Sustainable & smart distribution networks

Thesis

EE BEP Group B

Subgroup B - Machine Learning and Forecasting

Daan van Dingstee
Ruben Eland

TU Delft

Sustainable & smart distribution networks

Thesis

by

EE BEP Group B

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended privately on Monday June 10, 2022 at 11:00 AM.

Student number:	1234567	
Project duration:	April 19, 2022 – June 10, 2022	
Thesis committee:	Dr. P. P. Vergara,	TU Delft, supervisor
	Dr. ir. R. Santbergen,	TU Delft
	N. Vahabzad,	Tu Delft, daily supervisor

TU Delft

Abstract

Increasing distributed generation of and demand for electrical energy results ever more in problems like congestion. Forecasting the demand, photovoltaic power generation and the number of electric vehicles connected to charging station for a residential neighbourhood can be an important part of a smarter distribution network for such a neighbourhood and can thereby increase the usage of renewable energy. In this thesis an overview of the design steps for making such a forecasting system is given and the steps are applied to a fictional dutch residential neighbourhood of the future. Some key findings are that for accurately forecasting the load, the temperature and time are the most important features. For accurately forecasting the PV power generation, especially the irradiation is most important, but time, horizontal view and humidity are also important features. Furthermore, it is shown that random forest regression models can accurately forecast both the demand and PV power generation with an accuracy above 90%. Artificial neural networks are also adequate models for the forecasting problems, but because they are harder to understand and not necessarily better, it is recommended to start with random forest regressors before making neural networks. Support vector machines seem less suitable for these particular forecasting problems.

Preface

In this thesis the build plans for a new residential neighbourhoord, Vechtrijk, are used as a basis for an imaginary neighbourhood of the future. All data is converted to fit in this imaginary neighbourhood with an abundance of PV systems, EV charging stations and electric heating inside the houses. While this imaginary neighbourhood often simply is called 'the Vechtrijk neighbourhood', in fact it is not connected to the real build plans and always the imaginary neighbourhood is meant. We thank the project developer, L. Roodbol from Blauwhoed, for sharing detailed maps from the build plans.

We also thank Ir. J. Koeners for providing data on PV power generation at the faculty of EEMCS.

Finally, we thank our supervisor, Dr. P. P. Vergara and our daily supervisor, N. Vahabzad for their continuing support.

EE BEP Group B
Delft, June 2022

Contents

Nomenclature

ANN	Artificial Neural Network
DG	Distributed Generation
DL	Deep Learning
DSM	Demand Side Management
DSO	Distribution System Operator
EV	Electric vehicle; in this article hybrid vehicles are not considered as EVs
LV	Low Voltage
ML	Machine learning
MLA	Maximum Likelihood Estimation
MSE	Mean squared error
PCA	Principal Component Analysis
PV	Photovoltaic
RFR	Random Forest Regressor
SVM	Support Vector Machine

1 Introduction

Tackling global warming is seen by many as one of the biggest challenges mankind faces in the 21st century. Humans cause too much greenhouse gas emission, causing sunlight to warm up the surface of the earth more through the greenhouse effect. This effect results in (possibly irreversible) changes in the climate worldwide, which could lead to catastrophic disasters. In the Netherlands, the residential sector produced 15.4 Tg CO2 equivalent (CO2 and other greenhouse gasses) in 2019 with heating, water heating and cooking, which accounts for 8.5% of the total CO2 equivalent emissions of the Netherlands in 2019. Fossil-fueled road transportation produced 29.6 Tg CO2 equivalent, which accounts for 16.4% of the total CO2 equivalent emission of the Netherlands in 2019 [1]. There are options like using electric cars, installing photovoltaic (PV) systems and electric heating and cooking to reduce the greenhouse gas emission in these sectors. However, these newly developed sustainable solutions have a drawback. The current low-voltage (LV) grid is not designed for their combined operation and the increasing penetration of PV systems, EV charging stations and electric heating and cooking could result in more problems like congestion and excessive fluctuations in voltage and frequency. In Amsterdam, congestion already is a big problem [2].

Allowing more energy-sustainable options to be implemented and connected to the grid while preventing problems like congestion will be a challenge for engineers in the upcoming years. Upgrades to the grid and other solutions to cope with this challenge are researched and proposed in this thesis. Following this general introduction, first follows a section that elaborates on the neighbourhood of the future in general. Then a neighbourhood that is being developed right now in the Netherlands is introduced, this neighbourhood will be used as example neighbourhood of the future throughout the thesis. Finally congestion is explained, as coping with congestion is one of the biggest challenges that the neighbourhood of the future will create.

1.1. Neighbourhood of the future

In the neighbourhood of the future, the previously stated residential and road transportation sectors will probably have shifted for a large part to all-electric solutions. The advantage of all-electric solutions is that the source of energy is interchangeable, meaning that fossil fuels can be replaced by sustainable energy sources with a dramatically lower emission of greenhouse gasses like wind or solar energy. In the Netherlands electrical cooking and heating is quickly rising in popularity; the amount of heat extracted from the air with heat pumps for the heating of residential buildings has more than doubled between 2018 and 2020 [3]. Also, the number of electric vehicles (EVs) in the Netherlands has approximately doubled every year from 2017 until 2021 [4]. Finally, the number of photovoltaic (PV) systems installed on residential buildings has also been steadily increasing. It is easily visible when walking outside anywhere in the Netherlands that a lot of households have PV systems on their roofs. While these solutions are effective in reducing our dependence on fossil fuels, they create new challenges for grid operators. First of all, because the grid is not designed for the high currents that will run through it, congestion will occur. Secondly, as part of the power generation is happening outside the control of the grid operators, it is harder to prevent problematic voltage and frequency fluctuations.

1.2. Vechtrijk neighbourhood

In order to design, forecast and optimize a sustainable distribution network, first a neighbourhood needs to be considered. A new developing neighbourhood was found in Weesp in the Netherlands. It consists of 37 total houses, a combination of detached houses, semi-detached houses, terraced houses and apartments, which can be seen in Figure 1.1. It also consists of a central parking area where electric vehicles might be parked and charged. We think this is a good representation of an all-encompassing neighbourhood in the Netherlands and thus a good neighbourhood to design a sustainable grid for [5].

Figure 1.1: The Vechtrijk neighbourhood ground plan

1.3. Congestion: Definition and issues

Grid congestion occurs when an overloaded grid prevents electricity from reaching the consumers. It can be illustrated as a kind of traffic jam, where the electrons in the wire are symbolized by the vehicles on a highway. During rush hour, too many vehicles are present and the flow of traffic may come to a halt. A similar situation occurs on power lines as demand or supply of power is in excess. The high current the power line has to carry can exceed the maximum capacity of the lines. This could eventually lead to power outages and costly repairs on power lines. In the Netherlands congestion is becoming a large issue. Figure 1.2 illustrates the significance of the problem. It is visible, that congestion due to high demand happens the most in urban areas such as the province of North Holland and around Leeuwarden. Furthermore, it is visible that congestion due to high generation happens more in rural areas such as the provinces Drenthe, Overijssel and Gelderland in the east of the Netherlands. In order to avoid power outages Distribution System Operators (DSOs) disconnect parts of the system during power excess. This necessary measure has great impact; generation systems are put on standby or consumers will have to seize their activities. Additionally, the connection to the grid for a newly installed system might take more than a year [6].

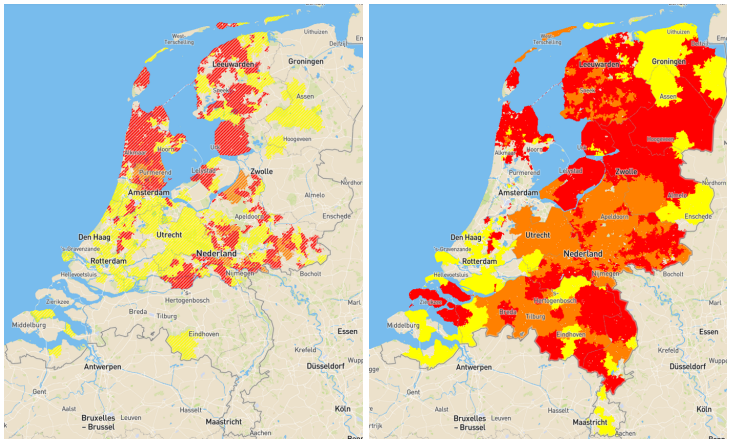


Figure 1.2: Consumption (left) and generation (right) congestion in The Netherlands [7]

1.4. Forecasting load and PV power generation

Knowledge of the demand for electrical energy and the amount of renewable power generated by households is crucial for knowing when problems like congestion in the low-voltage grid of a neighbourhood will occur, as overloads in either the demand or generation of power result in these problems. Thus, to be able to forecast when problems in the grid in a neighbourhood of the future will occur, both the demand and renewable energy generation should be accurately predicted. Forecasting the demand for electrical energy has been very important for the electric power industry for over a century [8]. Forecasting the demand for electrical energy is more frequently referred to as electric load forecasting and will in this thesis be addressed as load forecasting. Forecasting renewable power generation of course has become important only recently. While forecasting wind power generation and photovoltaic (PV) power generation both are studied extensively, PV power generation is assumed to be more important for preventing problems in the LV grid of a neighbourhood in the Netherlands, as wind turbines often are not directly connected to this grid.

1.4.1. Load forecasting

In the past, load forecasting was dominantly done on highly aggregated levels. The rise in distributed generation (DG) has partially shifted the focus to less aggregated levels (such as neighbourhoods). The increase in installed smart meters offers the required data that is sufficiently granular, both spatial and temporal, for accurate forecasts [8][9]. The two most occurring types of forecasts are point forecasts and probabilistic forecasts. While point forecasts are more mature, probabilistic forecasting has increasingly become a promising research area, especially for load forecasting on separate households or companies because of their highly irregular nature [8][9].

The field of load forecasting is often separated into four different stages based on the forecasting period. The stages are very short term (VSTLF), short term (STLF), medium term (MTLF) and long term load forecasting (LTLF). While the exact duration of the periods differs among studies, the cut-off between these stages in this thesis is taken at 1 day, 2 weeks and 3 years respectively, as in [8]. This can be seen in Figure 1.3. To prevent problems in the grid of a neighbourhood from an unexpected overload, especially VSTLF and STLF are interesting.

Machine Learning (ML) and Deep Learning (DL) are excellent tools for making predictions based on a lot of data, making them useful for load forecasting as well. The number of publications on using ML and DL for load forecasting has increased strongly in the last two decades [10][11]. When looking at the results of ML and DL models, case sensitivity should be kept in mind; which data is used to train the models, from which area, which climate and how accurate is that data? All these factors account for differences in trained models. Therefore comparing different models which are trained on different data is something that should be avoided. A study that compares differences between traditional ML models and ANNs on the same input data is [11]. It clearly shows that a multilayer perceptron (MLP) model, which is a feed-forward ANN with multiple layers, is the most accurate model of the 11 models which are compared. The root-mean-square-error (RMSE) of the MLP is 82% lower than the closest rival, the Gaussian process.

Some recent studies create a hybrid model that adds extra complexity to a DL model in order to improve the accuracy even more. Examples are adding an extreme learning machine (ELM) [12], controlled Gaussian mutation [13], a random forest [14] and there are of course many more hybrid models proposed. It is not easy to compare these hybrid models because of the case sensitivity which has been described earlier. A publication that compares different hybrid DL models has not been found.
Data pre-processing and feature selection are important steps in ML that often give insight into which features of the input data are more important for the prediction and how the data could be altered in order to get a more accurate model. Some useful pre-processing steps like outlier detection and how to distinguish between workdays and holidays are shown in [15]. Most studies use weather forecasts as well as past load and weather data as input to their models. The load will probably become even more dependent on weather data in the future, as most buildings will switch to electric heating and will get PV systems on their roof.
Peaks in the load could also come from EVs, as they charge at high power; usually 11 or 22 kW [**?**]. If every household were to have at least one car and all of those cars were electric, a significant increase in load would occur when all these vehicles were charged at the same time. Even when only 30% of all cars were electric, a 50% increase in demand peaks would occur [16]. Therefore forecasting the charging of EVs is also increasingly important for accurate load forecasting.

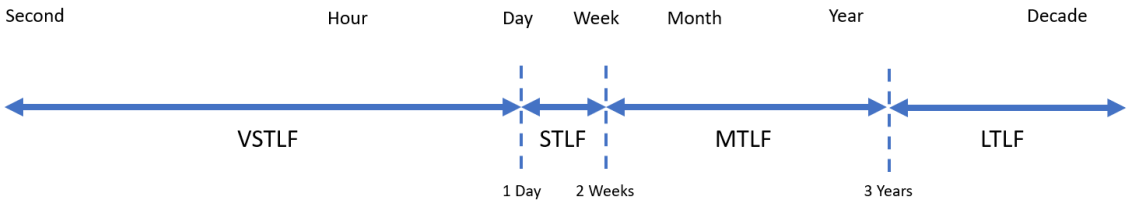


Figure 1.3: Distinction between VSTLF, STLF, MTLF and LTLF, obtained from [8].

1.4.2. PV power generation forecasting

Excess power generation by PV systems can also result in congestion. Besides congestion, the voltage can get too high, which can introduce other problems [17]. Forecasting the amount of renewable power that will be generated is thus very important for the design and optimization of a smart grid. In the paper by Huang [18] two different forecasting models are compared; a physical and a statistical model. The physical method consists of making a model of a PV system that takes the position of the sun throughout the year into account and uses the specifications of a PV system to predict the output power. The statistical method consists of making a Machine Learning model based on previous output power and weather data. Both methods use weather forecasts as input data. The article shows that both methods are quite similar in forecasting the PV power and that the size of the errors is mainly dependent on the accuracy of the weather forecasts. Besides this, the article by Pierro [19] concludes that the forecasting horizon is a lot less impactful on the accuracy than pre- and post-processing of the data. In this article an ANN is used. Intuitively, it seems that the statistical method is easier to implement than the physical method. This assumption is underlined by the fact that most recent studies implement the statistical method. Similar to load forecasting, in most recent studies Deep Learning is used more than traditional Machine Learning methods. [17] shows the accuracy of three different DL models; a convolutional neural network (CNN), a long short-term memory NN and a hybrid model that combines both. The RMSE is 1.563%, 1.393% and 1.434%, respectively for each model with one year of data. Furthermore, the paper shows that with their data training the models on more historic data results in a more accurate model up to 3 years. If more than 3 historic years of data are used, the models become less accurate. In [20] a model that takes the best-predicted pattern among the prediction results is used. These prediction results come from an adaptive neuro-fuzzy inference system, a multilayer perceptron ANN, and a radial basis function ANN to accurately forecast the load and weather data. This improves the accuracy significantly. The RMSE of the different models differ from 0.044% to 0.59%. [21] also uses an ANN but in combination with short long term memory as a base model to increase the accuracy of the ANN. The RMSE ranged between 1 and 2% depending on the case study. The article of Ehsan [22] shows that the multi-layer perceptron is a very accurate implementation of an ANN in forecasting PV power production. It has a RMSE of only 3.38%. Furthermore, the paper by Tiechui Yao [23] shows

that adding satellite images as input data for a hybrid ANN will result in a increase in the accuracy (RMSE) of up to 6.69% of the model.

1.5. Thesis subdivision

The design of a sustainable and smart distribution network is divided into three subgroups/subprojects. The Topology subgroup (A) will develop a new design of a residential distribution network for the neighbourhood which includes the selection of components in terms of size and capacity. The main goal is to withstand congestion and integrate PV systems and electrical charging points for at least each resident. The Forecasting subgroup (B, this group) will be making three forecasting models based on Machine Learning. The models will forecast the load, PV power generation and number of charging EVs of the neighbourhood and can be used to determine whether congestion or voltage issues will occur. It will use old load data of several households (updated with EV charging and electric heating data), old PV generation data, old weather data and weather forecasts as inputs to forecast the load and generation demand. The Optimization Control subgroup (C) will create a controller for the grid by creating an optimisation function for the power flow. Examples of the decision variables are the rate of charge for the EV batteries and the battery storage system.

