The data are multivariate because we have two variables, Quantity 1 and Quantity 2. In this problem example we want to predict using the two input variables only Quantity 2. This problem type is a Multivariate Input, single-variate Output problem. The conversion to supervised learning data format looks like as depicted in Table 6.6.

Time	**Quantity 1**	**Quantity 2**
1	10	-0.5
2	5	-0.1
3	25	0.7
4	20	0.4
5	70	0

Table 6.5: Multivariate timeseries data example

X1	**X2**	**X3**	**Y**
?	?	10	-0.5
10	-0.5	5	-0.1
5	-0.1	25	0.7
25	0.7	20	0.4
20	0.4	70	0
70	0	?	?

Table 6.6: Multivariate timeseries data converted to supervised example

The examples 1,2,3 shown are all examples of Single-step forecast, since the prediction lies one step ahead, at time step t+1. Table 6.3 shows example of single-variate time series data and Table 6.7 the conversion to supervised data format where a two-step forecast is required. The number of time steps ahead to predict is called the sliding window and in this case the sliding window=2. The sliding window needs to be appropriately selected, to achieve good model performance and most likely it requires experimentation.

X	Y1	Y2
?	10	5
10	5	25
5	25	20
25	20	70
20	70	?
70	?	?

Table 6.7: Single-variate timeseries data converted to supervised, two-step output example

The model characteristics as previously shown in Table 5.1 have different number of variables, different number of Input steps and different number of Output steps. The number of variables is equivalent to having multiple quantities as in the example of Table 6.5. The number of input steps is how many past observation we are using for each prediction. It is equivalent to the number of X variables as in Table 6.6. For example from our models, the single-variate hourly model with input step = 4, will use the energy consumption of the hour t-1, t-2, t-3 and t-4 for each timestep t as input. The number of output steps are equivalent to the number of Y variables shown in the example of Table 6.7. From our models the hourly will be Single step, while 3 single-variate Multi-step models will be tested, with 15, 30 and 60 output steps correspondingly.

A function is written which converts the time series data into supervised format data. This function, named series_to_supervised. This function is universal for all the models created as the input steps and output steps are two of its parameters.

6.7 Conversion to Supervised for a Single-variate Multi-step LSTM model

The input to an LSTM model should have the shape [samples, timesteps, features]. For the Single-variate Multi-step model with input steps=15 and output steps=15, we will use the prior 15 minutes and predict the next 15 minutes. The
samples = number of instances of training set / 15. The timesteps = 15, one timesteps for each minute and features = 1, since the model is Single-variate and the variable is only the energy consumption. Therefore the training and validation sets are shaped as [length(training/val)/15, 15, 1]. The training data are prepared and changed shaped

to have overlapping windows, such that the training and val sets have shapes, X = [length(training/val), 15, 1] and Y = [length(training/val), 15]. For the multi-variate models the X shape will be X = [length(training/val), 15, n], where n are the number of features.

Table 6.8 and Table 6.9 demonstrate the change in shape of the training and validation sets.

Input	**Output**
$[t01, t02, t03, ..., t13, t14, t15]$	$[t16, t17, t18, ..., t28, t29, t30]$
$[t31, t32, t33, ..., t58, t59, t60]$	$[t61, t62, t63, ..., t73, t74, t75]$
...	...

TABLE 6.8: Initial training validation set shape

Input	**Output**
$[t01, t02, t03, ..., t13, t14, t15]$	$[t16, t17, t18, ..., t28, t29, t30]$
$[t02, t03, t04, ..., t14, t15, t16]$	$[t17, t18, t19, ..., t29, t30, t31]$
$[t03, t04, t05, ..., t15, t16, t17]$	$[t18, t19, t20, ..., t30, t31, t32]$
...	...

TABLE 6.9: Training validation set shape changed to overlapping windows

6.8 Train Validation - Test split

The data are split in Train-Validation and Test sets in a 85% - 15% fashion. The selected data transformations are applied to the Train-Validation data. The it is split into Train and Validation sets, such that the Train set accounts for 70% and Validation set for 15% of the whole data. After the split is made the timeseries Train and Validation sets are converted to supervised data using the procedure explained in Section 6.6 and Section 6.7.

6.9 Models Evaluation

6.9.1 Evaluation Metrics

In the multi-step problem, each forecast is made up of either 15, 30 or 60 values. The 1st value is the forecast for the energy consumption of the next minute, the 2nd value for the minute after that and so on, with the last value being the forecast at time [15, 30 or 60 minutes] after the current time. For the evaluation of multi-step models, the error will be calculated for each of the [15,30,60] time steps as well as the average of these values. In this way, we will be cable to quantify how well the model is performing at different lead times (+1min, +2min, etc). The metrics chosen to evaluate the model's performance are stated below along with their equations in Figure 6.9.

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}|$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y})^2$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y})^2}$$

FIGURE 6.9: Evaluation metrics equations (sourced from [1])

- **Root Mean Squared Error (RMSE):** The square root of the mean squared error, between the forecasted values and the actual values. Its advantage is that its unit is the same as the quantity being predicted which is helpful to conceptualize the performance [6].
- **Mean Absolute Error (MAE):** The average error of the absolute errors. The absolute errors is the absolute value of the difference of the predicted and true value. [3].

6.9.2 Walk Forward Validation

For evaluating the models, we use a technique called walk-forward validation. With time series data walk-forward validation is the most preferred solution to get most accurate results. If we want to make a prediction for the energy consumption in the next 10 minutes using our model our input is the previous 10 minute's consumption. For making a prediction for the 10 minutes after that, the actual data that were previously predicted now exist and so they are used for making the prediction. Table X below shows an example of how walk forward validation works in our problem.

6.10 Models

Some of the Neural Network model's architectures described below are plotted and depicted in Section B.

6.10.1 LSTM models

The LSTM model is has a single layer of 100 LSTM units, followed by a Dense layer of 50 neurons, which is followed by the output layer.

The Encoder-Decoder LSTM model has two layers comprised of 200 LSTM units. The first layer is the encoder and the other the decoder. The decoder is then connected to a fully connected layer, which is followed by the output layer. This encoder-decoder architecture differs from the standard LSTM when multi-step predictions are required. The output layer does not output all time steps as a single sequence, but outputs a single time step of the output sequence.

In the CNN-LSTM hybrid model, a CNN is used as the encoder and an LSTM as the decoder. The CNN has two convolutional layers with filters = 64 and kernel size =3. They are followed by a max pooling layer and then flattened so that the input to the decoder is a one dimensional vector. The decoder is the same as the decoder used in the Encoder-Decoder LSTM model.

6.10.2 CNN models

The CNN models take as inputs sequences of data which are one dimensional, from which features are extracted and learned. The convolutional layers of the network are therefore one dimensional, unlike the corresponding layers of CNNs that deal with images, which are 2 dimensional. The CNN input shape is the same as the LSTM, i.e [samples, features, time steps].

The Single-variate CNN is comprised of a convolutional layer with filter maps = 16 a and kernel size = 3, followed by a Max Pooling layer, then the NN dimensions are flattened, followed by Dense layer of 10 neurons and lastly the output layer.

The Multi-channel CNN is a model used with the Multivariate data. Each time series feature (Energy + 6 weather) is fed to the model via a separate channel. This means the CNN will use different kernel for each feature data sequence and different filter maps to learn from each feature independently. This model has two convolutional layers, with filter maps = 32 and kernel size = 3, followed by a Max pooling layer with pool size=2, followed by another convolutional layer with filters=16 and kernel size=3, followed by a Max Pooling layer with pool size=2. The dimensions are then flattened and connected to a Dense layer of 100 neurons which is lastly followed by the output layer.

The Multi-headed CNN model is also used with the Multivariate data. For each of the 7 features a different CNN model is defined. These CNNs have two convolutional layers with filters=32 and kernel size=3, followed by a max pooling layer with pool size =2. The dimensions of all 7 CNNs are then flattened and concatenated. These neurons are then connected to a dense layer of 200 neurons, followed by another dense layer of 100 neurons which is lastly followed by the output layer.

Chapter 7

Results and Discussion

7.1 SARIMA

The results of the best 3 SARIMA models are shown in Table 7.1. Due to large number of model configurations as explained in 6.5 the models where trained and tested only on a portion of the data.