1.6. Thesis outline

The literature review shows that Machine Learning is one of the most common methods to make accurate time-series forecasts for regression problems such as forecasting aggregated load and PV power generation. A lot of studies focus on getting the best accuracy possible by making complex hybrid Deep Learning models. This thesis chooses a different path; the aim is to make an overview of the design steps one has to make when creating ML models for forecasting data in sustainable and smart distribution networks of dutch neighbourhoods. The steps are applied to the Netherlands by using the build plans of the Vechtrijk neighbourhood as a basis and by using as much data from the Netherlands as possible. The importance of different features of the data for making accurate forecasts is analyzed and addressed. Simpler ML models such as a Random Forest Regressor as well as a more complex model, an Artificial Neural Network are built and their performances are compared. This thesis contributes to making electrical energy distribution systems more sustainable and smart in the Netherlands by showing how to make accurate forecasts of the demand and generation, something that is considered a crucial part of such systems.

2

Program of Requirements

2.1. Problem definition

As pointed out in the introduction, the low-voltage grid in the Netherlands is not designed for the increasing penetration of PV systems and all-electric systems in residential neighbourhoods which are required to reduce our dependency on fossil fuels. This can lead to congestion and problematic fluctuations in voltage and frequency. In the future, these problems will only get worse as most households should get all-electric heating systems, EVs and PV systems. In this thesis, only the problems in the low-voltage grid are addressed and not the problems in the medium- and high-voltage transmission lines. The assumption is made that in the Netherlands, wind turbines are rarely connected directly to the low-voltage grids in neighbourhoods and thus they fall outside the scope of this thesis. Knowing when the largest peaks in supply and demand will occur is a necessary step towards a smart and sustainable grid that is able to operate without problems in a neighbourhood of the future. Therefore three types of ML models will be made, which forecast the load, the number of charging EVs and the PV power generation for an imaginary neighbourhood of the future. Forecasts of the number of charging EVs do not contribute directly to less congestion, but it could be used by an optimisation program to use EVs as batteries if necessary. Random forests, support vector machines (SVMs) and ANNs will be used as state-of-the-art Machine Learning techniques for the forecasts and their accuracy will be compared with cross-validation. The forecasts could be used to find out when the load or generated power will exceed the boundaries of the grid and therefore result in problems, but this is deemed as outside the scope of this thesis. Grid operators can use the forecasts to effectively prevent problems, for example by efficient charging or discharging of batteries or a different optimization strategy. One of the limitations in the design process is that smart meter data is private information and protected by European law. This law was put in place to ensure the privacy of the customers where a smart meter was installed. Unfortunately for this thesis therefore there only is a small and dated (2013) data set available for training the models. Below the requirements of the system following the MoSCoW method are listed [24].

2.2. Must Have

1. The system must forecast PV power generation based on historical weather data, weather forecasts and historical PV power generation data.
2. The system must forecast the load based on historical weather data, weather forecasts and historical load data.
3. The system must forecast the number of charging EVs based on historical EV charging data.
4. Different models like Random Forest, SVM and ANN models must be created in order to find the most accurate model.
5. The best model for each forecasting problem must have an accuracy of at least 70% at all times.
6. The data set must be updated to represent a neighbourhood of the future.

7. The system must forecast the load, PV power generation and number of charging vehicles for the whole neighbourhood aggregated.
8. All load data used for forecasting must be kept anonymous.
9. The system must be applied to a residential neighbourhood.

2.3. Should have

1. The best model for each forecasting problem should have an accuracy of at least 80% at all times.
2. The system should distinguish weekdays and workdays when forecasting the load and number of charging EVs.

2.4. Could have

1. The forecasting models could be more complex neural networks such as convolutional neural networks or recurrent neural networks.
2. The system could estimate the state of charge of EVs or batteries used for charging-discharging solutions.
3. The best model for each forecasting problem could have an accuracy of at least 90% at all times.
4. The system could forecast per type of house (detached, semi-detached, terraced or apartments).
5. The forecasts could be used to find out when the load or generated power will exceed boundaries of the grid and therefore result in problems.
6. The models could use recent data (the past few hours) as input for making decisions.

2.5. Will not have

1. Wind power generation will not be forecasted.
2. The power generation of any other power source than PV systems will not be forecasted.
3. The optimal strategy of lowering overall load will not be calculated.
4. Besides residential users, the system will not take other kinds of low-voltage users, such as companies, into account.

3

Machine Learning Models

Three Machine Learning forecasting models will be created; one for the aggregated load of the neighbourhood, one for the aggregated PV power generation and one for the number of EVs that are being charged. First, a general introduction to ML is given. Then, an overview of the models and their input data is shown, accompanied by design choices and explanations. Finally, a detailed description of all types of ML models that are used for forecasting problems is given.

3.1. Machine Learning

Machine Learning is a broad research area that is about understanding data and recognizing patterns. In Machine Learning there is a distinction between supervised learning and unsupervised learning. Supervised learning involves building a statistical model for predicting one specific, known output based on one or more inputs. With unsupervised learning, there is no specific output, but relationships in the input data can still be learned and used to generate outputs. Supervised learning is further divided into regression and classification. Regression is about predicting a continuous or quantitative output value, while with classification a classified or qualitative output is predicted [25].

All three forecasting problems can be interpreted as supervised regression problems because for each problem the output is one specific continuous value. ML models are an excellent tool for making forecasts. First of all, because they are flexible, i.e. it is easy to use different input data and quickly train new models. Furthermore, most ML models are capable of handling large amounts of data and as short-term forecasting has relatively small intervals, there is a lot of data. Also, the fact that most recent studies use ML for load and PV power forecasting, as stated in the literature review, shows that it is a good tool for making these forecasts. Therefore all forecasts are made with ML models. In the literature review, it was found that ANN models could be adequate models for the specified forecasting models. However, simpler models will be built as well and their performance is compared with the performance of the ANN model.

$$R^2 = 1 - \frac{Var(y)}{Var(y_{predict})} \tag{3.1}$$

$$MSE = \sum_{i=1}^{n} (y_i - y_{predict,i})^2 \tag{3.2}$$

Before models are built, the data is separated into the output of the model, often called y, and the features, often called x. The model has to make its predictions for y based on the features, x. The predicted value creates a curve which is called a model fit. After this separation, the data from both x and y is separated into a train set and a test set. Typical distributions are 70/30 or 80/20. Because there is sufficient data, here a 70/30 division is chosen for all models. When training models, one

should always be cautious about overfitting the model. Overfitting means that a model is trained 'too well', i.e. the training error is made so small, that the model fit does not follow the trend of y anymore, but instead it makes strange looking bends in order to get closer to the values of y. A good strategy to prevent overfitting is cross-validation. In this thesis, k-fold cross-fitting is applied. K-fold cross-validation means splitting the training data in k folds and finding the best parameters of the model for each fold. The parameters are then compared and the best overall parameters are chosen [25]. Measures for performance are the mean-squared-error (MSE) and the R^2 score. The MSE simply is the mean of all errors between predicted y and actual y, squared. R^2 shows how much the model prediction explains the variance of the real y and therefore can be used as a measure of how good a model fit is (There are exceptions, f.e. a low R^2 does not always mean that a model fit is bad, but such exceptions are deemed to detailed for this thesis). The R^2 score is easier to understand because it is relative to the data, so it is used to compare the performance of different models. See Equation 3.1 and Equation 3.2 to find out how the R^2 and MSE are computed.

3.2. Overview models

When looking at Figure 3.1, it is visible that weather forecasts are inputs of the PV and load models. Especially the PV forecasts are expected to be mainly determined by the weather forecast features, such as irradiation, cloud cover and temperature. But also for the load data weather forecasts are important, since it is assumed that in a neighbourhood of the future all houses will have electric heating, so cold days will result in higher loads. Since the EV data set does not consist of real measurements, but generated data based on user profiles, it is not possible to use weather data as an input to the EV forecasting model. The time of the measurement is an input of all models, but it is expected to be more important for forecasting the load and number of charging EVs, as both are lower at night and higher during the day. Time also could be an important input to the PV forecasts, as the power generation of PV panels is zero at night and nonzero during the day, but it is expected that weather features such as irradiation are more important. Finally, the type of the day; whether it is a workday or a holiday - all Saturdays and Sundays are also added to the holiday list - is an input of load and EV models and is expected to increase the accuracy somewhat as the load and number of charging EV curves are different for work- and holidays. All expectations are tested and discussed in chapter 5.

See Table 3.1 for an overview of all features used by the models. In total 22 features were given in the KNMI data set, however, some were left out because they were expected to have little importance for the forecasts and working with all the features dramatically extended the run times. For example, the air pressure, wind direction and whether weather data was automatically or manually registered, all should be irrelevant features for the models.

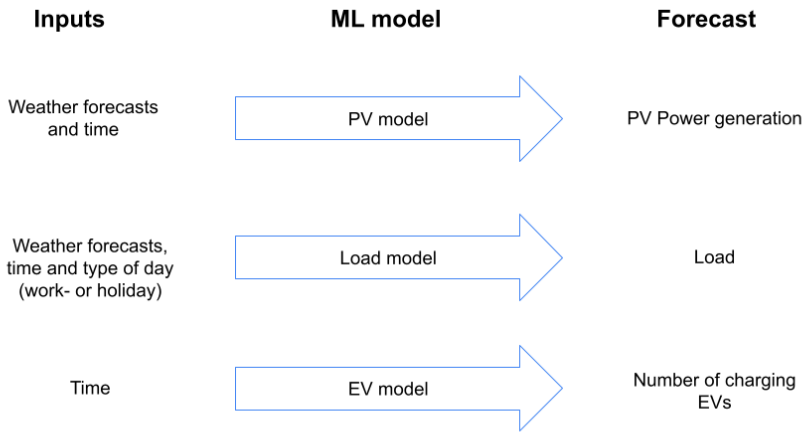


Figure 3.1: Inputs and outputs of the three ML models

Load model features	PV model features	EV model features
Time [integers: 0015; 0030 etc]	Time [int: 0015; 0030, etc.]	Time [0015; 0030, etc.]
Workday [0 = holiday, 1 = workday]	Season [0 = winter, 1 = spring, etc.]	-
Wind speed [0.1 m/s]	Wind speed [0.1 m/s]	-
Temperature [0.1 ° Celsius]	Temperature [0.1 °Celsius]	-
Sunshine duration [0.1 hours]	Irradiation [J / cm^2]	-
Irradiation [J / cm^2]	Horizontal view [0-89, 0 = less than 100m]	-
Duration of rainfall [0.1 hours]	Clouds [0-9; 9 = sky invisible]	-
Clouds [0-9; 9 = sky invisible]	Humidity [%]	-
-	Fog [0 = no fog, 1 = fog]	-
-	Rain [0 = no rain, 1 = rain]	-
-	Snow [0 = no snow, 1 = snow]	-
-	Thunder [0 = no thunder, 1 = thunder]	-

Table 3.1: Features used by each model

3.3. Different model types

Three different types of Machine Learning models are used for each forecasting problem. The model types are; Support Vector Machine (SVM), Random Forest Regressor (RFR), and Artificial Neural Network (ANN). All these functions can be easily implemented in Python using Scikit-learn[26]. Below short descriptions of these model types can be found.

3.3.1. Support Vector Machine

A Support Vector Machine uses an Ordinary Least Squares (OLS) function to predict the outcome. In an OLS the goal is to minimize the squared error using the function seen in Equation 3.3. However, unlike most linear regression models, the SVM also tries to minimize the coefficients. This is done by defining an absolute error and tuning the model accordingly. [26] When implementing this method there will however be values outside of these absolute errors. This means that to make a good regressor slack variables should be implemented. Slack variables are the margin that certain variables fall outside the absolute error. The resulting function can be found in Equation 3.4. The function has constraints that can be found in Equation 3.5. The model contains two hyperparameters. Hyperparameters are parameters in the model that the user can tune to maximise accuracy. ϵ is the hyperparameter that denotes the absolute error. C is a hyperparameter that controls the tolerance for points outside ϵ.a

$$MIN \sum_{i=1}^{n} (y_i - w_i * x_i)^2 \tag{3.3}$$

$$MIN \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} |\xi_i| \tag{3.4}$$

$$|y_i - w_i * x_i| \leq \epsilon + |\xi_i| \tag{3.5}$$

3.3.2. Random Forest Regressor

A Random Forest Regressor is a Machine Learning model that uses randomly selected criteria to predict a value based on the different variables. [26] The model uses different decision trees with randomly selected splits in the variables. A decision tree is a model which has a structure similar to a flowchart. An example of a decision tree can be found in Figure 3.2. These different decision trees are then averaged which will hopefully result in good predictions for the output. The Random forest regressor also has a few important hyperparameters that can be tuned in order to make the model as accurate

as possible. The most important hyperparameters are the number of trees in the forest, the maximum number of features that can be used to split a node, and the maximum number of layers per tree.

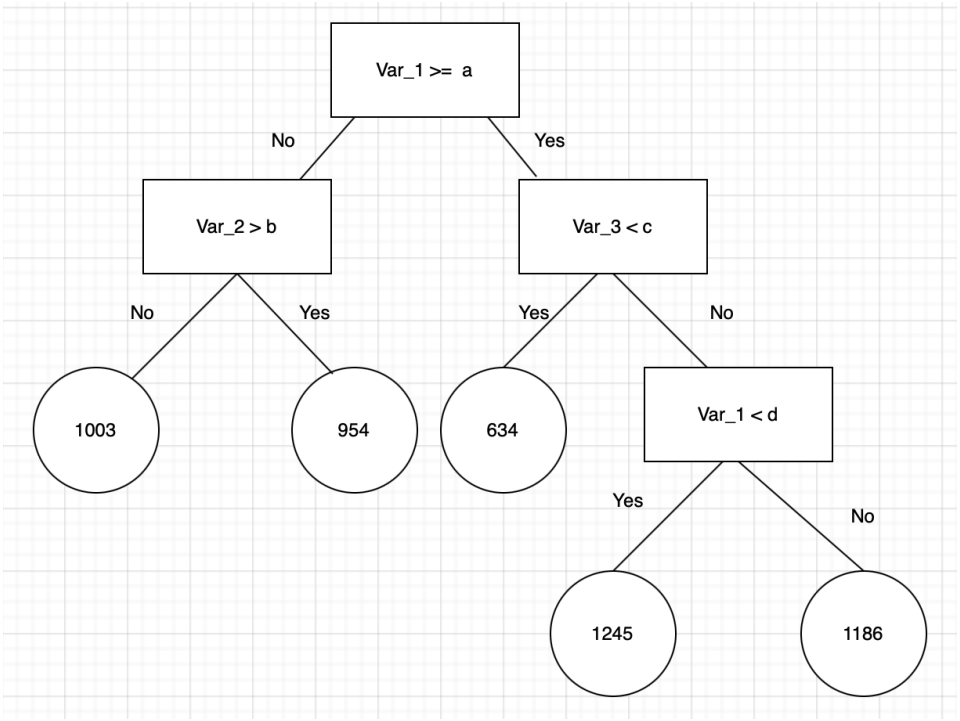


Figure 3.2: Example of a decision tree, obtained from [27]

3.3.3. Artificial Neural Network

The Artificial Neural network is the most complex model tested. An ANN aims to mimic the workings of a biological neural network. Per neuron, also called nodes or units, a non-linear function is used to process incoming signals. The output is then sent to the next layer of neurons. Per node, a certain weight is assigned. This weight is then adjusted as the model its training proceeds. As mentioned before ANNs consist of several layers, of which the input layer and the output layer are the only layers visible to the user. Even though the weights and functions of the hidden layers are unknown to the end-user, the number of layers and the number of nodes a certain layer consists of are defined by the user. These are thus hyperparameters. Several other hyperparameters can be defined before training. An ANN trains itself in a certain amount of iterations with a certain learning speed. The number of iterations is also called Epochs. These Epochs and learning speed can also be adjusted by the user. A visual representation of an ANN can be found in Figure 3.3.

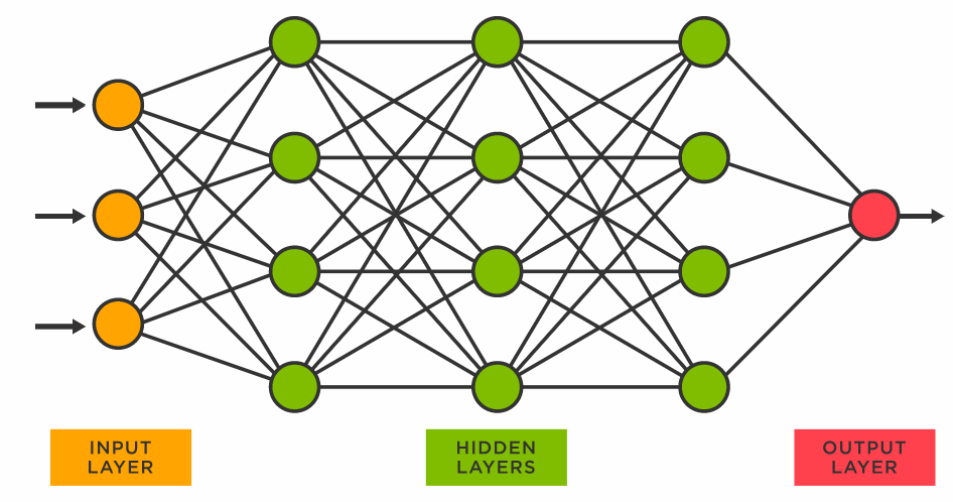


Figure 3.3: Example of an artificial neural network, obtained from [28]

4

Data

Data is essential for the models to properly work. As described before, each model has several inputs and one output. The weather data is obtained from the Dutch weather institute (KNMI). Of course, historical output data is also necessary to train the model on what decision it should make based on its input. In this chapter, the gathering, analyzing, updating, and pre-processing of the weather and output data is described.

4.1. Load model data gathering and analysis

The load model is going to use two data sets; the load data of several households and the weather data from the same period. The load data was gathered from [29]. This is an open data set provided by Liander, a DSO from the Netherlands. The data set consists of 82 households and their electricity and gas usage over one year. Upon inspecting this data it was found that some households have a lot of gaps in their data or there are missing specifications regarding the house type. So 37 households should be selected to make a data set suitable for the forecast of the Vechtrijk Neighbourhood. Besides this, the gas usage is in intervals of one hour, but the electricity usage is in intervals of 15 minutes. As it is the goal to deliver the forecasts in intervals of 15 minutes, the gas usage should be interpolated to obtain 15-minute intervals.

4.2. Load model data updating and pre-processing

4.2.1. Load data set

First, the gas usage is converted into electricity usage. Typical divisions of the gas usage are 75% for heating the house, 20 % for heating water and the remaining 5 % for cooking [30]. Heating the house and water can both be done using a heat pump with a coefficient of performance (COP) of approximately 3 [31]. Heat pumps can achieve COPs up to 5, however, in a neighbourhood of the future houses will be isolated better, so the gas usage from 2013 would be reduced in a neighbourhood of the future. To account for this better isolation, a COP of 3 has been chosen. This can be used in Equation 4.1 to calculate 95% of the gas power converted to equivalent electrical power. Here η_{boiler} is 33.3%. After this conversion, this usage is interpolated into 15-minute intervals and added to the electricity usage data set. As mentioned earlier, the load data set has several households without a type of house specification. The Vechtrijk Neighbourhood consists of 18 terraced houses, 1 corner house, 3 detached houses, 4 semi-detached houses, and 11 apartments. The closest approximation that could be achieved out of the Liander household data set consists of 32 households of which 4 are detached houses, 16 are semi-detached houses, 11 are terraced houses and there is one apartment. To get the 37 households needed, 4 terraced houses and one apartment were copied and used twice in the updated data set.

After selecting 37 households with a minimal amount of gaps, the few gaps that remain have to be filled in. When a gap was detected in a certain household the average of the other households in that time slot is calculated and filled in into the gap. Finally, the load of the 37 households is aggregated.

The code used can be found in subsection A.1.1 and consists out of two scripts. The first script converts the gas usage into electricity load and the second script selects the 37 correct households and removes outliers.

$$P_{el}[kW] = \frac{P_{gas}[kW] \cdot 0.95 \cdot \eta_{\text{boiler}}}{\text{COP}_{\text{heat pump}}} \tag{4.1}$$

The remaining 5% can be converted using formula. Here η_{furnace} = 85% and $\eta_{\text{induction}}$ = 35% [31]. Equation 4.2.

$$P_{el}[kW] = \frac{P_{gas}[kW] \cdot 0.05 \cdot \eta_{\text{furnace}}}{\eta_{\text{induction}}} \tag{4.2}$$

4.3. Weather

The weather data from the KNMI is easily accessible and obtained from its website [32]. Weather data from 2013 is used for the load forecasting and weather data from 2021 is used for the PV forecasting, both matching the exact dates and times from the load and PV data sets. The weather data consists of 22 variables in one-hour intervals. The data therefore is interpolated into 15-minute intervals. Some variables will not impact the load significantly, so they are removed from the weather data set. See Table 3.1 for the chosen features from the weather set for each model. These features are extracted and then interpolated into 15-minute intervals to use them in combination with the load data. The interpolating is done in Matlab, the code can be found in section A.4.

4.4. PV data gathering

The PV data is obtained from [33], a website that distributes data from a PV panel installation on the roof of the EEMCS Faculty. PV data for one full year was obtained, the year ranging from 23/01/2021 00:00 until 22/01/2022 23:59. The PV generation is measured in terms of energy after the AC conversion, so there should not be accounted for losses from conversion to AC. The measurement unit is watt-hours [Wh] per 15 minutes. In total, the PV system consists of 7 PV panels installed, 7 optimizers for each panel, and a DC-AC converter. The administrator mentioned that 1 or 2 panels have shut down because of malfunctioning optimizers. As the maximum generated power in the data is about 1500W, it is assumed that the system consists of 5 PV panels, as a maximum power generation of 1500/5 = 300W is reasonable for a PV panel. The panels are orientated towards the south, under an angle of 38 degrees. In the Vechtrijk neighbourhood, most PV panels will have a different orientation and angle. However, this is a detail outside the scope of this thesis and thus not much attention has been given to it. Since, if the proposed forecasting models would be put into practice for a real neighbourhood, real data should be used from that neighbourhood and the conversions from the original data sets which are required in this thesis no longer are necessary. For the same reasons, the degradation of the PV system is not considered. To get some sense of the PV generation in the Vechtrijk neighbourhood, the data is converted to the size of the total PV capacity after the forecasts have been made.

4.5. PV Data analysis and updating

Before the PV Data was used, it was manually analyzed in Excel. In this process, a lot of gaps, outliers and other irregularities were detected. The detected and removed irregularities are separated into irregularities within the PV data set itself, and iregularities seen when the weather data was added. All irregularities and their removal are described below.

The irregularities within the original data set:

1. **The inconsistent number of measurements per interval:** Mostly there are several measurements per 15-min interval, sometimes there are no measurements for hours or even days. The latter results in 794 gaps, and 15-min intervals with no generation data. First, one data point is created per 15-min interval. This is done by counting all the data points in 15-min intervals and adding them. The gaps are filled by taking the value from the day before. Since these are only 794 gaps, which is 2.4% of the final data set, the effect of using values from a day before is assumed to be acceptable.

2. **Offset**: From 13/08/2021 until the end of the data set an offset is present, day and night, which starts at roughly 4.2 and slowly increases up to 6.5 at the end of the data set. The offset is not constant and has a variance of about 0.2 per night. The offset is removed by calculating the average offset between 00:00 and 04:00 and subtracting this offset from the day and night before.

3. **Extreme values:** Based on the number of panels in the set and the date and time of the measurement, the maximum generated power is found to be 378.75 Wh per 15 minutes. 45 outliers with too large values, ranging from roughly 800 Wh up to 70,000 Wh are detected. The extreme values are removed by taking the average of the values of the first two data points before and after, which do not contain extreme values themselves. Furthermore, there are some data points with a negative generation value during the night, when the generation should be 0 Wh. These outliers are simply removed by setting all negative values equal to 0.

Error detected after adding the weather data:

1. **PV power generation while the irradiance is equal to zero:** Besides the offset, more data points were found which are nonzero at night. This problem is solved by simply setting all generation data points equal to zero when the irradiation is equal to zero. This also improves the data by setting all remaining deviations from the offset equal to zero on the nights.

2. **No PV power generation while the irradiance is nonzero:** While it could be correct that there is no PV power generation when there is a nonzero irradiance, for example when the irradiance is very weak in the morning, there are also data points where irradiance is quite large and the generated power still is zero. The benchmark is taken at 1 J/cm^2 and all rows where the generated power is 0 wh, while the irradiance is bigger than 1 J/cm^2 are removed from the data set. In total there are 2047 rows, resulting in a final data set with 32,992 data points per feature.

The updating of the PV data set is distributed over three python scripts. The first script combines all data points into one data point per 15-min interval and sets all negative values equal to zero. The script takes the last data point as the final data point and gives this data point the sum of all values from the past 15-min interval. This takes several hours, so it is separated from the second script. In the second script, the gaps are filled and the offset and extreme values are removed. Both scripts can be found in subsection A.1.2. The two remaining errors, a PV generation while irradiance is equal to zero and no generation while irradiance is nonzero, have to be removed when the weather data is added to the data set. This happens in the final PV script where also the PV forecasting models are created. The third PV script can be found in subsection A.2.2.

4.6. EV model data gathering and analysis

The EV model is based on one data set. The EV data set was retrieved from [34]. In this model, 348 electric vehicles are randomly selected from a data set created by the model proposed by Muratori. This data set is in ten-minute intervals and does not contain any gaps. In this data set, 6.6 kW charging is assumed.

4.7. EV model data updating and pre-processing

Since this model is separate from the other models it is not a problem that the data set has intervals of 10 minutes, this will thus not be adjusted into 15-minute intervals. The data set does however consist of 348 vehicles even though the central parking place in the Vechtrijk neighbourhood only has space for 42 vehicles. This is why 42 vehicles are selected from the data set and the rest is dropped. Since the wanted forecast is the number of vehicles charging in a certain time interval, the charging power is unnecessary information and is thus changed into a binary value, ’0’ if the car is not charging and ’1’ if the car is charging. After this, the total cars charging in a certain time is calculated and the individual cars are dropped from the data set. The code used can be found in subsection A.1.3.

5 Results

In this chapter, the results of the thesis are discussed. First, the hyperparameter tuning is addressed, then the importance of features is given for the load and PV models. Because there are not many features, it was decided to use each feature for every model. Still, it gives more understanding to find out the importance of each feature. The best parameters are used to create models which predict y on the features from the test set. Then R^2 scores of these predicted y values and the actual y values from the test set are computed. The R^2 scores are used as measure of performance and are presented and compared. All code can be found in section A.2.

5.1. Hyperparameter tuning

In order to find the best model, for each model the hyperparameters are tuned. Hyperparameters are the parameters of a ML model, which can be manually tuned to get better accuracy. The parameters of a model are the parameters which define the model, so changing them means changing the model. The best parameters are found by Scikit learn using the function (model_name).*fit(X_train, y_train)* A random search is used to optimize the hyperparameters, and the function *RandomizedSearchCV()* from the Scikit learn tool has been used. It takes random values within a certain range and tests how accurate the model is using cross-validation. Here it is chosen to use n=50 iterations for the random search to find a close approximation of the best hyperparameters available while avoiding dramatically long run times. In order to find the best hyperparameters, all available values within the range should be tested in combination with all other hyperparameters and their values. This would take very long and is less efficient than using this randomized method. In Table 5.1 the hyperparameters are shown for the SVM. The second column consists of the range of values the random search will choose between. Table 5.2 and Table 5.3 show the same for the RFR and the ANN respectively.

Hyperparameter	**Range**
Tolerance	from 0.0001 to 0.9
ϵ	from 0.01 to 1
C	from 0.1 to 5

Table 5.1: Hyperparameters and range of the SVM

Hyperparameter	**Range**
Number of trees	from 200 to 2000
Number of features per split	auto or sqrt
Maximum of levels per tree	from 10 to 110
Minimum of samples required for split	2, 5 or 10
Minimum of samples required per leaf node	1,2 or 4
Bootstrap	true or false

Table 5.2: Hyperparameters and range of the RFR

Hyperparameter	Range
Amount of hidden layers	from 1 to 100
Activation of the layers	identity, logistic, tanh or relu
α	from 10^{-5} to 10^{-3}
Learning rate type	constant, invscaling or adaptive
Learning rate start	from 10^{-5} to 10^{-2}
Tolerance	from 10^{-5} to 10^{-3}
Maximum Epochs	from 2 to 1000

Table 5.3: Hyperparameters and range of the ANN

5.2. Feature importance

The feature importance for the load and PV model are stated here. The feature importance is found using the Scikit learn attribute (model_name).*feature_importances_*. The ANN and SVM functions from scikit learn have no feature importance attribute, but as the RFR model is quite accurate for all forecasting problems, from the feature importance of this model type conclusions on the overall importance of features can be drawn. As the EV model has only one feature, computing the feature importance is not possible.

5.2.1. Load model feature importance

It can be seen in Table 5.4 that the time and temperature are by far the most important features for making accurate forecasts for the load, as was expected. It is noticeable that the feature 'workday' has the lowest importance for making forecasts. This means that either in the used data set the load curve is not very different for workdays and holidays, or that the classified input (0 = holiday, 1 = workday) is not understood correctly by the regression model.

Time	Workday	Wind	Sun time	Temp	Irradiation	Rain	Clouds
0.335	0.014	0.082	0.066	0.409	0.035	0.021	0.039

Table 5.4: Feature importance for the best load model, rounded to 3 decimals

5.2.2. PV model feature importance

As is visible in Table 5.5, irradiation is the most important feature for making accurate PV power generation forecasts. Also the time, horizontal view and humidity features are quite important.