(p,d,q)	(P, D, Q, m, t)	**RMSE**
(2 ,0 ,2)	(0, 1, 1, 12, 'c')	15.28
(2 ,0 ,2)	(0, 1, 1, 12, 'n')	15.32
(1, 0, 2)	(0, 1, 1, 12, 'n')	15.88

Table 7.1: SARIMA top 3 models configurations

7.2 Single-step models

Table 7.2 shows the results of the uni-variate and multi-variate models with characteristics, Hourly time resolution, Single-step output and data from Stations in Perth. For the LSTM models input steps [4,6,12,24,168], where tried except CNN-LSTM, which due to the kernel map sizes the smallest valid input sequence was 6. For the Multi-headed and channel CNNs, very small sequences were not tried due to their inferior results in the standard CNN. The input sequence lengths of 24 is equivalent to 1 day of information prior to the forecast , 168 is equivalent to 1 week of information and 336 2 weeks prior. The results indicate that for the uni-variate models, a model has the best performance when the input sequence is of length 24 hours, while with the multi-variate models the larger input sequence of 168 has the best performance. The different model types do not have a big difference in performance. For example the standard LSTM and standard CNN models with input sequence lengths of 24 hours only differ by 0.1

RMSE and MAE. It is deduced, thus the correct choice of input sequence length is of higher importance. The 24 and 168 input length models have better performance than the lower input sequence lengths because the data are periodical at 24 hours and giving less prior information to the model than 1 period is not sufficient for the model to learn. The results between uni-variate and multi-variate model also show that the additional features introduced to the model do not result in greater performance. The multi-variate predicted Energy consumption peaks is smoothed out significantly, looking very similar to sinusoidal wave with varying amplitudes depending on the actual Energy peaks of the day.

Variables	**Model**	**Input time steps**	**RMSE**	**MAE**
Uni-variate	LSTM	4	15.1	10.1
Uni-variate	LSTM	6	14.7	9.8
Uni-variate	LSTM	12	15.0	9.7
Uni-variate	LSTM	24	**13.1**	**8.9**
Uni-variate	LSTM	168	Nan	Nan
Uni-variate	Encoder Decoder LSTM	4	15.1	10.0
Uni-variate	Encoder Decoder LSTM	6	14.8	9.7
Uni-variate	Encoder Decoder LSTM	12	14.8	9.5
Uni-variate	Encoder Decoder LSTM	24	13.7	9
Uni-variate	Encoder Decoder LSTM	168	**13.4**	**8.7**
Uni-variate	Encoder Decoder CNN-LSTM	6	15.0	10.0
Uni-variate	Encoder Decoder CNN-LSTM	12	14.8	9.4
Uni-variate	Encoder Decoder CNN-LSTM	24	**13.4**	**8.8**
Uni-variate	Encoder Decoder CNN-LSTM	168	15.7	10.3
Uni-variate	CNN	4	15.4	10.4
Uni-variate	CNN	6	15.0	9.0
Uni-variate	CNN	12	14.9	9.6
Uni-variate	CNN	24	**13.2**	**8.8**
Uni-variate	CNN	168	13.2	9.0
Uni-variate	CNN	336	13.3	9.0
Multi-variate	Multi-Headed CNN	24	13.6	**8.7**
Multi-variate	Multi-Headed CNN	168	**13.4**	8.9
Multi-variate	Multi-Headed CNN	336	13.8	9.1
Multi-variate	Multi-Channel CNN	12	15.6	10.0
Multi-variate	Multi-Channel CNN	24	14.6	9.6
Multi-variate	Multi-Channel CNN	168	**14.0**	**9.2**
Multi-variate	Multi-Channel CNN	336	14.1	9.3

Table 7.2: Single-step output models with data of stations in Perth with Hourly time resolution

7.3 Multi-step models

Table 7.3 shows the results of the single-variate multi-output models with characteristics, Minutely time resolution and data from all the stations of the Council.

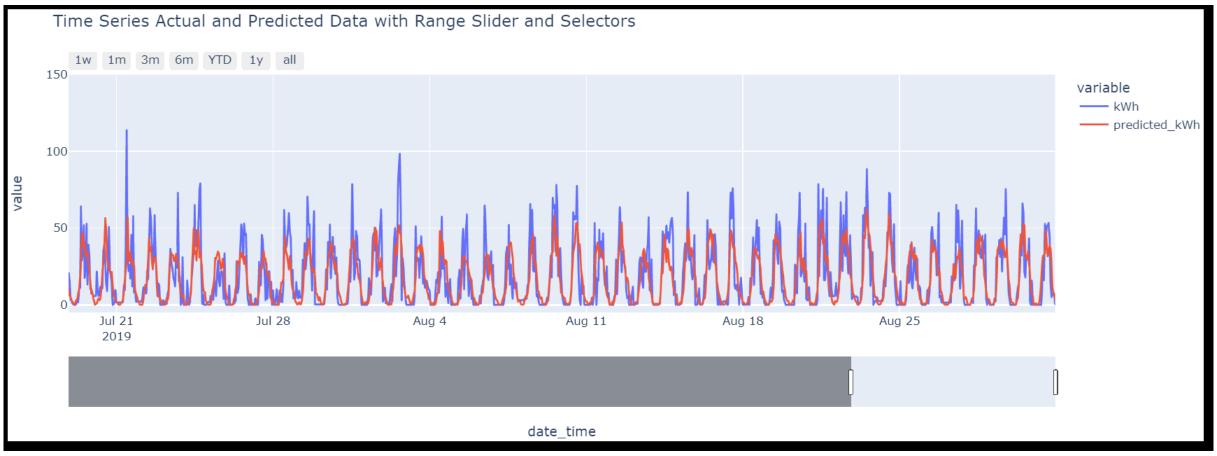


FIGURE 7.1: Single-variate CNN, input steps = 24

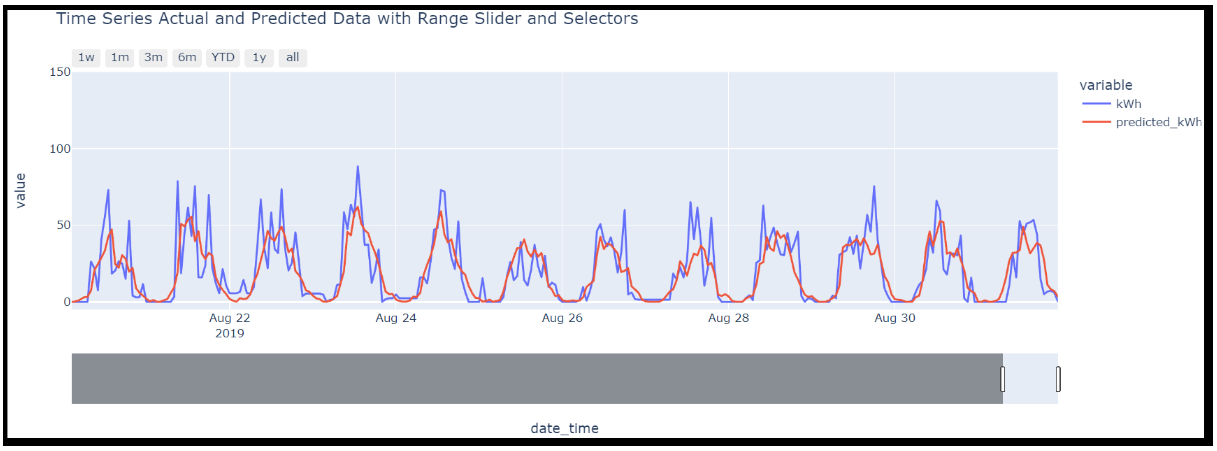


FIGURE 7.2: Single-variate CNN, input steps = 24, Zoomed in

The LSTM and Encoder Decoder LSTM models were trained with a small number of input steps and output steps = 15 minutes. They took a lot of time to train compared to the CNNs and since CNNs achieved competitive results against the LSTM models, it was decided to only train and evaluate CNN models with various output time steps and larger input sequences. The input time steps tested were [15, 30, 60, 120, 180, 1440], the values corresponding to 15, 30 minutes, 1, 2, 3 hours and 1 day of prior information to make the predictions. The output time steps were [15,30,60] meaning these input time steps were used for making predictions 15, 30 minutes and 1 hour ahead. If the output time step = 15 then RMSE and MAE are the average values of those 15 predictions. The corresponding average applied for 30 and 60 output values. For 15 output time steps the best performing model was the CNN with 60 input steps with regards to RMSE and with regards MAE the 30 input steps. For 30 output time steps the best performing model was the CNN with 120 input steps with regards to RMSE and with regards MAE the 30 input steps. For 60 output time steps the best performing model was the CNN with 1440 input steps with regards to both RMSE and MAE. It is deduced that the higher the number of output time steps, then the higher the number of input steps needed