Time	Season	Wind	T	Irr	View	Clouds	Humidity	Fog	Rain	Snow	Thunder
0.127	0.026	0.028	0.102	0.383	0.106	0.024	0.193	0.000	0.006	0.000	0.002

Table 5.5: Feature importance for the best PV model, rounded to 3 decimals

5.3. Load model results

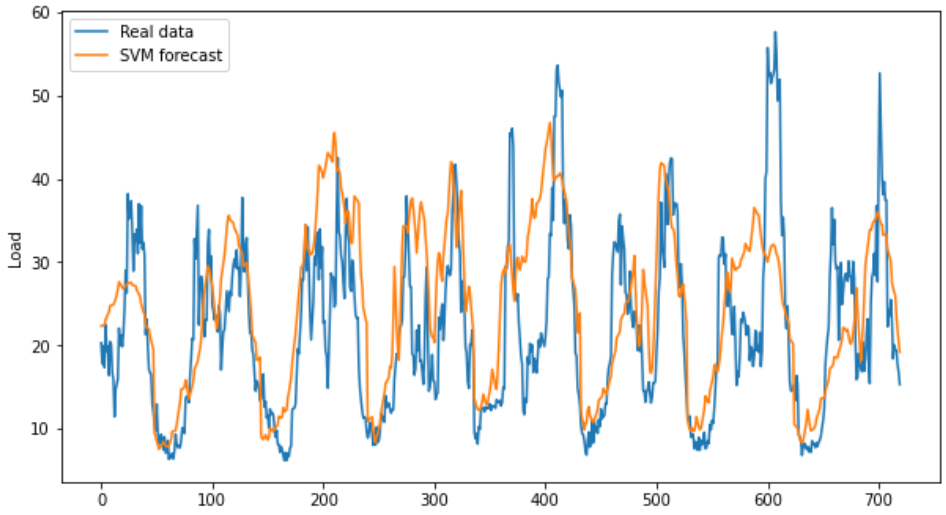
First, load forecasting models are trained on data sets that contain both workdays and holidays. Afterward, separate models are trained only on the data of holidays and the data of workdays. The results are compared.

5.3.1. Holiday & workday data combined

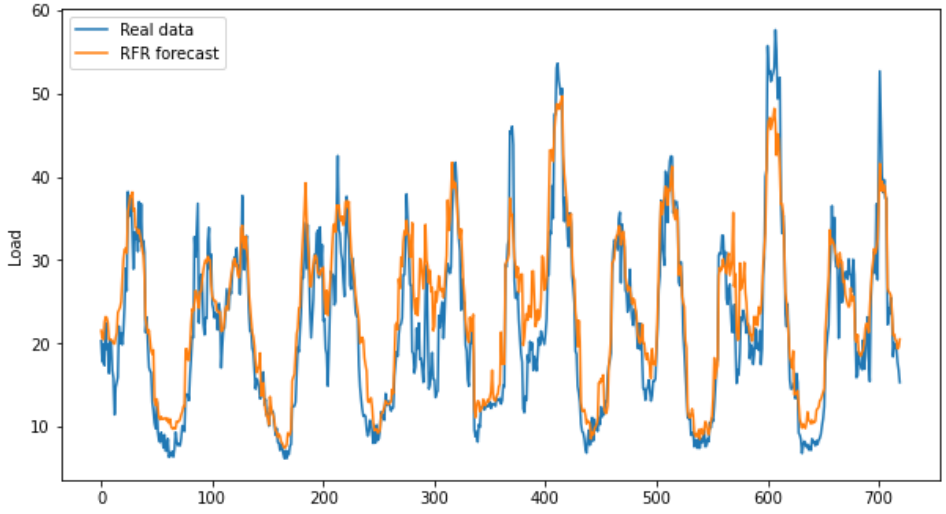
In Figure 5.1 load forecasts of a five-day interval can be seen. It is visible that the RFR model matches the trend the best. Also, from Table 5.6 it can be seen that the RFR model has the best R^2 score, so it is concluded that in this particular forecasting problem the RFR model performs the best.

	RFR	SVM	ANN
R^2	0.93	0.74	0.86

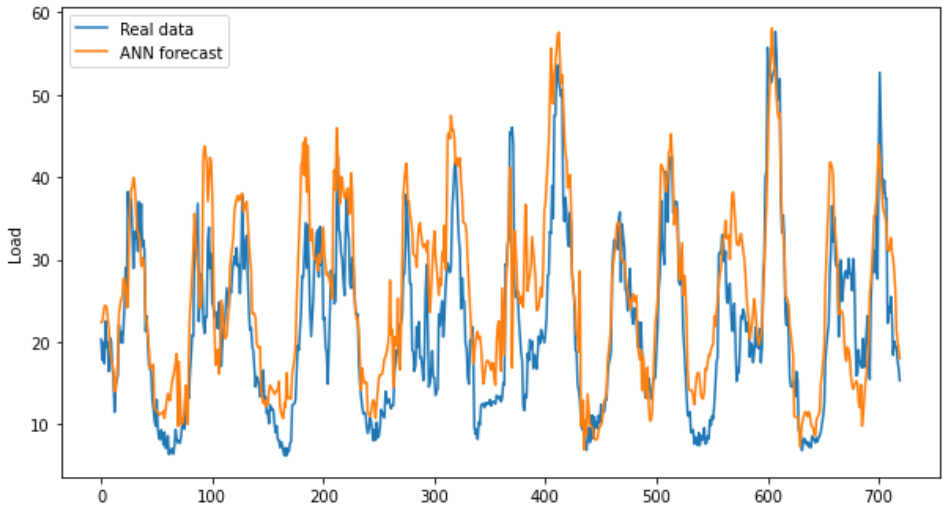
Table 5.6: R^2 score of best model prediction vs actual y_{test} data



(a) Load forecasted with svm



(b) Load forecasted with RFR



(c) Load forecasted with ANN

Figure 5.1

5.3.2. Support vector machine

The SVM achieved an optimal form with the hyperparameters found in Table 5.7. The R^2 of the SVM with these hyperparameters is 0.74 with a standard deviation of $8 * 10^{-3}$.

Hyperparameter	Value
Tolerance	0.327
ϵ	0.95
C	4.951

Table 5.7: Hyperparameters and values of the SVM

5.3.3. Random forest regressor

The RFR achieved an R^2 of 0.93 with a standard deviation of 10^{-3} using the hyperparameters found in Table 5.8. Based on this R^2 and the very low standard deviation, it is concluded that for this particular forecasting problem the RFR model performs the best.

Hyperparameter	Value
Number of trees	400
Number of features per split	sqrt
Maximum of levels per tree	none
Minimum of samples required for split	2
Minimum of samples required per leaf node	1
Bootstrap	false

Table 5.8: Hyperparameters and values of the RFR

5.3.4. Artificial neural network

In Table 5.9 the hyperparameters of the best ANN model found by the randomized search can be found. Using these hyperparameters the model got a R^2 score of 0.856 with a standard deviation of $3 * 10^{-3}$. The ANN model also proves to be an adequate model for this particular forecasting problem.

Hyperparameter	Value
Amount of hidden layers	85
Activation of the layers	tanh
α	$3.94 * 10^{-4}$
Learning rate type	constant
Learning rate start	$7.37 * 10^{-3}$
Tolerance	$4.27 * 10^{-4}$
Maximum Epochs	755

Table 5.9: Hyperparameters and values of the ANN

5.3.5. Split holidays & workdays data sets

After splitting the load data set into separate holidays and workdays load data sets, again the best hyperparameters are found and the results are obtained by letting the models predict y on the features from the test set and compute the R^2 scores of these predicted y values and the actual y values from the test set. In Table 5.10 it is visible that the R^2 score slightly increased when only using the holidays data; 0.02 for the RFR, 0.03 for the SVM and 0.01 for the ANN compared to the original load data set. However, results on only the workdays load were different; the R^2 score increased with 0.01 for the RFR and decreased with 0.03 for both the SVM and the ANN compared to the original results. Based on the holidays load data, it could be argued that for making forecasts on the load data from holidays, it is better to split the data into holidays and workdays data sets and only train on the data from the holidays than to make a classified input (holiday = 0, workday = 1). However, the performance of the forecasts of the load on workdays actually decreased after splitting the data for both the SVM and the ANN and only increased with 0.01 for the RFR. Thus there can not be concluded on these results that

splitting the load data into an holidays and workdays data significantly increases the forecasts on the load.

Holidays load data set				Workdays load data set			
	RFR	**SVM**	**ANN**		**RFR**	**SVM**	**ANN**
R^2	0.95	0.77	0.87	R^2	0.94	0.71	0.83

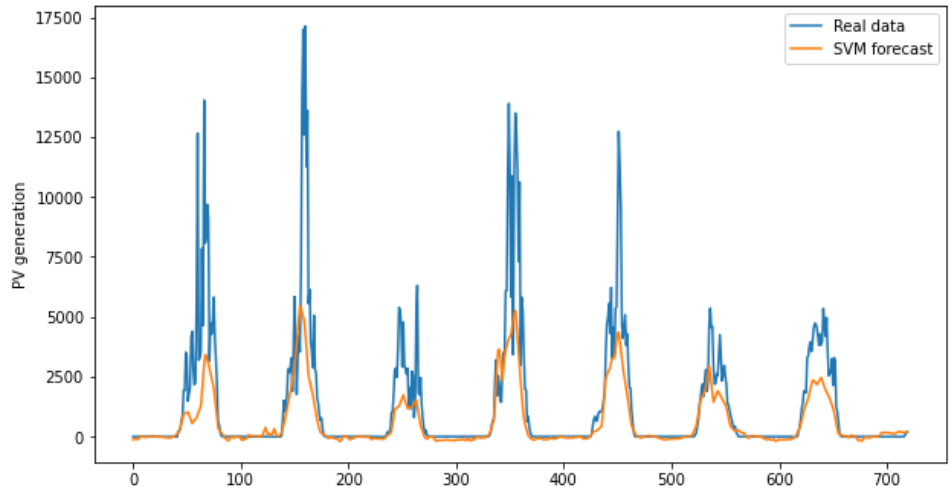
Table 5.10: R^2 scores of load forecasts for separate holidays and workdays data sets

5.4. PV model results

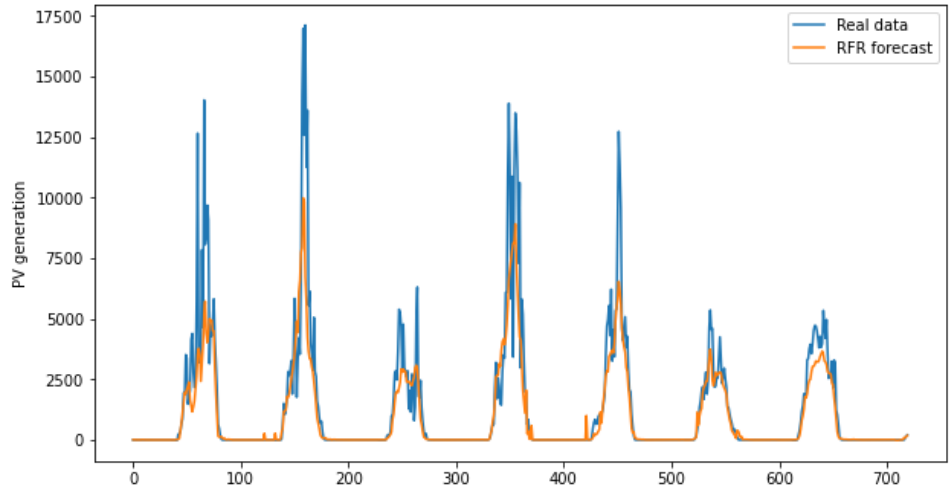
The PV model was first trained on a data set that contains every season. Then the model was trained with the seasons separated. This did not increase the accuracy of the model as can be seen in Table 5.11 so it is chosen to train the model with the complete data set. It is again the case that the RFR is the most accurate model of all for this particular forecasting problem. With an accuracy of over 90% and again very low standard deviation, it is the best option for forecasting PV generation. In Figure 5.2 the forecasts are shown on a five-day interval.

Model type	**Whole data set**	**Only winter**	**Only spring**	**Only summer**	**Only fall**
SVM	0.51	0.2	0.18	0.07	0.04
RFR	0.91	0.88	0.89	0.88	0.93
ANN	0.82	0.82	0.82	0.82	0.88

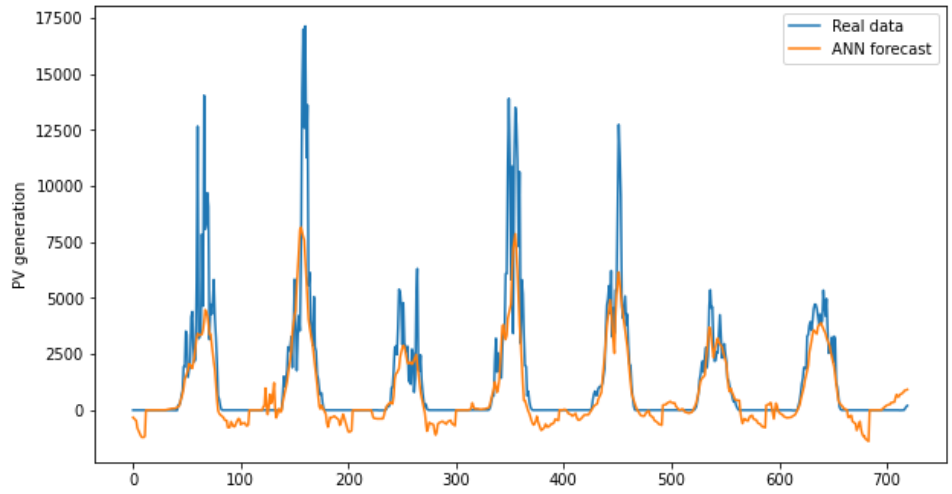
Table 5.11: R^2 score of best model prediction vs actual y_{test} data



(a) PV generation forecasted with svm



(b) PV generation forecasted with RFR



(c) PV generation forecasted with ANN

Figure 5.2

5.4.1. Support vector machine

The SVM achieved a R^2 score of 0.508 with a standard deviation of $2 * 10^{-2}$. It achieved this score using the hyperparameters found in Table 5.12. As can be seen, the hyperparameters indicate that this data is difficult to forecast using a SVM since the model uses a high tolerance, high C and a high ϵ.

Hyperparameter	Value
Tolerance	0.818
ϵ	0.99
C	5

Table 5.12: Hyperparameters and values of the SVM

5.4.2. Random forest regressor

A R^2 score of 0.913 was achieved by the RFR using the hyperparameters found in Table 5.13. The standard deviation is $2 * 10^{-3}$.

Hyperparameter	Value
Number of trees	400
Number of features per split	sqrt
Maximum of levels per tree	none
Minimum of samples required for split	2
Minimum of samples required per leaf node	1
Bootstrap	false

Table 5.13: Hyperparameters and values of the RFR

5.4.3. Artificial neural network

In Table 5.9 the hyperparameters used by the ANN can be found. With these hyperparameters, the ANN got a R^2 of 0.82 with a standard deviation of $4 * 10^{-3}$.