Model	**Output time steps**	**Input time steps**	**RMSE**	**MAE**
LSTM	15	15	0.388	0.236
LSTM	15	30	0.383	0.232
LSTM	15	60	0.384	0.227
Encoder Decoder LSTM	15	15	0.395	0.237
Encoder Decoder LSTM	15	30	0.385	0.230
CNN	15	15	0.399	0.245
CNN	15	30	0.388	**0.229**
CNN	15	60	**0.379**	0.234
CNN	15	120	0.380	0.234
CNN	15	180	0.383	0.233
CNN	15	1140	0.391	0.246
CNN	30	30	0.488	**0.306**
CNN	30	60	0.487	0.307
CNN	30	120	**0.472**	0.310
CNN	30	180	0.482	0.310
CNN	30	1140	0.490	0.318
CNN	60	60	0.552	0.367
CNN	60	120	0.554	0.364
CNN	60	180	0.558	0.365
CNN	60	1440	**0.538**	**0.3555**

Table 7.3: Single-variate Multi-step output models with data of all stations and Minutely time resolution

for the model to perform the best. In other words the further ahead we are predicting the more prior information is required to make a good prediction. Models with lower number of input time steps than output time steps were not tested. For example a model with 60 output steps and 15 input steps is not tested. This model would get 15 minute of prior information to forecast the energy consumption 60 minutes ahead. Such model characteristics would clearly perform the worst in their class. As the number of output time steps increases the mean performance across all input steps decreases. This is because the further ahead a model is predicting the more errors are accumulated at later time steps. Table 7.4 shows the detailed prediction error at each time step of a 15 minute output model.

Performance	Time step 1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
RMSE	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.5	0.5
MAE	0.1	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
Average RMSE	**0.386**														
Average MAE	**0.234**														

Table 7.4: CNN model, input steps = 180, output steps= 15

To calculate RMSE and MAE shown in the Tables is for the minutely time resolution. To calculate the error metrics for the 15 time step performance, we multiply the number of output steps with the average minutely error. The results of this procedure as shown in Table 7.5

Timeframe	RMSE	MAE
15 minutes	5.69	3.51
30 minutes	14.16	9.30
60 minutes	32.38	21.33

TABLE 7.5: Total RMSE and MAE for all time steps of the intervals

These results show an improvement over the results of **??**, which produced result for the 15 and 30 minute time frames.

Figure 7.3 shows the true and predicted energy consumption of the last 10 days of the test set. The predictions were made using the best performing CNN model with 15 output steps. Figure 7.3 shows the true and predicted energy consumption of the last 10 days of the test set. The predictions were made using the best performing CNN model with 60 output steps. Both Plots are scaled at 15 minute time intervals. All the figures produced by these models can be found in C.

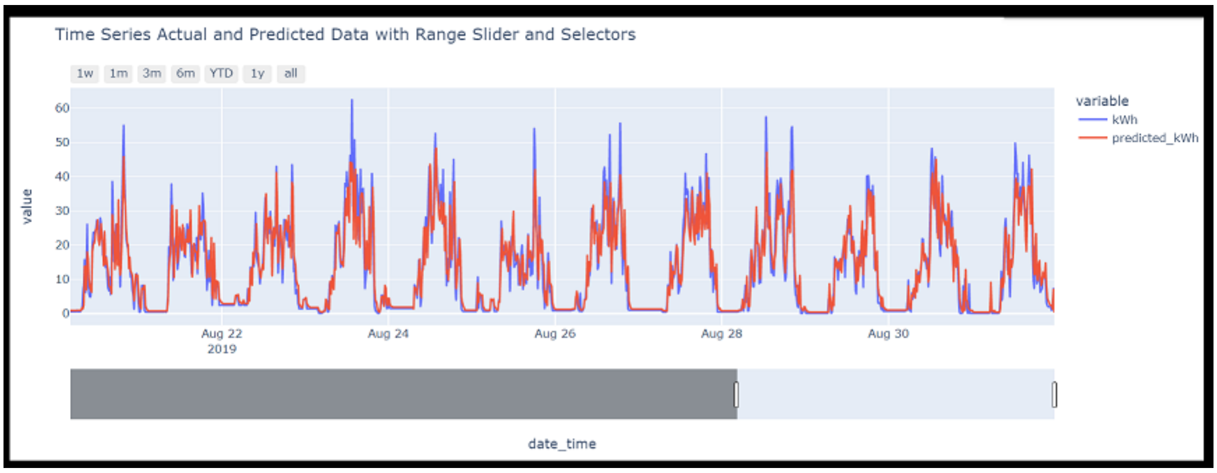


FIGURE 7.3: Multi-step CNN, output steps = 15, Zoomed in

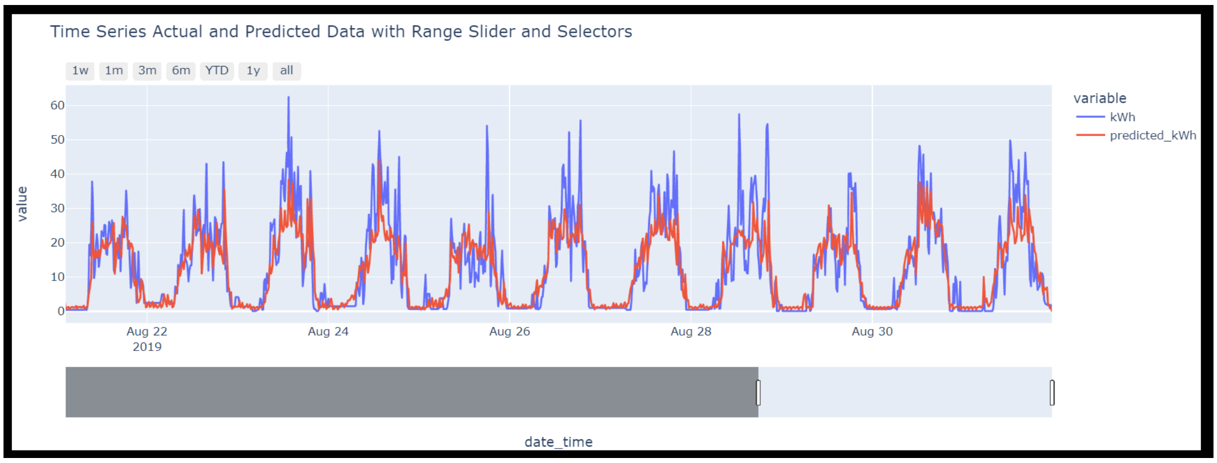


FIGURE 7.4: Multi-step CNN, output steps = 60, Zoomed in

Chapter 8

Conclusion and Future improvements

In this project it was demonstrated than for a complex problem such as the EV charging load forecasting Deep Learning techniques are promising and in particular multi-step predictions, achieving improved results over literature. Deep Learning models achieved superior performance to the statistical method SARIMA. It was found that 1 dimensional CNNs can work well with time series data and can be trained much faster than LSTM models. Multi-variate predictions, using weather features did not show improvements for 1 hour ahead predictions. It was found that using solely the previous Energy consumption is better method for predicting the peak values in a given day. However, further investigation is required to conclude that weather information is irrelevant to the energy consumption and a useless feature for a NN model to use. In this project, 6 weather features which were thought to be relevant were used with the multi-variate models. In the future, experiments should be conducted with each of the features individually and with combinations of a couple of features only to access the performance of the models. As of the current experiments though, for ultra-short term prediction the weather is not useful information to the models. For this reason experiments for other time frame predictions, such as medium and long term should be conducted.