Hyperparameter	Value
Amount of hidden layers	85
Activation of the layers	relu
α	$1.92 * 10^{-4}$
Learning rate type	invscaling
Learning rate start	$6.85 * 10^{-3}$
Tolerance	$4.27 * 10^{-4}$
Maximum Epochs	755

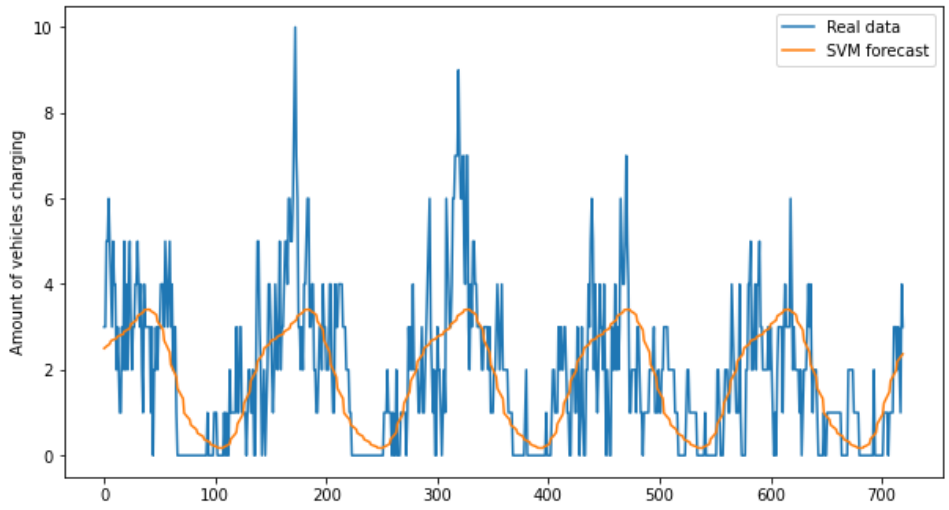
Table 5.14: Hyperparameters and values of the ANN

5.5. EV model results

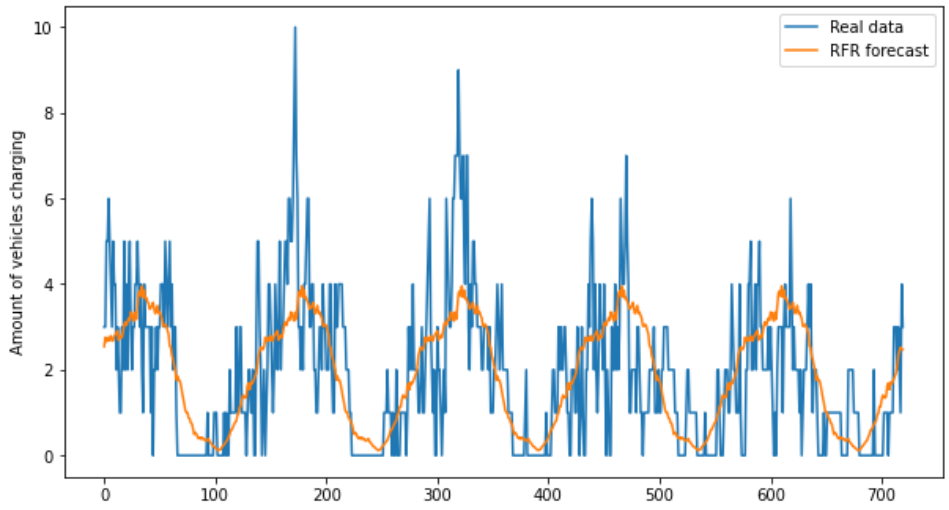
In Figure 5.3, plots are shown with the actual EV data on a five-day interval, the other lines are the forecasts by the different model types. It can be seen that all models follow the trend, however, they all struggle to match the peaks. This could mean that the data set with 42 cars is too small to accurately make forecasts. As can be seen in Table 5.15 the ANN model is the most accurate model type for this use case. However, after the cross-validation it was found that the ANN did have a high standard deviation, which makes it hard to choose one particular model as the best performer.

	RFR	SVM	ANN
R^2	0.427	0.419	0.428

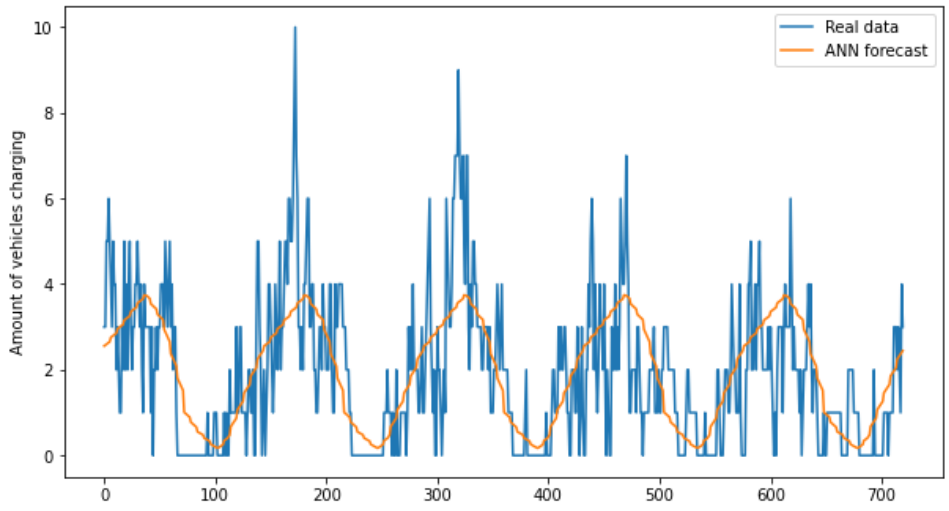
Table 5.15: R^2 score of best model prediction vs actual y_{test} data



(a) Amount of charging vehicles forecasted with svm



(b) Amount of charging vehicles forecasted with RFR



(c) Amount of charging vehicles forecasted with ANN

Figure 5.3

5.5.1. Support vector machine

In Table 5.16 the optimal hyperparameters found by the randomized search are shown. When using these hyperparameters, the SVM model achieved an R^2 value of 0.419 with a standard deviation of $7 * 10^{-3}$.

Hyperparameter	Value
Tolerance	0.5091
ϵ	0.28
C	4.604

Table 5.16: Hyperparameters and values of the SVM

5.5.2. Random forest regressor

The optimal values for the specified hyperparameters can be found in Table 5.17. using these hyperparameters, the RFR achieved an R^2 of 0.427 with a standard deviation of $4 * 10^{-3}$.

Hyperparameter	Value
Number of trees	200
Number of features per split	auto
Maximum of levels per tree	10
Minimum of samples required for split	5
Minimum of samples required per leaf node	4
Bootstrap	true

Table 5.17: Hyperparameters and values of the RFR

5.5.3. Artificial neural network

The ANN had the highest accuracy of all three model types with an R^2 score of 0.428 and a standard deviation of $2.7 * 10^{-2}$. The reason the best model only has an accuracy of 42.8% is the lack of features in the data set. With the only feature being a certain time and estimating how many cars are charging at that time is very difficult and not very precise. An extra feature was added to try and increase the accuracy but this was unfortunately not the case. The extra feature added was information on whether it was a week or weekend day. The hyperparameters that achieved this accuracy can be found in Table 5.18.

Hyperparameter	Value
Amount of hidden layers	85
Activation of the layers	relu
α	$1.9 * 10^{-4}$
Learning rate type	invscaling
Learning rate start	$6.8 * 10^{-3}$
Tolerance	$6.2 * 10^{-5}$
Maximum Epochs	694

Table 5.18: Hyperparameters and values of the ANN

6

Conclusion and further work

The following conclusions can be drawn from the results and may be interesting for future projects using similar data sets. First of all, for accurately forecasting the load, the temperature and time are the most important features. For accurately forecasting the PV power generation, especially the irradiation is most important, but time, horizontal view and humidity are also important features. Furthermore, it is shown that random forest regression models can accurately forecast both the load and PV power generation in a residential neighbourhood with 37 households with an accuracy above 90%. Artificial neural networks are also adequate models for these forecasting problems, but because they are harder to understand and not necessarily better, it is recommended to start with random forest regressors before making neural networks. Furthermore, support vector machines have proven to be less suitable for forecasting the aggregated load and PV power generation in a neighbourhood.

When splitting the load data set into separate holidays and workdays load data sets, the performance of the models increased slightly when only using data from the holidays data set compared to the models trained with the complete load data set. However, the increase was quite marginal and with the separate workdays load data set, the perfomance actually decreased for ANN and SVM models. Therefore, based on these results it can not be concluded that splitting the load data into workdays and holidays load data sets is significantly better than keeping the load data in one data set.

The quality of the EV forecasts is a lot lower than the PV power and load forecasts. Possible reasons are that the data set contains too few EVs and that it contains too few features. For further research, it is recommended to try and find a real EV data set, because then weather data can be added to this data set and more features are available. Also, creating a data set with more EVs will increase the performance of forecasting models.

Bibliography

[1] P. R. et al., "Greenhouse gas emissions in the netherlands 1990–2019, national inventory report 2021," 2021. https://www.rivm.nl/bibliotheek/rapporten/2021-0007.pdf, accessed on: 29/04/2022. Doi: 10.21945/RIVM-2021-0007.

[2] N. Brannan, "Liander niet goed voorbereid op overvol stroomnet, gemeente overvallen door problemen," 2022. shorturl.at/ruzH9, accessed on: 18/05/2022.

[3] M. J. L. et al., "Hernieuwbare energie in nederland - buitenluchtwarmte," 2021. https://www.cbs.nl/nl-nl/longread/aanvullende-statistische-diensten/2021/hernieuwbare-energie-in-nederland-2020?onepage=true#c-7--Buitenluchtwarmte, accessed on 29/04/2022.

[4] C. B. voor de Statistiek (CBS), "Aantal stekkerauto's, 2014-2021," 2021. https://www.cbs.nl/nl-nl/maatwerk/2021/41/aantal-stekkerauto-s-2014-2021, accessed on: 29/04/2022.

[5] "Aanbod vechtrijk xb2." https://account.woneninweespersluis.nl/aanbod-vechtrijk/.

[6] "Netcapaciteit en netcongestie." https://www.rvo.nl/onderwerpen/zonne-energie/netcapaciteit-netcongestie.

[7] "Capaciteitskaart nederland." https://capaciteitskaart.netbeheernederland.nl/?_ga=2.56067567.538483043.1651476493-1899990241.1649077574.

[8] T. Hong and S. Fan, "Probabilistic electric load forecasting: A tutorial review," *International Journal of Forecasting*, vol. 32, no. 3, pp. 914–938, 2016.

[9] Y. Wang, Q. Chen, T. Hong, and C. Kang, "Review of smart meter data analytics: Applications, methodologies, and challenges," *IEEE Transactions on Smart Grid*, vol. 10, no. 3, pp. 3125–3148, 2019.

[10] A. Mosavi, M. Salimi, S. Faizollahzadeh Ardabili, T. Rabczuk, S. Shamshirband, and A. R. Varkonyi-Koczy, "State of the art of machine learning models in energy systems, a systematic review," *Energies*, vol. 12, no. 7, 2019.

[11] N. G. Paterakis, E. Mocanu, M. Gibescu, B. Stappers, and W. van Alst, "Deep learning versus traditional machine learning methods for aggregated energy demand prediction," pp. 1–6, 2017.

[12] M. Alamaniotis, "Synergism of deep neural network and elm for smart very-short-term load forecasting," pp. 1–5, 2019.

[13] P. Singh, P. Dwivedi, and V. Kant, "A hybrid method based on neural network and improved environmental adaptation method using controlled gaussian mutation with real parameter for short-term load forecasting," *Energy*, vol. 174, pp. 460–477, 2019.

[14] J. Moon, Y. Kim, M. Son, and E. Hwang, "Hybrid short-term load forecasting scheme using random forest and multilayer perceptron," *Energies*, vol. 11, no. 12, 2018.

[15] J. Lu, J. Qian, Q. Zhang, S. Liu, F. Xie, and H. Xu, "Best practices in china southern power grid competition of ai short-term load forecasting," pp. 1–5, 2020.

[16] R. Ghotge, Y. Snow, S. Farahani, Z. Lukszo, and A. van Wijk, "Optimized scheduling of ev charging in solar parking lots for local peak reduction under ev demand uncertainty," *Energies*, vol. 13, no. 5, 2020.

[17] K. Wang, X. Qi, and H. Liu, “A comparison of day-ahead photovoltaic power forecasting models based on deep learning neural network,” *Applied Energy*, vol. 251, p. 113315, 2019.

[18] Y. Huang, J. Lu, C. Liu, X. Xu, W. Wang, and X. Zhou, “Comparative study of power forecasting methods for pv stations,” pp. 1–6, 2010.

[19] M. Pierro, D. Gentili, F. R. Liolli, C. Cornaro, D. Moser, A. Betti, M. Moschella, E. Collino, D. Ronzio, and D. van der Meer, “Progress in regional pv power forecasting: A sensitivity analysis on the italian case study,” *Renewable Energy*, vol. 189, pp. 983–996, 2022.

[20] J. Faraji, A. Ketabi, H. Hashemi-Dezaki, M. Shafie-Khah, and J. P. S. Catalão, “Optimal day-ahead self-scheduling and operation of prosumer microgrids using hybrid machine learning-based weather and load forecasting,” *IEEE Access*, vol. 8, pp. 157284–157305, 2020.

[21] W. Khan, S. Walker, and W. Zeiler, “Improved solar photovoltaic energy generation forecast using deep learning-based ensemble stacking approach,” *Energy*, vol. 240, p. 122812, 2022.

[22] S. S. . V. P. Muhammad Ehsan, R., “Day-ahead forecasting of solar photovoltaic output power using multilayer perceptron,” *Neural Comput & Applic*, vol. 28, p. 3981–3992, 2017.

[23] T. Yao, J. Wang, H. Wu, P. Zhang, S. Li, K. Xu, X. Liu, and X. Chi, “Intra-hour photovoltaic generation forecasting based on multi-source data and deep learning methods,” *IEEE Transactions on Sustainable Energy*, vol. 13, no. 1, pp. 607–618, 2022.

[24] K. Brush, “Moscow method.” `https://www.techtarget.com/searchsoftwarequality/definition/MoSCoW-method`.

[25] G. J. et al., *An introduction to Statistical Learning*. Springer Texts in Statistics, 2021.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[27] N. Beheshti, “Random forest regression.” `https://towardsdatascience.com/random-forest-regression-5f605132d19d`.

[28] “What is a neural network?.” `https://www.tibco.com/reference-center/what-is-a-neural-network`.

[29] Liander, “Open data.” `https://www.liander.nl/partners/datadiensten/open-data/data`.

[30] V. de Vastelastenbond, “Gasverbruik: Gasverbruik in jouw huishouden,” 2020. `https://www.vastelastenbond.nl/energie/gasverbruik-in-uw-huishouden/`, accessed on: 31/05/2022.

[31] R. Khezri, A. Mahmoudi, and D. Whaley, “Optimal sizing and comparative analysis of rooftop pv and battery for grid-connected households with all-electric and gas-electricity utility,” *Energy*, vol. 251, p. 123876, 2022.

[32] KNMI. `https://www.knmi.nl/nederland-nu/klimatologie/daggegevens`.

[33] “Pvlewi.” `http://pvlewi.ewi.tudelft.nl/`, visited on: 07/06/2022.

[34] M. Muratori, “Impact of uncoordinated plug-in electric vehicle charging on residential power demand - supplementary data,” 6 2017.

A

Appendix

A.1. Python Code: Data pre-processing

A.1.1. Load data updating

```
# -*- coding: utf-8 -*-
"""
@author: Pieter Gommers

Description: This is part 1 of two scripts that update a load data set
to prepare it for training ML models. In this part, the gas usage data
is converted to electrical load by assuming all households will heat
their homes with heat pumps.
"""

import pandas as pd

# Variables
#predefinedIndices = [4, 30, 72, 77, 78]

# Importing from csv to dataframes
dataFrameReturn = pd.read_csv('Ruwe data/Zonnedael - slimme meter dataset - 2013 -
    Teruglevering.csv',
                              header=None, delimiter=';', on_bad_lines='skip', low_memory=
    False)
dataFrameElectrical = pd.read_csv('Ruwe data/Zonnedael - slimme meter dataset - 2013 -
    Levering.csv', delimiter=';', on_bad_lines='skip', low_memory=False)
dataFrameGas = pd.read_csv('Ruwe data/Zonnedael - slimme meter dataset - 2013 - Slimme meter.
    csv', delimiter=';', on_bad_lines='skip', low_memory=False)
dataFrameGas = dataFrameGas.loc[:, dataFrameGas.columns != 'Klant 77']

def timePrep():
    # Getting the timestamp in a single column
    dataFrameDateTime = dataFrameElectrical[:35040]['datetime']
    dataFrameDateTime = dataFrameDateTime.apply(lambda x: x.replace(' 0:', ' 00:'))
    dataFrameDateTime = dataFrameDateTime.apply(lambda x: x.replace(' 1:', ' 01:'))
    dataFrameDateTime = dataFrameDateTime.apply(lambda x: x.replace(' 2:', ' 02:'))
    dataFrameDateTime = dataFrameDateTime.apply(lambda x: x.replace(' 3:', ' 03:'))
    dataFrameDateTime = dataFrameDateTime.apply(lambda x: x.replace(' 4:', ' 04:'))
    dataFrameDateTime = dataFrameDateTime.apply(lambda x: x.replace(' 5:', ' 05:'))
    dataFrameDateTime = dataFrameDateTime.apply(lambda x: x.replace(' 6:', ' 06:'))
    dataFrameDateTime = dataFrameDateTime.apply(lambda x: x.replace(' 7:', ' 07:'))
    dataFrameDateTime = dataFrameDateTime.apply(lambda x: x.replace(' 8:', ' 08:'))
    dataFrameDateTime = dataFrameDateTime.apply(lambda x: x.replace(' 9:', ' 09:'))
    print('Datetime ready...')
    return dataFrameDateTime

def getUserIndices(dataFrameReturn, predefinedIndices):
    firstrow = dataFrameReturn.iloc[0]
    indices = []

```

```
    for i in range(len(firstrow)):
        if type(firstrow[i]) == str and int(firstrow[i]) == 0:
            indices.append(i)

    for index in predefinedIndices:
        if index in indices:
            indices.remove(index)

    print('Indices ready...')
    return indices

def ePrep():
    dataFrameE_prep = dataFrameElectrical.iloc[:35040, 2:79]
    dataFrameE_prep = dataFrameE_prep.fillna(0)
    dataFrameE_prep = dataFrameE_prep.apply(lambda x: x * 4 / 1000) # Conversion to kW

    print('E-prep ready...')
    return dataFrameE_prep

def gPrep():
    dataFrameG_prep = dataFrameGas.stack().str.replace(',', '.').unstack().iloc[:35040, 1:]
    dataFrameG_prep = dataFrameG_prep.astype(float)
    dataFrameG_prep = dataFrameG_prep.apply(lambda x: x * 35.17 / 3.6) # Conversion from m^3
    to kW
    dataFrameG_prep = dataFrameG_prep.apply(lambda x: x * 0.05 * 0.412 + x * 0.20 * 0.333 + x
     * 0.75 * 0.333)

    dataFrameG_ext = pd.DataFrame(columns=dataFrameG_prep.columns)
    for i in range(len(dataFrameG_prep)):
        if i == 2190: print('25% Gas conversion')
        if i == 4380: print('50% Gas conversion')
        if i == 6570: print('75% Gas conversion')
        if i == 8755: print('100% Gas conversion')
        for j in range(4):
            dataFrameG_ext = dataFrameG_ext.append(dataFrameG_prep.iloc[i], ignore_index=True
    )

    print('G-prep ready...')
    return dataFrameG_ext

dfGas = gPrep()

dfTime = timePrep()
#userIndices = getUserIndices(dataFrameReturn, predefinedIndices)
dfElectricity = ePrep()
#print(userIndices)

#print(dfTime.head)

df_full = dfElectricity.add(dfGas, fill_value=0)
#df_full = df_full.apply(lambda x: x / 1000) # Conversion from kW to MW
dfFullData = pd.concat([dfTime, df_full], axis=1)


#print(dfFullData.head)

dfFullData.to_csv('ElectricityGasDataFullkW2.csv', index=False)

# dfFullData = pd.concat([dfTime, dfElectricity, dfGas], axis=1)
# print(dfFullData.shape)

# dfFullData.to_csv('Split.csv', index=False)

# dfTime = timePrep()
# userIndices = getUserIndices(dataFrameReturn, predefinedIndices)
# dfElectricity = ePrep(userIndices)
# df_full = dfElectricity.add(dfGas, fill_value=0)
# dfFullData = pd.concat([dfTime, df_full], axis=1)

#print(dfFullData.loc[:, 'Klant 26'].max())

```

```
dataFrameE_prep = dataFrameElectrical.drop(['datetime', 'SOM', 'leverende klanten', 'niet leverenden',
                        'Klant 1', 'Klant 2', 'Klant 3', 'Klant 4', 'Klant 5',
                        'Klant 8', 'Klant 9', 'Klant 10', 'Klant 11', 'Klant 14' ,
                        'Klant 16', 'Klant 21', 'Klant 24', 'Klant 27', 'Klant 28',
                        'Klant 29', 'Klant 30', 'Klant 31', 'Klant 36', 'Klant 37',
                        'Klant 40', 'Klant 41', 'Klant 44', 'Klant 45', 'Klant 47',
                        'Klant 48', 'Klant 49', 'Klant 50', 'Klant 51', 'Klant 55',
                        'Klant 57', 'Klant 59', 'Klant 63', 'Klant 64', 'Klant 65',
                        'Klant 66', 'Klant 67', 'Klant 69', 'Klant 72', 'Klant 73',
                        'Klant 74', 'Klant 77', 'Klant 78', 'Klant 80', 'Klant 82'], axis = 1)

dataFrameE_prep = dataFrameE_prep.iloc[:35040, :]
for i in range(35040):
    if dataFrameE_prep.iloc[i].isna().any():
        dataFrameE_prep.iloc[i] = dataFrameE_prep.iloc[i].fillna(dataFrameE_prep.iloc[i].mean())
dataFrameE_prep = dataFrameE_prep.apply(lambda x: x * 4 / 1000) # Conversion to kW
dataFrameE_prep

dataFrameG_prep = dataFrameGas.drop(['datetime', 'Som',
                        'Klant 1', 'Klant 2', 'Klant 3', 'Klant 4', 'Klant 5',
                        'Klant 8', 'Klant 9', 'Klant 10', 'Klant 11', 'Klant 14' ,
                        'Klant 16', 'Klant 21', 'Klant 24', 'Klant 27', 'Klant 28',
                        'Klant 29', 'Klant 30', 'Klant 31', 'Klant 36', 'Klant 37',
                        'Klant 40', 'Klant 41', 'Klant 44', 'Klant 45', 'Klant 47',
                        'Klant 48', 'Klant 49', 'Klant 50', 'Klant 51', 'Klant 55',
                        'Klant 57', 'Klant 59', 'Klant 63', 'Klant 64', 'Klant 65',
                        'Klant 66', 'Klant 67', 'Klant 69', 'Klant 72', 'Klant 73',
                        'Klant 74', 'Klant 75', 'Klant 78', 'Klant 80', 'Klant 82'], axis = 1)

dataFrameG_prep = dataFrameG_prep.stack().str.replace(',', '.').unstack().iloc[:35040, :]
dataFrameG_prep = dataFrameG_prep.astype(float)
for i in range(8758):
    if dataFrameG_prep.iloc[i].isna().any():
        dataFrameG_prep.iloc[i] = dataFrameG_prep.iloc[i].fillna(dataFrameG_prep.iloc[i].mean())

dataFrameG_prep = dataFrameG_prep.apply(lambda x: x * 35.17 / 3.6) # Conversion from m^3 to kW
dataFrameG_prep = dataFrameG_prep.apply(lambda x: x * 0.05 * 0.412 + x * 0.20 * 0.333 + x * 0.75 * 0.333)
dataFrameG_ext = pd.DataFrame(columns=dataFrameG_prep.columns)

for i in range(len(dataFrameG_prep)):
    if i == 2190: print('25% Gas conversion')
    if i == 4380: print('50% Gas conversion')
    if i == 6570: print('75% Gas conversion')
    if i == 8755: print('100% Gas conversion')
    for j in range(4):
        dataFrameG_ext = dataFrameG_ext.append(dataFrameG_prep.iloc[i], ignore_index=True)
dataFrameG_ext

df_full = dataFrameE_prep.add(dataFrameG_ext, fill_value=0)
dfTime = timePrep()
#df_full = df_full.apply(lambda x: x / 1000) # Conversion from kW to MW
dfFullData = pd.concat([dfTime, df_full], axis=1)


#print(dfFullData.head)

dfFullData.to_csv('EGdataGroupB.csv', index=False)
dfFullData
```

```
"""
@author: Ruben Eland, Daan van Dingstee

Description: This is part 2 of two scripts that update a load data set
to prepare it for training ML models.
```

```
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import datetime as dt

#%% Creating dataframes
filename1 = 'C:/Users/raela/OneDrive/Elektro/year 4/BAP/Thesis/Data sets/EGdataGroupB.csv'
load_df = pd.read_csv(filename1)

#%% Removing part of sets and updating with electric heating data
#Drop all names
# load_df = load_df.drop([0, 1])

#Save dates & times
dates_df = load_df['datetime']
load_df = load_df.drop(['datetime'], axis=1)

# #Drop unrelevant columns & all households with more than 1000 gaps in the load
# load_df = load_df.drop(['datetime', 'SOM', 'leverende klanten', 'niet leverenden',
#                         'Klant 1', 'Klant 2', 'Klant 3', 'Klant 4', 'Klant 5',
#                         'Klant 8', 'Klant 9', 'Klant 10', 'Klant 11', 'Klant 14' ,
                        'Klant 16', 'Klant 21', 'Klant 24', 'Klant 27', 'Klant 28',
#                         'Klant 29', 'Klant 30', 'Klant 31', 'Klant 36', 'Klant 37',
#                         'Klant 40', 'Klant 41', 'Klant 44', 'Klant 45', 'Klant 47',
#                         'Klant 48', 'Klant 49', 'Klant 50', 'Klant 51', 'Klant 55',
#                         'Klant 57', 'Klant 59', 'Klant 63', 'Klant 64', 'Klant 65',
#                         'Klant 66', 'Klant 67', 'Klant 69', 'Klant 72', 'Klant 73',
#                         'Klant 74', 'Klant 77', 'Klant 78', 'Klant 80', 'Klant 82'], axis =
     1)

# #Drop unrelevant columns & all households with more than 1000 gaps in the load
# load_df = load_df.drop(['datetime', 'Klant 2', 'Klant 4', 'Klant 5', 'Klant 24',
#                         'Klant 36', 'Klant 37', 'Klant 40', 'Klant 41', 'Klant 44',
#                         'Klant 45', 'Klant 49', 'Klant 50', 'Klant 51', 'Klant 59',
#                         'Klant 64', 'Klant 65', 'Klant 66', 'Klant 67',
#                          'Klant 72', 'Klant 73', 'Klant 74', 'Klant 78',], axis = 1)


#Add households that should be used 2 times to data set
load_df['Klant 13 copy'] = load_df['Klant 13']
load_df['Klant 15 copy'] = load_df['Klant 15']
load_df['Klant 20 copy'] = load_df['Klant 20']
load_df['Klant 22 copy'] = load_df['Klant 22']
load_df['Klant 26 copy'] = load_df['Klant 26']

# #Remove nan values from gaps in data for 0 values (in original data set load never = 0)
# load_df = load_df.fillna(0)

# #Drop unrelevant columns & all households with gaps
# #load_df = load_df.drop(['datetime', 'SOM', 'leverende klanten', 'niet leverenden',
# # #                         'Klant 1', 'Klant 2', 'Klant 3', 'Klant 4', 'Klant 6',
# # #                         'Klant 9', 'Klant 17', 'Klant 21', 'Klant 29', 'Klant 30',
# # #                         'Klant 31', 'Klant 35', 'Klant 40', 'Klant 42', 'Klant 47',
# # #                         'Klant 48', 'Klant 53', 'Klant 55', 'Klant 59', 'Klant 63',
# # #                         'Klant 69', 'Klant 71', 'Klant 72', 'Klant 74', 'Klant 79',
# # #                         'Klant 80', 'Klant 81', 'Klant 82'], axis = 1)

# #create list with datetimes and numpy array with load data
dates_df = pd.to_datetime(dates_df, format='%d-%m-%Y %H:%M')
dates_times = dates_df.to_list()
load_data = load_df.to_numpy()

#%%Remove gaps by average load of 36 other households
n, households = load_data.shape

for j in range(households):
    for i in range(n):
          if (load_data[i,j] == 0):
            load_data[i,j] = np.mean(load_data[i])
```

```
76
77 #%% Aggregating and plotting 37 selected households
78 agg_data = np.zeros([n,1])
79 agg_data = np.sum(load_data, axis = 1)
80
81 # plt.plot(dates_times[960:1056], agg_data[960:1056])
82 # plt.plot(dates_times[672:768], agg_data[672:768])
83 plt.plot(dates_times[1920:2016], agg_data[1920:2016])
84
85
86 #%% Clustering data into weekdays and weekend-/holidays
87 #Liander operates mostly in northern regions in the Netherlands, therefore the
88 #holiday dates of the northern regions from 2013 are used
89
90 #Specifying all holidays
91 Christmas_start = dt.date(2013,1,1) #Christmas holiday is everywhere the same in NL
92 Christmas_end = dt.date(2013,1,7)
93
94 Summer_start = dt.date(2013,7,13) #These are the dates of high school summer vacation in the
       northern region of NL
95 Summer_end = dt.date(2013,9,2)
96
97 Goodfriday = dt.date(2013,3,29)
98 Easter = dt.date(2013,4,1)
99 Queensday = dt.date(2013,4,30)
100 Ascension1 = dt.date(2013,5,9)
101 Ascension2 = dt.date(2013,5,10)
102 Pentecost1 = dt.date(2013,5,19)
103 Pentecost2 = dt.date(2013,5,20)
104
105 Christmas2_start = dt.date(2013,12,21)
106 Christmas2_end = dt.date(2014,1,1)
107
108 #Creating list with holiday dates form start and end dates
109 Christmas = pd.date_range(Christmas_start, periods=6).tolist()
110 Summer = pd.date_range(Summer_start, periods=51).tolist()
111 Christmas2 = pd.date_range(Christmas2_start, periods=11).tolist()
112
113 Holidays = [Goodfriday, Easter, Queensday, Ascension1, Ascension2, Pentecost1, Pentecost2]
114 Holidays = Holidays + Christmas + Summer + Christmas2
115
116 #Creating set with holidays and making them all type date
117 holiday_set = []
118 for day in Holidays:
119     if type(day) == dt.date:
120         holiday_set.append(day)
121     else:
122         holiday_set.append(day.to_pydatetime().date())
123
124 #Adding weekends to holiday set
125 start = dt.datetime(2013, 1, 1)
126 weekend_index = set([5, 6])
127 days = pd.date_range(start, periods=365).tolist()
128
129 for day in days:
130     if day.weekday() in weekend_index:
131         holiday_set.append(day)
132
133 #Creating array that distinguishes between work- and holidays
134 n, houdeholds = load_data.shape
135 workday = []
136
137 for i in range(n):
138     if dates_times[i].date() in holiday_set:
139         workday.append(0)
140     else:
141         workday.append(1)
142
143 #Splitting load data set into weekdays and weekend days
144 # n, households = load_data.shape
145 # load_workday = np.zeros([1,37])
```

```
# load_holiday = np.zeros([1,37])
# date_time_workday = []
# date_time_holiday = []

# for i in range(n):
#     if dates_times[i].date() in holiday_set:
#         load_holiday = np.concatenate((load_holiday, [load_data[i]]), axis=0)
#         date_time_holiday.append(dates_times[i])

#     else:
#         load_weekday = np.concatenate((load_workday, [load_data[i]]), axis=0)
#         date_time_workday.append(dates_times[i])

# load_holiday = np.delete(load_holiday, 0, axis=0) #Remove row with 0's from initialization
# load_workday = np.delete(load_weekday, 0, axis=0) #Remove row with 0's from initialization


#%%Creating final data set and converting to excel sheet
final_data = pd.DataFrame(data=load_data)
final_data.insert(0, 'DateTime', dates_times)
final_data.insert(1, 'Workday', workday)
final_data.insert(2, 'Total load', agg_data)
final_data.to_excel('Final load data.xlsx')

#%%Creating separate work and weekend data set

work_dates = []
work_indices = []   #The indices are used later to link the correct weather data
work_load = np.zeros(1)
holiday_dates = []
holiday_indices = []
holiday_load = np.zeros(1)

for i in range(len(agg_data)):
    if workday[i] == 0:
        holiday_dates.append(dates_times[i])
        holiday_load = np.append(holiday_load, agg_data[i])
        holiday_indices.append(i)

    else:
        work_dates.append(dates_times[i])
        work_load = np.append(work_load, agg_data[i])
        work_indices.append(i)

#Removing first zeros from initialization
work_load = np.delete(work_load, 0)
holiday_load = np.delete(holiday_load, 0)

d_work = {'DateTime': work_dates, 'Total load': work_load, 'Indices': work_indices}
work_load_df = pd.DataFrame(data=d_work)
work_load_df.to_excel('Workday_load.xlsx')
d_holiday = {'DateTime': holiday_dates, 'Total load': holiday_load, 'Indices':
    holiday_indices}
holiday_load_df = pd.DataFrame(data=d_holiday)
holiday_load_df.to_excel('Holiday_load.xlsx')
```