Appendix A

Data Plots

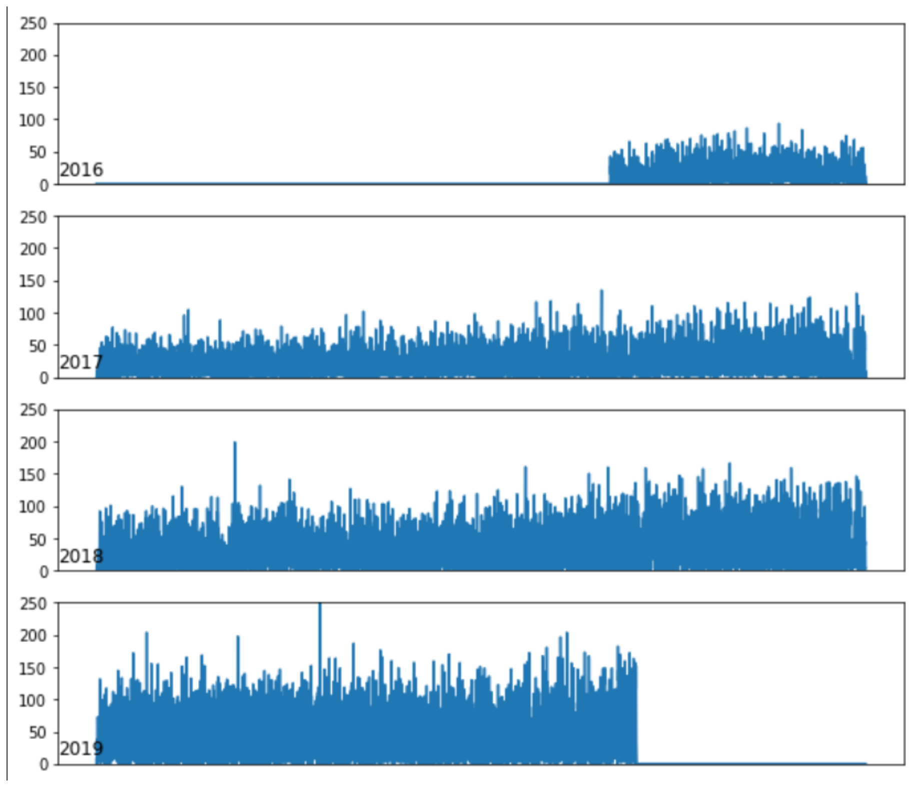


FIGURE A.1: Energy Consumption (all years)

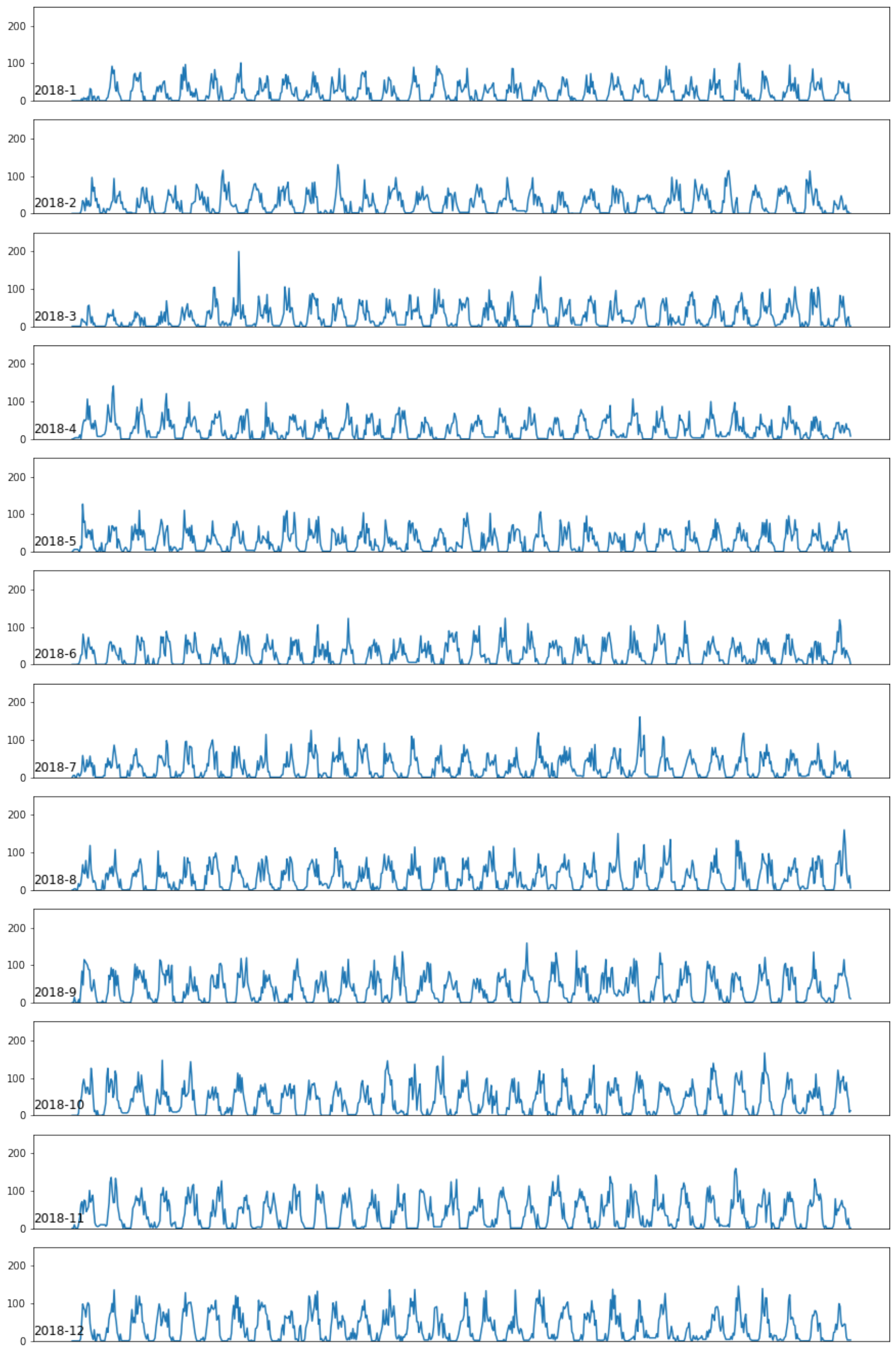


FIGURE A.2: Monthly energy consumption in 2018

Appendix B

Models

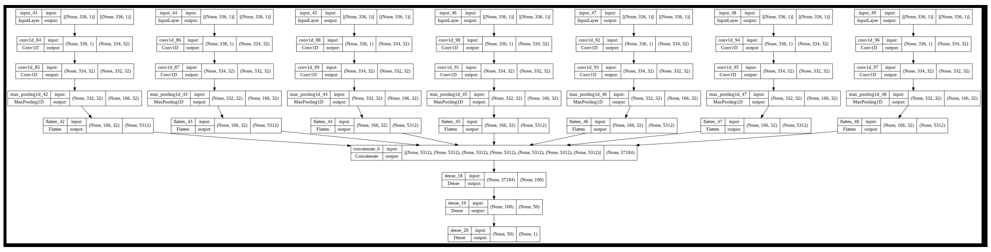


FIGURE B.1: Multi-Headed CNN, input length = 336

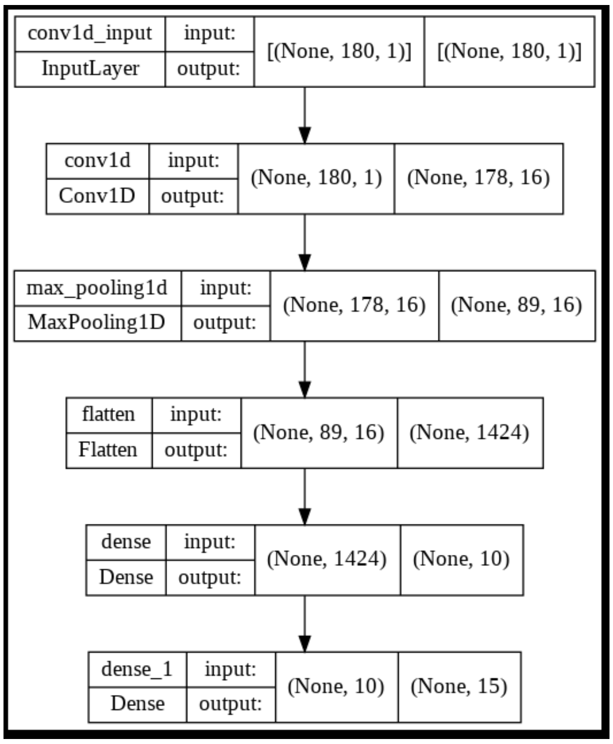
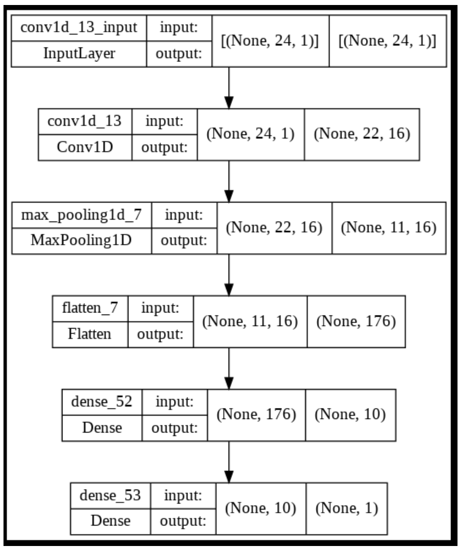
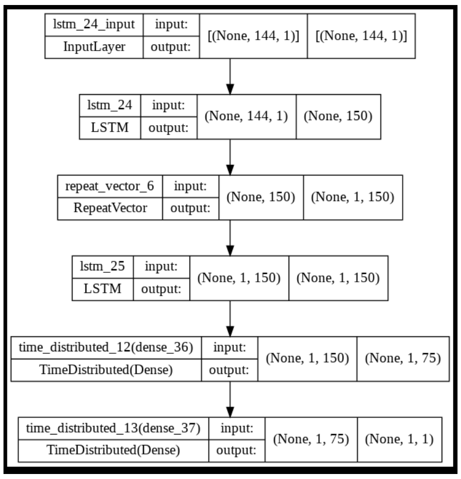


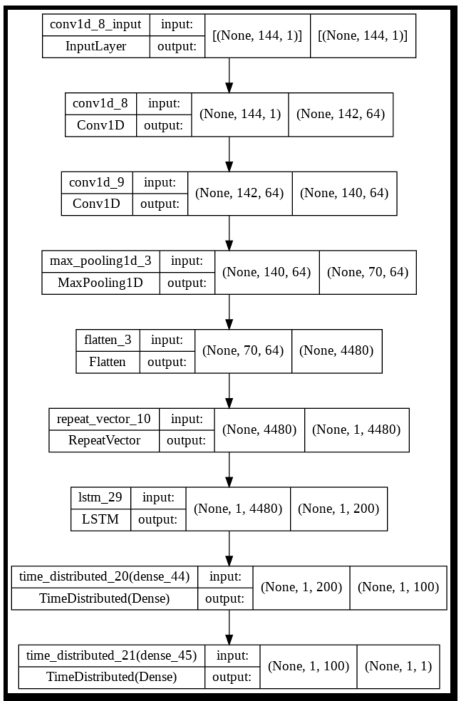
FIGURE B.2: CNN Multi-step, input length = 180, output length = 15


conv1d_13_input | InputLayer | input: | output: | [(None, 24, 1)] | [(None, 24, 1)]
conv1d_13 | Conv1D | input: | output: | (None, 24, 1) | (None, 22, 16)
max_pooling1d_7 | MaxPooling1D | input: | output: | (None, 22, 16) | (None, 11, 16)
flatten_7 | Flatten | input: | output: | (None, 11, 16) | (None, 176)
dense_52 | Dense | input: | output: | (None, 176) | (None, 10)
dense_53 | Dense | input: | output: | (None, 10) | (None, 1)

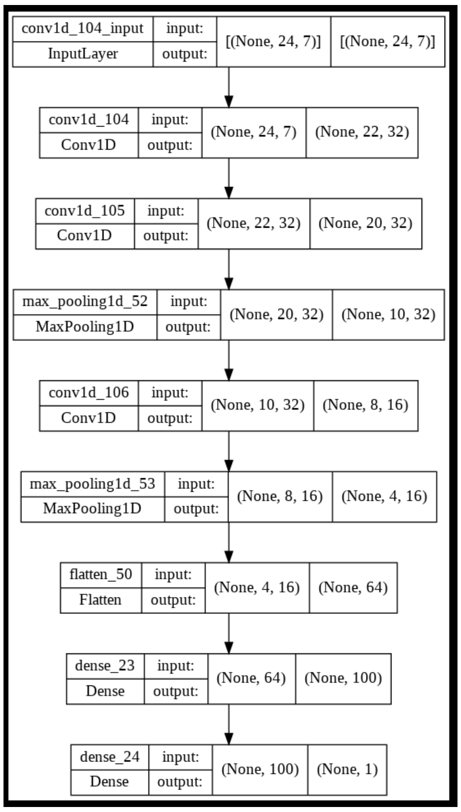
(a) Uni-variate CNN, input length=24


lstm_24_input | InputLayer | input: | output: | [(None, 144, 1)] | [(None, 144, 1)]
lstm_24 | LSTM | input: | output: | (None, 144, 1) | (None, 150)
repeat_vector_6 | RepeatVector | input: | output: | (None, 150) | (None, 1, 150)
lstm_25 | LSTM | input: | output: | (None, 1, 150) | (None, 1, 150)
time_distributed_12(dense_36) | TimeDistributed(Dense) | input: | output: | (None, 1, 150) | (None, 1, 75)
time_distributed_13(dense_37) | TimeDistributed(Dense) | input: | output: | (None, 1, 75) | (None, 1, 1)

(b) Encoder Decoder LSTM, input length = 144


conv1d_8_input | InputLayer | input: | output: | [(None, 144, 1)] | [(None, 144, 1)]
conv1d_8 | Conv1D | input: | output: | (None, 144, 1) | (None, 142, 64)
conv1d_9 | Conv1D | input: | output: | (None, 142, 64) | (None, 140, 64)
max_pooling1d_3 | MaxPooling1D | input: | output: | (None, 140, 64) | (None, 70, 64)
flatten_3 | Flatten | input: | output: | (None, 70, 64) | (None, 4480)
repeat_vector_10 | RepeatVector | input: | output: | (None, 4480) | (None, 1, 4480)
lstm_29 | LSTM | input: | output: | (None, 1, 4480) | (None, 1, 200)
time_distributed_20(dense_44) | TimeDistributed(Dense) | input: | output: | (None, 1, 200) | (None, 1, 100)
time_distributed_21(dense_45) | TimeDistributed(Dense) | input: | output: | (None, 1, 100) | (None, 1, 1)

(c) CNN, input length = 144


conv1d_104_input | InputLayer | input: | output: | [(None, 24, 7)] | [(None, 24, 7)]
conv1d_104 | Conv1D | input: | output: | (None, 24, 7) | (None, 22, 32)
conv1d_105 | Conv1D | input: | output: | (None, 22, 32) | (None, 20, 32)
max_pooling1d_52 | MaxPooling1D | input: | output: | (None, 20, 32) | (None, 10, 32)
conv1d_106 | Conv1D | input: | output: | (None, 10, 32) | (None, 8, 16)
max_pooling1d_53 | MaxPooling1D | input: | output: | (None, 8, 16) | (None, 4, 16)
flatten_50 | Flatten | input: | output: | (None, 4, 16) | (None, 64)
dense_23 | Dense | input: | output: | (None, 64) | (None, 100)
dense_24 | Dense | input: | output: | (None, 100) | (None, 1)