A.1.2. PV data updating

```
# -*- coding: utf-8 -*-
"""
@author: Daan van Dingstee, Ruben Eland

Description: This is part 1 of two scripts that update a PV data set
to prepare it for training ML models
"""

import numpy as np
import pandas as pd
from datetime import datetime
import datetime as dt
from random import randint
```

```

#%%First: creating pandas dataframe from original data set

PVData=pd.read_csv('db_export_ch202-2021-01-23-2022-01-23.csv')
PVData=PVData.drop('id', 1)
PVData=PVData.drop('idc', 1)
PVData=PVData.drop('scantime', 1)
PVData=PVData.drop('starttime', 1)

PVData['Date'] = pd.to_datetime(PVData['Date'], format='%Y-%m-%d %H:%M:%S')

#%%Adding up all values per 15 minutes, then setting all the used values to 'used'
#and setting the last value to the total of the last 15 minutes

q1=0
q2=0
q3=0
q4=0

for i in range(PVData.shape[0]):

  if PVData.wh[i]<0:
    PVData.wh[i]=0      #setting negative values equal to 0

  if i % 100 ==0:
    print(i)
  if PVData.wh[i]!='used':
    if PVData.Date[i].minute<15:
       q1=q1+PVData.wh[i]
       PVData.wh[i]='used'
       if PVData.Date[i+1].minute>=15:
           PVData.wh[i]=q1
           q1=0
    elif PVData.Date[i].minute<30:
       q2=q2+PVData.wh[i]
       PVData.wh[i]='used'
       if PVData.Date[i+1].minute>=30:
         PVData.wh[i]=q2
         q2=0
    elif PVData.Date[i].minute<45:
       q3=q3+PVData.wh[i]
       PVData.wh[i]='used'
       if PVData.Date[i+1].minute>=45:
         PVData.wh[i]=q3
         q3=0
    else:
      q4=q4+PVData.wh[i]
      PVData.wh[i]='used'
      if PVData.Date[i+1].minute<15:
        PVData.wh[i]=q4
        q4=0

#deleting all used values
PVData = PVData[PVData.wh != 'used']

#setting seconds to 0
for i in range(PVData.shape[0]):
  PVData.Date[i]=PVData.Date[i].replace(second=0)

d = {'Date': PVData['Date'], 'wh': PVData['wh']}
PVTemp = pd.DataFrame(data = d)
PVTemp.to_excel('PVDataAangepast2.xlsx')
```

```
# -*- coding: utf-8 -*-
"""
@author: Ruben Eland, Daan van Dingstee

Description: This is part 1 of two scripts that update a PV data set
to prepare it for training ML models
"""

```

```
import numpy as np
import pandas as pd
import datetime as dt
from random import randint

#%% Downloading data set and creating dataframe
filename = 'C:/Users/raela/OneDrive/Elektro/year 4/BAP/Thesis/Data sets/PVDataAangepast2.xlsx'
PV_df = pd.read_excel(filename, sheet_name = 0)

PV_df = PV_df.drop(['Unnamed: 0'], axis=1)
dates_temp = PV_df['Date'].tolist()
dates = []
dates_times = []

for time in dates_temp:
    dates_times.append(time.strftime("%Y-%m-%d, %H:%M")) #Converting dates_times to strings


#%% Removing outliers: removing gaps

#First all gaps are removed. This is done by checking if every date in PVData
#has 96 indices (24 hrs x 15 mins). If not, they are filled with 0 values

start = dt.datetime(2021, 1, 23, 00, 14,)
numIntervals = 35039
all_dates_times = [(start + dt.timedelta(minutes=15*x)).strftime("%Y-%m-%d, %H:%M") for x in range(numIntervals)]
indices = []

#Checking where the gaps are, by looping all dates_times that should be
#in the date_times column from the data set and checking if they are there.
#Since there are some data points where the time is for example 0012 or 0009
#in stead of 0014, all points in the last quarter have to be checked.
for time in all_dates_times:
    time_dt = dt.datetime.strptime(time, "%Y-%m-%d, %H:%M")
    time_1 = (time_dt - dt.timedelta(minutes=1)).strftime("%Y-%m-%d, %H:%M")
    time_2 = (time_dt - dt.timedelta(minutes=2)).strftime("%Y-%m-%d, %H:%M")
    time_3 = (time_dt - dt.timedelta(minutes=3)).strftime("%Y-%m-%d, %H:%M")
    time_4 = (time_dt - dt.timedelta(minutes=4)).strftime("%Y-%m-%d, %H:%M")
    time_5 = (time_dt - dt.timedelta(minutes=5)).strftime("%Y-%m-%d, %H:%M")
    time_6 = (time_dt - dt.timedelta(minutes=6)).strftime("%Y-%m-%d, %H:%M")
    time_7 = (time_dt - dt.timedelta(minutes=7)).strftime("%Y-%m-%d, %H:%M")
    time_8 = (time_dt - dt.timedelta(minutes=8)).strftime("%Y-%m-%d, %H:%M")
    time_9 = (time_dt - dt.timedelta(minutes=9)).strftime("%Y-%m-%d, %H:%M")
    time_10 = (time_dt - dt.timedelta(minutes=10)).strftime("%Y-%m-%d, %H:%M")
    time_11 = (time_dt - dt.timedelta(minutes=11)).strftime("%Y-%m-%d, %H:%M")
    time_12 = (time_dt - dt.timedelta(minutes=12)).strftime("%Y-%m-%d, %H:%M")
    time_13 = (time_dt - dt.timedelta(minutes=13)).strftime("%Y-%m-%d, %H:%M")

    if (time not in dates_times and time_1 not in dates_times and time_2 not in dates_times
        and time_3 not in dates_times and time_4 not in dates_times
        and time_5 not in dates_times and time_6 not in dates_times
        and time_7 not in dates_times and time_8 not in dates_times
        and time_9 not in dates_times and time_10 not in dates_times
        and time_11 not in dates_times and time_12 not in dates_times
        and time_13 not in dates_times):
        indices.append(all_dates_times.index(time))

#Adding extra hour of winter time because it is in original PV data set
start2 = dt.datetime(2021, 10, 31, 2, 14)
for x in range(4):
    winter_time = (start2 + dt.timedelta(minutes=15*x)).strftime("%Y-%m-%d, %H:%M")
    all_dates_times.insert((21128 + x), winter_time)

final_dates_times = []
wh_temp = np.copy(wh)
for i in range(len(all_dates_times)):
    final_dates_times.append(all_dates_times[i])
    if i in indices:
        wh_temp = np.insert(wh_temp, i, 'nan')
```

```
78
79 # Removing extra hour of winter time because it is not in the weather data set
80 pop_i = 21132
81 for i in range(4):
82     final_dates_times.pop(pop_i)
83     wh_temp = np.delete(wh_temp, pop_i)
84
85 def check_gap_removal(indices):
86     r = randint(25000, 34000)
87     if final_dates_times[r] == dates_times[r - diff_i]:
88         print('gaps in dates succesfully removed!')
89     else:
90         print('error in removing gaps from dates!')
91     if wh_temp[r] == wh[r - diff_i]:
92         print('gaps in wh succesfully removed!')
93     else:
94         print('error in removing gaps from wh!')
95
96 diff_i = len(indices) - 4
97 print('Number of gaps is: ', diff_i)
98 check_gap_removal(indices)
99
100
101
102 # Removing outliers: replacing nan values from gaps, removing offset
103
104 nan_i = np.asarray(np.where(np.isnan(wh_temp)))
105 for i in range(len(nan_i)):
106     wh_temp[nan_i] = wh_temp[nan_i - 96] #value of the day before
107
108 #Still nan values remaining -> try with gen from 2 days before
109 nan_i2 = np.asarray(np.where(np.isnan(wh_temp)))
110 for i in range(len(nan_i2)):
111     wh_temp[nan_i2] = wh_temp[nan_i2 - 192] #value of 2 days before
112
113 if np.asarray(np.where(np.isnan(wh_temp))).size == 0:
114     print('all gaps are filled!')
115 else:
116     print('still unfilled gaps in wh!')
117
118 #now all gaps are filled!
119
120 # d = {'Date': final_dates_times, 'wh': wh_temp}
121 # PVWOGaps = pd.DataFrame(data = d)
122 # PVWOGaps.to_excel('PVDataWOGaps.xlsx')
123
124 #Rmoving offset from part of the data set
125
126 #It is assumed that offset starts at 2021-08-13, 20:44 (index = 19474 in wh_temp
127 # and final_dates_times), because before every night wh = 0 and at 2021-08-12
128 # 20:44 the generation was 0.06. From this point on the offset is almost always present
129 # and increases from roughly 4.2 up to 6.5.
130 # !The offset is not present at some indices, f.e. (33402 until 33433), (26411 until 26425),
131 # these values will be ignored, as the values in the night will be set to 0 later on
132 # and the remaining set of values will be so small that its effect will be marginal.
133
134 #Plan to remove offset:
135 #1: take average offset between 00:00 and 04:00 (check if no value above
136 # max offset value is taken)
137 #2: subtract the offset from each night from night & day before
138
139 first_offset = 1
140 count = 0
141 ave_offset = 0
142 for i in range(19487, 35040, 1):
143     if count == 96:
144         count = 0
145     if count == 16: #count average offset between 00:00 and 04:00
146         ave_array = wh_temp[(i-16):i]
147         ignore = ave_array > 7
148         ave_array = np.delete(ave_array, ignore)
```

```
        ave_offset = np.mean(ave_array)
        if first_offset == 1:
            wh_temp[i-29:i-16] = wh_temp[i-29:i-16] - ave_offset
            first_offset = 0

        wh_temp[i-112:i-16] = wh_temp[i-112:i-16] - ave_offset

    if i == 35039: #Remove last average offset from last day
        wh_temp[i-96:i] = wh_temp[i-96:i] - ave_offset

    count += 1

if np.asarray(np.where(wh_temp < (-3))).size == 0:
    print('succesfully eliminated offset!')
else:
    print('errors when eliminating offset: ',
          np.asarray(np.where(wh_temp < (-2))).size,
           'values are below -2 and thus they had no offset!')

# d = {'Date': final_dates_times, 'wh': wh_temp}
# PVWOoffset = pd.DataFrame(data = d)
# PVWOoffset.to_excel('PVDataWOoffset.xlsx')

#%%Removing outliers: too large values

#From analyzing weather data set we learned: a generated power above 400 Wh can
#seen as outlier. All outliers can be removed by taking the average between values
#before and after outlier, the first value that is not an outlier itself is taken

max = 400

print()

for i in range(34245):
    if  wh_temp[i] > max:
        if wh_temp[i-1] < max and wh_temp[i+1] < max:
          wh_temp[i] = (wh_temp[i-1] + wh_temp[i+1])/2
        elif wh_temp[i-2] < max and wh_temp[i+1] < max:
          wh_temp[i] = (wh_temp[i-2] + wh_temp[i+1])/2
        elif wh_temp[i-2] < max and wh_temp[i+2] < max:
          wh_temp[i] = (wh_temp[i-2] + wh_temp[i+1])/2
        elif wh_temp[i-2] < max and wh_temp[i+2] < max:
          wh_temp[i] = (wh_temp[i-2] + wh_temp[i+1])/2
        elif wh_temp[i-2] < max and wh_temp[i+3] < max:
          wh_temp[i] = (wh_temp[i-2] + wh_temp[i+3])/2
        elif wh_temp[i-3] < max and wh_temp[i+3] < max:
          wh_temp[i] = (wh_temp[i-3] + wh_temp[i+3])/2


#Run two times, because for some reason some max values are missed if only run 1 time
for i in range(34245):
    if  wh_temp[i] > max:
        if wh_temp[i-1] < max and wh_temp[i+1] < max:
          wh_temp[i] = (wh_temp[i-1] + wh_temp[i+1])/2
        elif wh_temp[i-2] < max and wh_temp[i+1] < max:
          wh_temp[i] = (wh_temp[i-2] + wh_temp[i+1])/2
        elif wh_temp[i-2] < max and wh_temp[i+2] < max:
          wh_temp[i] = (wh_temp[i-2] + wh_temp[i+1])/2
        elif wh_temp[i-2] < max and wh_temp[i+2] < max:
          wh_temp[i] = (wh_temp[i-2] + wh_temp[i+1])/2
        elif wh_temp[i-2] < max and wh_temp[i+3] < max:
          wh_temp[i] = (wh_temp[i-2] + wh_temp[i+3])/2
        elif wh_temp[i-3] < max and wh_temp[i+3] < max:
          wh_temp[i] = (wh_temp[i-3] + wh_temp[i+3])/2

print('max wh value is: ', np.max(wh_temp))

start = dt.datetime(2021, 1, 23, 00, 15,)
numIntervals = 35039
correct_dates_times = [(start + dt.timedelta(minutes=15*x)) for x in range(numIntervals)]

```

```
d = {'Date': correct_dates_times, 'wh': wh_temp}
PVFinal = pd.DataFrame(data = d)
PVFinal.to_excel('PVDataFinal.xlsx')


#%%What still should happen in combined PV & weather data set:

#Set all wh = 0 where irr = 0
#Remove all values where irr is not 0 and wh = 0
#Scikit outlier detection
```