(d) Multi-channel CNN, input length = 24

Figure B.3: Model architectures plots

Appendix C

Forecasting Plots

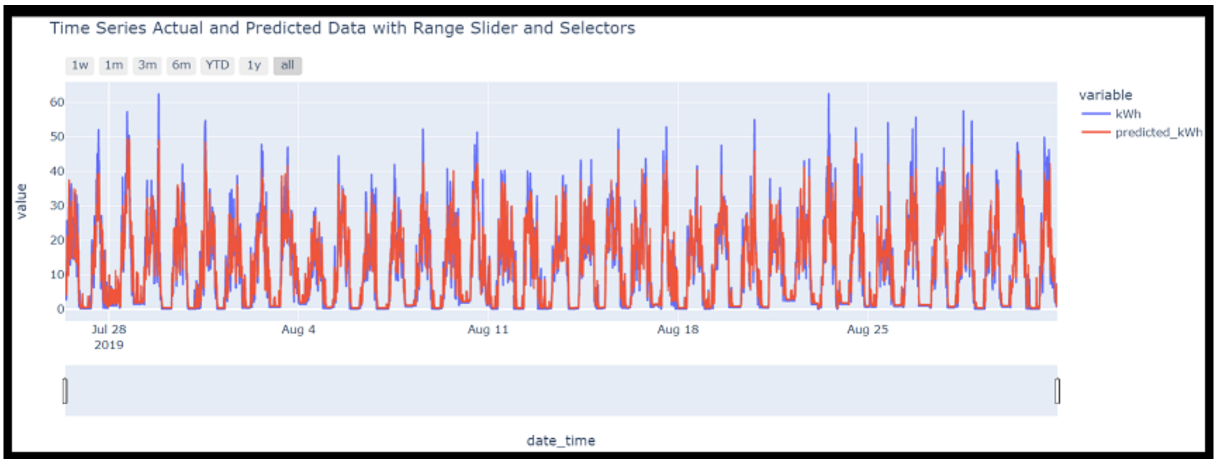


FIGURE C.1: Multi-step CNN, output steps = 15, all test set (15 minute resolution)

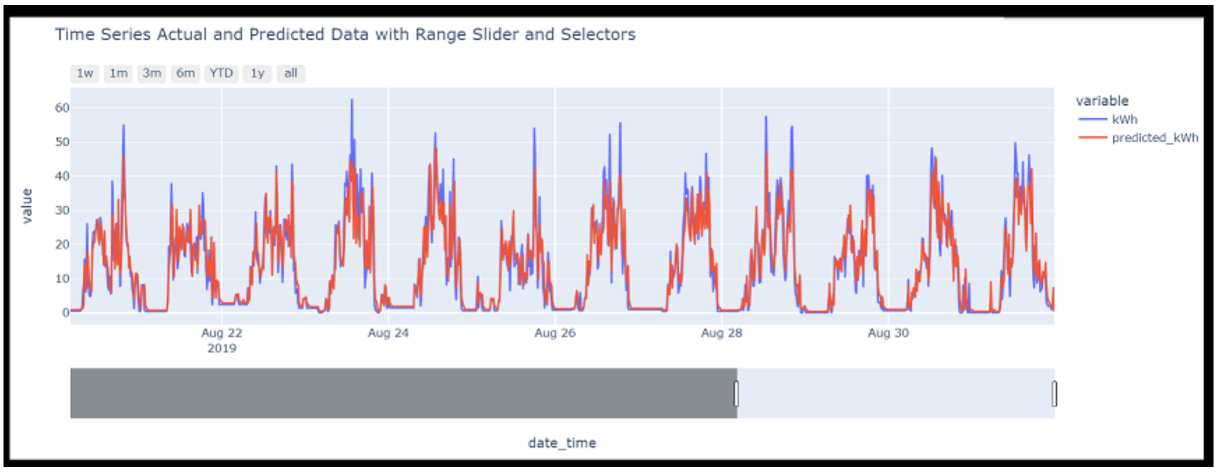


FIGURE C.2: Multi-step CNN, output steps = 15, last 10 days of test set (15 minute resolution)

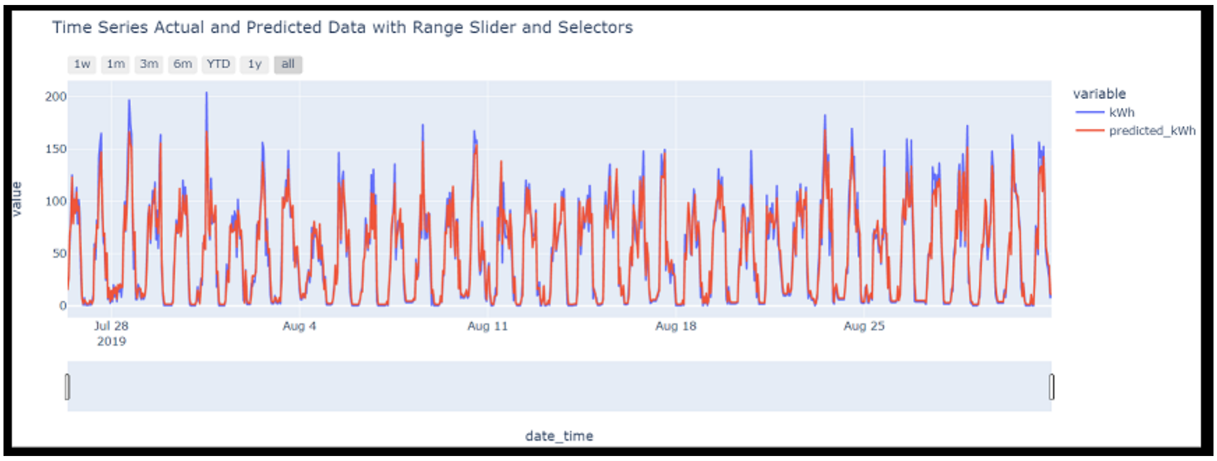


FIGURE C.3: Multi-step CNN, output steps = 15, all test set (1 hour resolution)

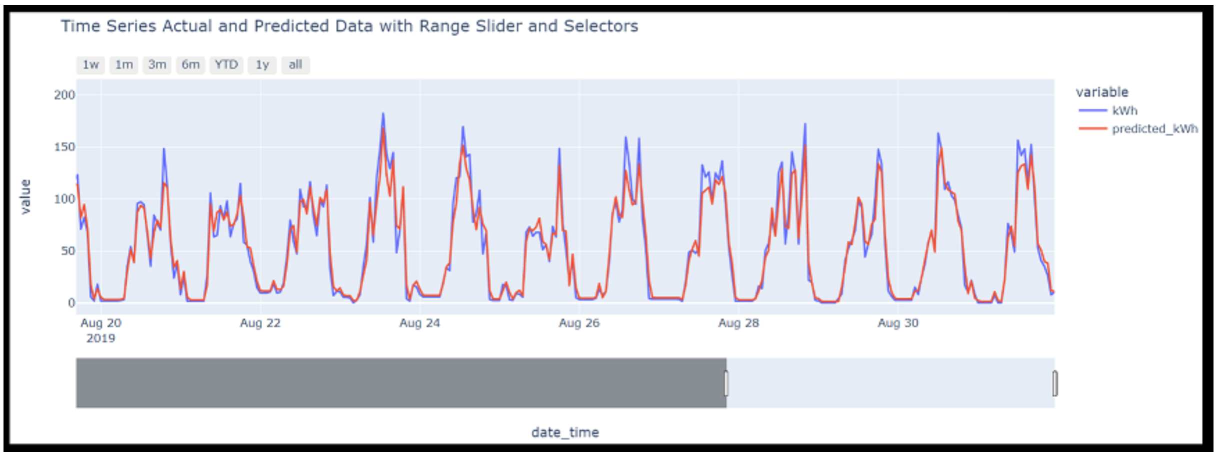


FIGURE C.4: Multi-step CNN, output steps = 15, last 10 days of test set (1 hour resolution)

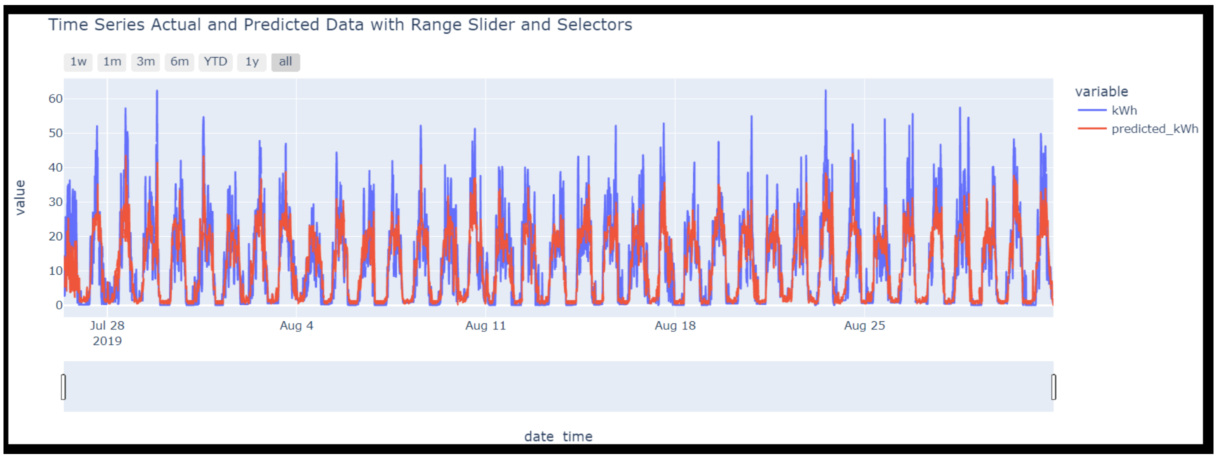


FIGURE C.5: Multi-step CNN, output steps = 60, all test set (15 minute resolution)

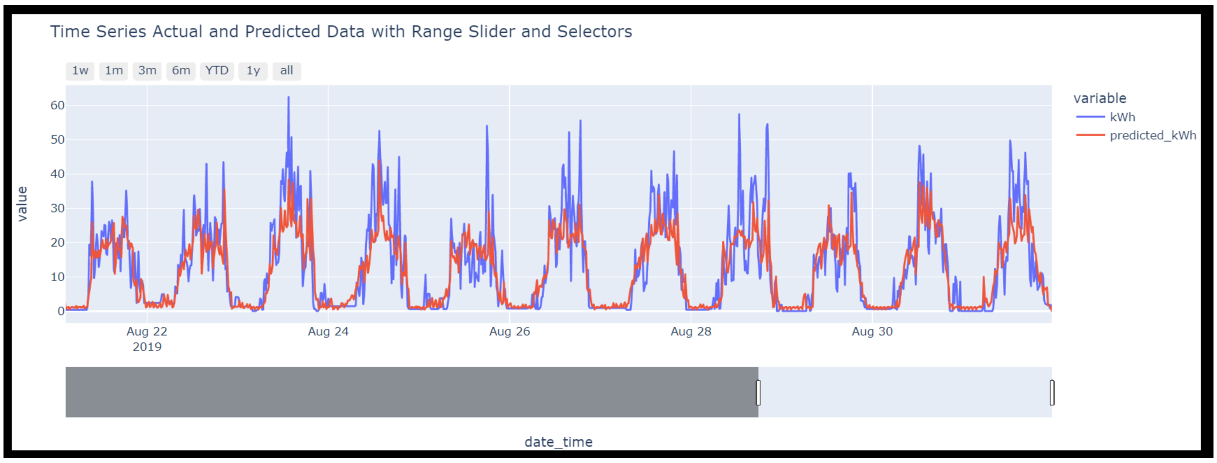


FIGURE C.6: Multi-step CNN, output steps = 60, last 10 days of test set (15 minute resolution)

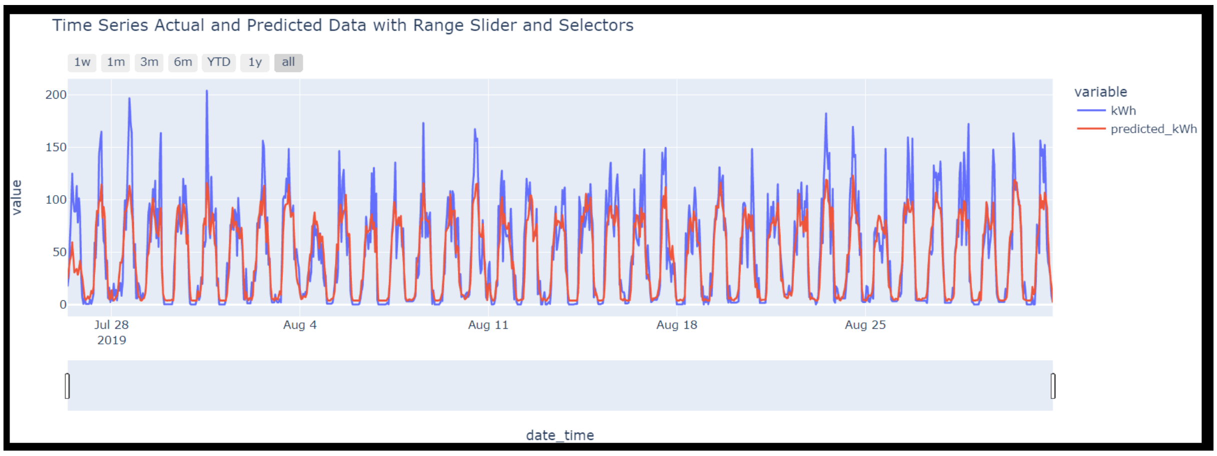


FIGURE C.7: Multi-step CNN, output steps = 60, all test set (1 hour resolution)

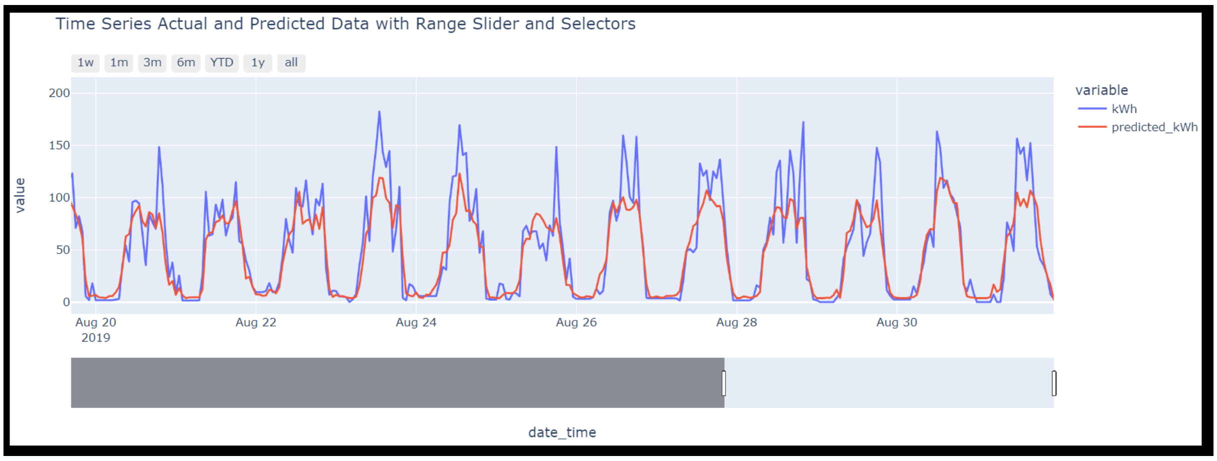
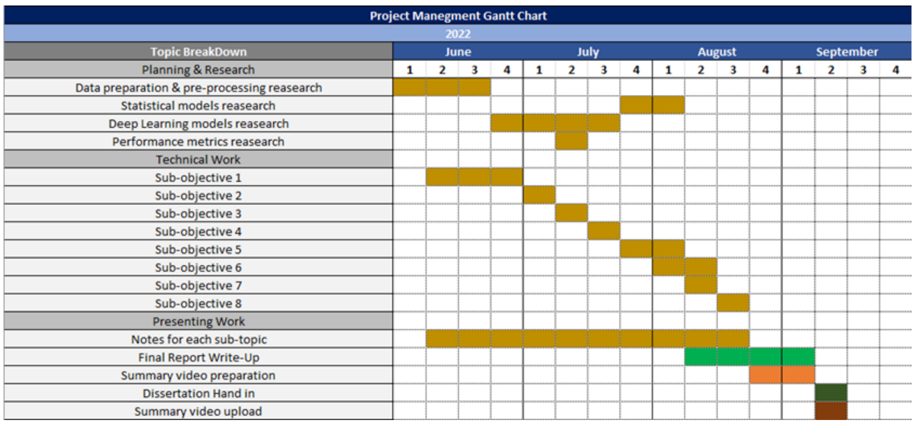


FIGURE C.8: Multi-step CNN, output steps = 15, last 10 days of test set (1 hour resolution)

Appendix D

Project Management

Figure D.1 shows the Gantt chart and Figure D.2 the explanation of the sub-objectives that appear in the Gantt chart. Both were presented in the Project Plan submitted in June 2022.