A.1.3. EV data updating

```
# -*- coding: utf-8 -*-
"""
@author: Daan van Dingstee, Ruben Eland
"""

import numpy as np
import pandas as pd
from datetime import datetime
from numpy import mean
from numpy import std

total=EVData.sum(axis='columns')
EVData['Total']=total
Time=EVData.Time
EVData=EVData.drop('Time', 1)
EVData=EVData/6600
EVData.astype(int)
EVData['Time']=Time

EVData=EVData.drop('Vehicle 1', 1)
EVData=EVData.drop('Vehicle 2', 1)
EVData=EVData.drop('Vehicle 3', 1)
EVData=EVData.drop('Vehicle 4', 1)
EVData=EVData.drop('Vehicle 5', 1)
EVData=EVData.drop('Vehicle 6', 1)
EVData=EVData.drop('Vehicle 7', 1)
EVData=EVData.drop('Vehicle 8', 1)
EVData=EVData.drop('Vehicle 9', 1)
EVData=EVData.drop('Vehicle 10', 1)
EVData=EVData.drop('Vehicle 11', 1)
EVData=EVData.drop('Vehicle 12', 1)
EVData=EVData.drop('Vehicle 13', 1)
EVData=EVData.drop('Vehicle 14', 1)
EVData=EVData.drop('Vehicle 15', 1)
EVData=EVData.drop('Vehicle 16', 1)
EVData=EVData.drop('Vehicle 17', 1)
EVData=EVData.drop('Vehicle 18', 1)
EVData=EVData.drop('Vehicle 19', 1)
EVData=EVData.drop('Vehicle 20', 1)
EVData=EVData.drop('Vehicle 21', 1)
EVData=EVData.drop('Vehicle 22', 1)
EVData=EVData.drop('Vehicle 23', 1)
EVData=EVData.drop('Vehicle 24', 1)
EVData=EVData.drop('Vehicle 25', 1)
EVData=EVData.drop('Vehicle 26', 1)
EVData=EVData.drop('Vehicle 27', 1)
EVData=EVData.drop('Vehicle 28', 1)
EVData=EVData.drop('Vehicle 29', 1)
EVData=EVData.drop('Vehicle 30', 1)
EVData=EVData.drop('Vehicle 31', 1)
EVData=EVData.drop('Vehicle 32', 1)
EVData=EVData.drop('Vehicle 33', 1)
EVData=EVData.drop('Vehicle 34', 1)
EVData=EVData.drop('Vehicle 35', 1)
EVData=EVData.drop('Vehicle 36', 1)
EVData=EVData.drop('Vehicle 37', 1)
EVData=EVData.drop('Vehicle 38', 1)
EVData=EVData.drop('Vehicle 39', 1)
```

```
EVData=EVData.drop('Vehicle 40', 1)
EVData=EVData.drop('Vehicle 41', 1)
EVData=EVData.drop('Vehicle 42', 1)
EVData=EVData.drop('Workday', 1)
EVData.head(5)

data['Time']=data['Time'].dt.time

EVData.to_excel('EVDataAangepastfinal.xlsx')
```

A.2. Python Code: Importing Model types

A.2.1. Load forecasting

```
# -*- coding: utf-8 -*-
"""
@author: Daan van Dingstee, Ruben Eland
"""


import os
#Importing datasets
# clone the github repository if it not already cloned
if not os.path.exists(os.getcwd()+'/Smart-Sustainable-Grids-BAP'):
    !git clone https://github.com/PieterGo/Smart-Sustainable-Grids-BAP.git
else:
    print('"/Smart-Sustainable-Grids-BAP" already exists')

from sklearn.decomposition import PCA
from sklearn.metrics import explained_variance_score
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import pandas as pd
from datetime import datetime
from sklearn import preprocessing
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
import seaborn as sns
from scipy.linalg import svd
from sklearn import metrics
from sklearn import datasets, svm
from sklearn.neural_network import MLPRegressor
from sklearn.datasets import make_regression
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

#converting dattasets from excel to dataframe and dropping the unnecessary columns
LoadData = pd.read_excel('Smart-Sustainable-Grids-BAP/Final load data.xlsx')
WeatherData=pd.read_excel('Smart-Sustainable-Grids-BAP/weerdata_aangepast_final.xlsx')

print(LoadData.shape)
print(WeatherData.shape)

#converting DateTime into string
LoadData['DateTime']=LoadData['DateTime'].dt.time

for i in range(LoadData.shape[0]):
  LoadData.DateTime[i] = LoadData.DateTime[i].strftime("%H%M%S")

#combine the load and weather data
AllData = pd.concat([LoadData, WeatherData], axis=1)

y=AllData.TotalLoad
features=AllData.drop('TotalLoad', 1)
#split the data in 70% train and 30% test
```

```
X_train, X_test, y_train, y_test = train_test_split(features, y, test_size=0.3,shuffle=True,
    random_state=42)
y_train=y_train.ravel()
y_test=y_test.ravel()

#standard normalize the data
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

A.2.2. PV forecasting

```
# -*- coding: utf-8 -*-
"""
@author: Daan van Dingstee, Ruben Eland
"""

import os

# clone the github repository if it not already cloned
if not os.path.exists(os.getcwd()+'/Smart-Sustainable-Grids-BAP'):
    !git clone https://github.com/PieterGo/Smart-Sustainable-Grids-BAP.git
else:
    print('"/Smart-Sustainable-Grids-BAP" already exists')

from sklearn.decomposition import PCA
from sklearn.metrics import explained_variance_score
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import pandas as pd
from datetime import datetime
from sklearn import preprocessing
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
import seaborn as sns
from scipy.linalg import svd
from sklearn import metrics
from sklearn import datasets, svm
from sklearn.neural_network import MLPRegressor
from sklearn.datasets import make_regression
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

PVData = pd.read_excel('Smart-Sustainable-Grids-BAP/PVDataFinal.xlsx')
WeatherData=pd.read_excel('Smart-Sustainable-Grids-BAP/Weerdata_PV.xlsx')
PVData=PVData.drop('Unnamed: 0', axis=1)
WeatherData=WeatherData[['Wind', 'Temp', 'Irr', 'View', 'Clouds', 'Humidity', 'Fog', 'Rain',
    'Snow', 'Thunder']]

print(PVData.shape)
print(WeatherData.shape)

# Combining PV data & weather data

PVData['Date'] = pd.to_datetime(PVData['Date'], format='%Y-%m-%d, %H:%M')


PVData['Date']=PVData['Date'].dt.time

for i in range(PVData.shape[0]):
 PVData.Date[i] = PVData.Date[i].strftime("%H%M")
 #increase data to amount of pv generation our neighbourhood would have
 PVData.wh[i]=PVData.wh[i]*168.285

PVData['season']=season
```

```
AllData = pd.concat([PVData, WeatherData], axis=1)

# Removing outliers in combined PV & weather data set

#1: Set all wh = 0 where irr = 0
#2: Remove all values where irr is not 0 and wh = 0
#3: Scikit outlier detection

wh = AllData['wh'].to_numpy()
irr = AllData['Irr'].to_numpy()

#1: Set all wh = 0 where irr = 0
wh[irr == 0] = 0

#2: Remove all values where irr is not 0 and wh = 0
indices = []
for i in range(len(wh)):
  if wh[i] == 0 and irr[i] > 1:
    indices.append(i)

AllData = AllData.drop(indices)

y=AllData.wh
features=AllData.drop('wh', 1)

X_train, X_test, y_train, y_test = train_test_split(features, y, test_size=0.3,shuffle=True,
    random_state=42) #split the data in 70% train and 30% test
y_train=y_train.ravel()
y_test=y_test.ravel()

#standard normalize the data
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

A.2.3. EV forecasting

```
# -*- coding: utf-8 -*-
"""
@author: Daan van Dingstee, Ruben Eland
"""

import os

# clone the github repository if it not already cloned
if not os.path.exists(os.getcwd()+'/Smart-Sustainable-Grids-BAP'):
    !git clone https://github.com/PieterGo/Smart-Sustainable-Grids-BAP.git
else:
    print('"/Smart-Sustainable-Grids-BAP" already exists')

from sklearn.decomposition import PCA
from sklearn.metrics import explained_variance_score
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import pandas as pd
from datetime import datetime
from sklearn import preprocessing
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
import seaborn as sns
from scipy.linalg import svd
from sklearn import metrics
from sklearn import datasets, svm
from sklearn.neural_network import MLPRegressor
from sklearn.datasets import make_regression
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
```

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

data = pd.read_excel('Smart-Sustainable-Grids-BAP/Final EV data.xlsx')

y=data.Total
features=data.drop('Total', 1)
for i in range(features.shape[0]):
  features.Time[i] = features.Time[i].strftime("%H%M%S")

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, y, test_size=0.3,shuffle=True,
    random_state=42) #split the data in 70% train and 30% test
y_train=y_train.ravel()
y_test=y_test.ravel()

from sklearn import preprocessing
#standard normalize the data
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

A.3. Python Code: Training and testing the models

```
# -*- coding: utf-8 -*-
"""
@author: Daan van Dingstee, Ruben Eland
"""

from sklearn.decomposition import PCA
from sklearn.metrics import explained_variance_score
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import pandas as pd
from datetime import datetime
from sklearn import preprocessing
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
import seaborn as sns
from scipy.linalg import svd
from sklearn import metrics
from sklearn import datasets, svm
from sklearn.neural_network import MLPRegressor
from sklearn.datasets import make_regression
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


#PCA
X = features
X = scaler.transform(X)
N, M = X.shape
X_PCA = X - np.ones((N,1))*X.mean(axis=0) #subtract the mean
U, S, Vh = svd(X_PCA, full_matrices = False)

#Compute variance explained by principal components
rho = (S*S) / (S*S).sum()

threshold = 0.90

# Plot variance explained
fig = plt.figure()
plt.plot(range(1,len(rho)+1),rho,'x-')
plt.plot(range(1,len(rho)+1),np.cumsum(rho),'o-')
```

```
plt.plot([1,len(rho)],[threshold, threshold],'k--')
plt.title('Variance explained by principal components');
plt.xlabel('Principal component');
plt.ylabel('Variance explained');
plt.legend(['Individual','Cumulative','Threshold = 0.9'])
plt.grid()
plt.show()

#PCA with ExtraTreesRegressor
model = ExtraTreesRegressor()
model.fit(X, y)
print(model.feature_importances_)



#Finding the best RFR model
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}


# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 50 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter =
    50, cv = 3, verbose=0, random_state=42, n_jobs = -1)
# Fit the random search model
rf_random.fit(X_train,y_train)
rf_random.best_params_

#see improvement over basemodel
base_model = RandomForestRegressor()
base_model.fit(X_train, y_train)
y_predictbase=base_model.predict(X_test)
print("R^2=", r2_score(y_test, y_predictbase))
best_random = rf_random.best_estimator_
y_predictbest=best_random.predict(X_test)
print("R^2=", r2_score(y_test, y_predictbest))


#Finding the best SVM model
tol= [float(x) for x in np.linspace(0.0001, 0.9, num = 100)]
# tolerance
c=[float(x) for x in np.linspace(0.1, 5, num = 100)]
# Minimum number of samples required to split a node
epsilon=[float(x) for x in np.linspace(0.01, 1, num = 100)]
# Method of selecting samples for training each tree
# Create the random grid
random_grid2 = {'tol':tol,
               'C': c,
               'epsilon': epsilon}

```

```
117
118 svr = svm.SVR()
119 # Random search of parameters, using 3 fold cross validation,
120 # search across 50 different combinations, and use all available cores
121 svr_random = RandomizedSearchCV(estimator = svr, param_distributions = random_grid2, n_iter =
      50, cv = 3, verbose=0, random_state=42, n_jobs = -1)
122 # Fit the random search model
123 svr_random.fit(X_train,y_train)
124 svr_random.best_params_
125
126 #see improvement over basemodel
127 base_model2 = svm.SVR()
128 base_model2.fit(X_train, y_train)
129 y_predictbase2=base_model2.predict(X_test)
130 print("R^2=", r2_score(y_test, y_predictbase2))
131 best_random2 = svr_random.best_estimator_
132 y_predictbest2=best_random2.predict(X_test)
133 print("R^2=", r2_score(y_test, y_predictbest2))
134
135
136 #Finding the best ANN model
137 hidden_layer_sizes= [int(x) for x in np.linspace(1, 100, num = 50)]
138 #Number of hidden layers
139 activation = ['identity', 'logistic', 'tanh', 'relu']
140 #different kinds of activation for the layers
141 alpha=[float(x) for x in np.linspace(0.00001, 0.001, num = 50)]
142 #alpha
143 learning_rate=['constant', 'invscaling', 'adaptive']
144 #different kinds of learning rate evolution
145 learning_rate_init=[float(x) for x in np.linspace(0.01, 0.00001, num = 20)]
146 #different learning rates
147 tol= [float(x) for x in np.linspace(0.00001, 0.001, num = 20)]
148 #tolerance
149 max_iter=[int(x) for x in np.linspace(2, 1000, num = 50)]
150 #max epochs
151
152 random_grid3 = {'tol':tol,
153                 'max_iter': max_iter,
154                 'alpha': alpha,
155                 'hidden_layer_sizes': hidden_layer_sizes,
156                 'activation': activation,
157                 'learning_rate': learning_rate,
158                 'learning_rate_init':learning_rate_init}
159
160
161 MLP = MLPRegressor()
162 # Random search of parameters, using 3 fold cross validation,
163 # search across 50 different combinations, and use all available cores
164 MLP_random = RandomizedSearchCV(estimator = MLP, param_distributions = random_grid3, n_iter =
      50, cv = 3, verbose=0, random_state=42, n_jobs = -1)
165 # Fit the random search model
166 MLP_random.fit(X_train,y_train)
167 MLP_random.best_params_
168
169 #see improvement over basemodel
170 base_model3 = MLPRegressor()
171 base_model3.fit(X_train, y_train)
172 y_predictbase3=base_model3.predict(X_test)
173 print("R^2=", r2_score(y_test, y_predictbase3))
174 best_random3 = MLP_random.best_estimator_
175 y_predictbest3=best_random3.predict(X_test)
176 print("R^2=", r2_score(y_test, y_predictbest3))
177
178
179 #calculate cross validation
180 cv = KFold(n_splits=5, random_state=1, shuffle=True)
181
182 # evaluate models
183 scores1 = cross_val_score(best_random, features, y, scoring='r2', cv=cv, n_jobs=-1)
184 # report performance
185 print('Accuracy: %.3f (%.3f)' % (mean(scores1), std(scores1)))
```

```
print(scores1)

# evaluate models
scores2 = cross_val_score(best_random2, features, y, scoring='r2', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores2), std(scores2)))
print(scores2)

# evaluate models
scores3 = cross_val_score(best_random3, features, y, scoring='r2', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores3), std(scores3)))
print(scores3)
```

A.4. Matlab Code: interpolating Weather Data

```
% get irradiance per hour
weerdata_2013 = readmatrix('C:\Users\vandi\Desktop\BAP\weerdata_2021.xlsx');
data.date = weerdata_2013(:,1);
data.hour = weerdata_2013(:,2);
data.wind= weerdata_2013 (:,3);
data.temp= weerdata_2013 (:,6);
data.sun = weerdata_2013 (:,9);
data.irr = weerdata_2013 (:,10);
data.rain= weerdata_2013 (:,13);
data.cloud= weerdata_2013(:,16);

%interpolate
time = [0:1:8759];
interpolated = [0:0.25:8759];
data.date_15 =  interp1(time,data.date,interpolated);
data.hour_15 =  interp1(time,data.hour,interpolated);
data.wind_15=  interp1(time,data.wind,interpolated);
data.temp_15=  interp1(time,data.temp,interpolated);
data.sun_15 =  interp1(time,data.sun,interpolated);
data.irr_15 =  interp1(time,data.irr,interpolated);
data.rain_15=  interp1(time,data.rain,interpolated);
data.cloud_15=  interp1(time,data.cloud,interpolated);

data.date_15 = data.date_15(:);
data.hour_15 = data.hour_15(:);
data.wind_15= data.wind_15(:);
data.temp_15= data.temp_15(:);
data.sun_15 = data.sun_15(:);
data.irr_15 =data.irr_15 (:);
data.rain_15= data.rain_15(:);
data.cloud_15=data.cloud_15(:);
% write weather data back to excell in sheet 3
writematrix(data.date_15, 'weerdata_aangepast_2021.xlsx', 'range', 'B2');
writematrix(data.hour_15, 'weerdata_aangepast_2021.xlsx', 'range', 'C2');
writematrix(data.wind_15, 'weerdata_aangepast_2021.xlsx', 'range', 'D2');
writematrix(data.temp_15, 'weerdata_aangepast_2021.xlsx', 'range', 'E2');
writematrix(data.sun_15, 'weerdata_aangepast_2021.xlsx', 'range', 'F2');
writematrix(data.irr_15, 'weerdata_aangepast_2021.xlsx', 'range', 'G2');
writematrix(data.rain_15, 'weerdata_aangepast_2021.xlsx', 'range', 'H2');
writematrix(data.cloud_15, 'weerdata_aangepast_2021.xlsx', 'range', 'I2');
```