Project Manegment Gantt Chart

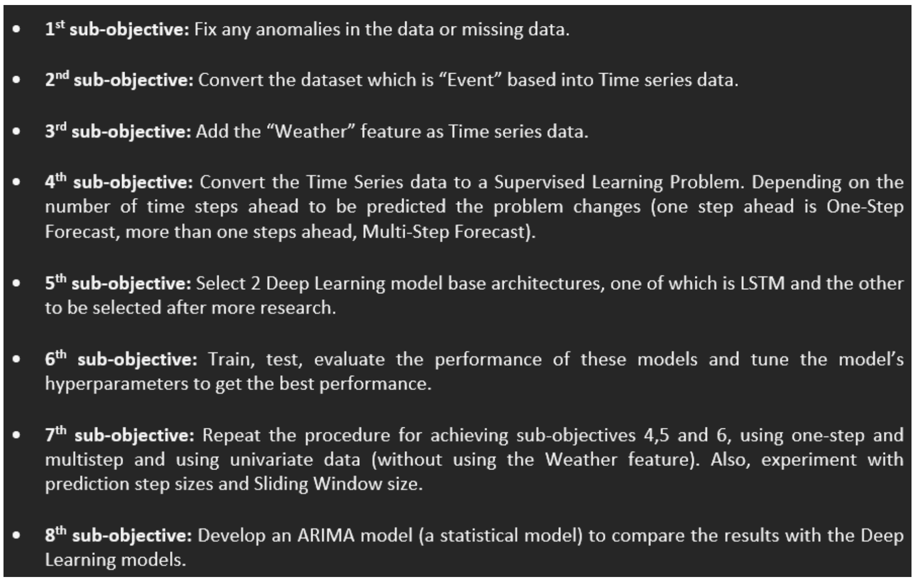
2022

Topic BreakDown	June				July				August				September			
Planning & Research	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Data preparation & pre-processing reasearch	🟨	🟨	🟨													
Statistical models reasearch								🟨	🟨							
Deep Learning models reasearch				🟨	🟨	🟨	🟨									
Performance metrics reasearch						🟨										
Technical Work																
Sub-objective 1		🟨	🟨	🟨												
Sub-objective 2					🟨											
Sub-objective 3						🟨										
Sub-objective 4							🟨									
Sub-objective 5								🟨	🟨							
Sub-objective 6									🟨	🟨						
Sub-objective 7										🟨						
Sub-objective 8											🟨					
Presenting Work																
Notes for each sub-topic		🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨					
Final Report Write-Up										🟩	🟩	🟩	🟩			
Summary video preparation												🟧	🟧			
Dissertation Hand in														🟩		
Summary video upload														🟫		

FIGURE D.1: Original project Gantt chart

As part of the project, all eight sub-objectives were completed as proposed. Sub-objective 1, took 1 week longer to complete as due to the dataset "date" and "time" feature being in incorrect format at random instances throughout the dataset which required detection and correcting. This procedure was explained in 6.1.1. For this reason the project was completed 1 week after he scheduled time. However, because the delay happened at the start of the project it was decided to start the write-up of the report 1 week earlier. This was possible, since throughout the project duration, notes were taken, results and figures produced saved.

Another small change to the project plan was the execution of the sub-objectives. Sub-objectives 1-4 had to be executed first and in order, which were. Then either sub-objectives 5-7 had to be executed,as proposed or the 8th sub-objective first and then 5-7. The 8th sub-objective was completed before the 5-7th. It was decided so because



- **1st sub-objective:** Fix any anomalies in the data or missing data.
- **2nd sub-objective:** Convert the dataset which is "Event" based into Time series data.
- **3rd sub-objective:** Add the "Weather" feature as Time series data.
- **4th sub-objective:** Convert the Time Series data to a Supervised Learning Problem. Depending on the number of time steps ahead to be predicted the problem changes (one step ahead is One-Step Forecast, more than one steps ahead, Multi-Step Forecast).
- **5th sub-objective:** Select 2 Deep Learning model base architectures, one of which is LSTM and the other to be selected after more research.
- **6th sub-objective:** Train, test, evaluate the performance of these models and tune the model's hyperparameters to get the best performance.
- **7th sub-objective:** Repeat the procedure for achieving sub-objectives 4,5 and 6, using one-step and multistep and using univariate data (without using the Weather feature). Also, experiment with prediction step sizes and Sliding Window size.
- **8th sub-objective:** Develop an ARIMA model (a statistical model) to compare the results with the Deep Learning models.

FIGURE D.2: Sub-objectives in Gantt Chart

the statistical method would act as a baseline model and made more sense to execute it before 5-7th. The time periods assigned to execute each sub-objective remained the same and completed as such (except the 1st, as explained above).

Bibliography

[1] A blog about data science and machine learning. regression model accuracy (mae, mse, rmse, r-squared) check in r, 2019.

[2] Evolutionary design of convolutional neural networks for human activity recognition in sensor-rich environments - scientific figure on researchgate, 2022.

[3] Statistics how to, 2022.

[4] https://medium.com/analytics-vidhya/encoder-decoder-seq2seq-models-clearly-explained-c34186fbf49b, sample year.

[5] Pranshu Aggarwal. Sarima using python – forecast seasonal data, 2019.

[6] Stephen Allwright. Rmse vs mape, which is the best regression metric, 2022.

[7] Goude Y. Massart P. Poggi J. Amara-Ouali, Y. and H. Yan. 2021. a review of electric vehicle load open data and models. energies, 14(8), p.2233.

[8] Jason Brownlee. Deep learning for time series forecasting, predict the future with mlps,cnns and lstms in python, 2018.

[9] C.Olah. “understanding lstm networks”, colah’s blog, 2015. [online]. available: https://colah.github.io/posts/2015-08-understanding-lstms/.

[10] DeepAI. ”convolutional neural network”, 2022.

[11] Y. Chang Y. Guo K. Zhu J. Zhu, Z. Yang and J. Zhang. A novel lstm based deep learning approach for multi-time scale electric vehicles charging load prediction. 2019 ieee innovative smart grid technologies - asia (isgt asia), 2019, pp. 3531-3536, doi: 10.1109/isgt-asia.2019.8881655.

[12] Spiliotis E. Makridakis, S. and V. Assimakopoulos. 2018. statistical and machine learning forecasting methods: Concerns and ways forward. plos one, 13(3), p.e0194889.

[13] Zap Map. ”map of electric charging points for electric cars uk: Zap-map”, 2022.

[14] Medium. ”a comprehensive guide to convolutional neural networks–the eli5 way”, 2022.

[15] Medium. ”encoder-decoder seq2seq models, clearly explained”, 2022.

[16] Viktor Mehandzhiyski. https://365datascience.com/tutorials/time-series-analysis-tutorials/autoregressive-model/, 2020.

[17] Alexander Orzechowski. Machine learning for public ev charging station forecasting, 2021.

[18] Paperswithcode.com. ”papers with code - max pooling explained”, 2022.

[19] Open Data Team. Electric vehicle charging points, perth and kinross council, 2017.

[20] Yugesh Verma. Analytics india magazine, ”using encoder-decoder lstm in univariate horizon style for time series modelling”, 2022.

[21] Worldweatheronline.com. ”14 day weather forecast — world weather online”, 2022.

[22] Maryam Ansari Alireza Zadeh, Pooya Joudaki. (2021). a survey on deep learning applications for electric vehicles in micro grids. 1-6. 10.1109/iot52625.2021.9469715.

[23] Juncheng Zhu, Zhile Yang, Yuanjun Guo, Jiankang Zhang, and Huikun Yang. Short-term load forecasting for electric vehicle charging stations based on deep learning approaches.applied sciences, 9(9), p.1723.

[24] Yang Z. Mourshed M. Guo Y. Zhou Y. Chang Y. Wei Y. Zhu, J. and 2019. Feng, S. Electric vehicle charging load forecasting: A comparative study of deep learning approaches. energies, 12(14), p.2692.