UNIVERSITY OF PIRAEUS

DEPARTMENT OF DIGITAL SYSTEMS

MSc DIGITAL SYSTEMS AND SERVICES

MAJOR: BIG DATA AND ANALYTICS

Subject:

Evaluation of Machine Learning Models for Predicting the System Marginal Price of an Electricity System – Italian SMP Day-Ahead forecasting

Aristeidis Dardamanis

ME 2008

Supervisor: Dr.M.Philippakis

Athens 2022

Master Thesis

Summary

In a power market, electricity price forecasts are of great importance for market participants. The presence of nonlinearity, nonstationarity, and multiple seasonality in electricity prices renders price prediction a challenging task. Many studies have been conducted in this field, however more robust and up to date price forecast methods are required. Europe is a great and complex energy market comprised of many countries who exchange vast amounts of electricity on a daily basis. Italy is one of the most significant energy markets in the southern European area. In this research, three types of machine learning models were investigated with the aim of predicting as much accurately as possible the Italian SMP. Namely, one autoregressive, one MLP and three LSTM models were constructed in order to forecast the Italian Day-Ahead SMP at an hourly level. The predictions of each model were evaluated for a timeframe extending up to two weeks ahead through the calculation of the respective MAPE and RMSE metrices. A number of attributes were initially selected as candidates to train our machine learning models, which is also the key difference between other similar studies. Specifically, among the attributes initially selected, electricity prices of major European countries were included. This is crucial owe to the fact that Italy is now coupled with many neighbouring countries suggesting a much stronger interdependence with its borders. Three feature selection methods were applied to filter out any possible redundant attributes, the SFS, the Random Forest Regressor and the F_regression methods. The Hungarian SMP, the 24 lag SMP price (Italian SMP shifted by 24 hours in the past) and the Residual[1] attributes were determined as the most important. The dataset used covers in total the period of 2021/06/01 to 2021/10/22. Historical data were collected from various sources such as, GME, Nord Pool, Entso-e and Terna. The training and test samples were identical for each model in order to secure consistency. The evaluation results are promising especially for the Bidirectional LSTM and the Stacked LSTM. Namely, concerning a forecast of up to two weeks ahead MAPE and RMSE values of the Stacked LSTM and the Bidirectional LSTM were found to be 6.965 per cent, 19.628 Euro / MWh and 7.249 per cent, 20.237 Euro / MWh respectively. In addition, these proposed deep learning algorithms showed an improved performance when trained with a bigger sample of data. However, there are more risks adding to the forecast error, which originate from the uncertainty in the prediction of both the Hungarian SMP and the Residual features. On the grounds that Stacked LSTM produced the most prominent results, a sensitivity analysis was performed on it to check at what extent the model's performance would change. It was decided to alter by +15 per cent and by -15 per cent the values of HU. The performance of the model deteriorated and more specifically for a +15 per cent change MAPE and RMSE increased to 8.720 per cent and 23.648 Euro /MWh correspondingly, while for a -15 per cent change MAPE and RMSE increased to 8.856 per cent and 24.561 Euro / MWh respectively.

[1] It is adequately described in the Methodology section.

Table of Contents

Definitions

Attributes: These are input variables, also called features, working as predictors in a machine learning algorithm that affect a given outcome.
Back propagation algorithm: In machine learning, back propagation is a widely used algorithm for training feedforward neural networks. In the process of designing a neural network, back propagation utilizes the slope of the loss function relative to the weights of the network for a single input / output example. This function makes it possible to use tilt functions to train multilevel networks and update weights to minimize loss. The back propagation algorithm works by calculating the slope of the loss function with respect to each weight, from the chain rule, calculating the gradation for one level at a time, repeating the process backwards from the last level, to avoid unnecessary calculations of intermediates terms in the chain rule.
Training: Neural network training is the process of finding values for weights and bias, so that for a given set of input values, the respective calculated output values match well with the known, correct target values.
Validation: The validation dataset provides an unbiased assessment of the fit of a model to the training dataset, while coordinating the model hyperparameters (e.g., the number of hidden layers) in a neural network.
Cross Validation: The training set is divided into k sub sets (k fold cross validation). Each time a subset is removed from the training set to be used for verification and the training is performed with the remaining (k-1) subsets. The process is repeated k times and k trained networks are used for the final prediction (e.g., majority or average).
Testing: A test dataset is independent of the training dataset, and both should follow the same probability distribution. The main purpose of testing is to determine whether the trained model fits well to the test dataset or not. A better fitting of the training dataset as opposed to the test dataset usually points to over-fitting. Testing is therefore a set of examples used only to assess the performance of a trained model and to do this, the final model is used to predict classifications of examples in the test dataset.
Learning Rate: It is a regulated hyperparameter used in the training process of neural networks, which has a small positive value, often in the range between 0 and 1. The learning rate controls how quickly the model adapts to the problem. It is probably the most important hyper parameter for the model.
Momentum: Momentum is a coefficient that is applied to an extra term in the weights update. It is a simple technique that often improves both the speed and accuracy of a neural network training.
Epoch: One epoch is training the neural network with all the training data for one cycle. A forward pass and a backward pass together are counted as one pass. Thus, epoch is the number of passes of the entire training dataset the model has completed.
Time Series: A time series is a series of data points in chronological order. Most commonly, a time series is a sequence of data taken at successive equally spaced points in time.
SMP: SMP is the wholesale electricity price of a country, as this is determined through the respective Energy Exchange.
PUN: Is the national single wholesale electricity price of Italy, which is derived from the average of zonal prices in the Day-Ahead Market, weighted for total purchases and net of purchases for Pumped-Storage Units and of purchases by neighbouring countries' zones.
Geographical Zone: Represent a portion of the national grid. Geographical zones are northern Italy (NORD), central-northern Italy (CNOR), central-southern Italy (CSUD), southern Italy (SUD), Sicilia (SICI) and Sardegna (SARD).
Bidding Zone: Bidding Zone refers to the largest geographical area within which Market Participants are able to exchange energy without transmission capacity allocation.
CET: Central European Time is the time zone of most European countries including Italy, Germany, Hungary and France.
Congestion: Congestion refers to situations between contiguous market areas, in which the demand for transferring power exceeds the limits of the respective available transmission capacities. Namely, congestion between certain zones in Italy will lead in the formation of different wholesale electricity prices for these zones as a result.
be brought on line.

1 Introduction

Prior to electricity markets deregulation, power price fluctuations were not considerable and to a great extent were controlled by regulators. Electricity price evolution was directly dependent on the government's social and industrial policy, and price forecasting was predominantly driven by underlying costs such as, fuel prices and technological innovation [1]. Nowadays, deregulation has a great impact on the electricity market. Wholesale power prices are more volatile, entailing higher risks for the different stakeholders in the market. In a deregulated electricity market, the System Marginal Price (SMP) is determined via the intersection of the power supply and demand curves [2]. Due to price volatility, power generators have to cope with risks accruing from SMP determination, as they sell energy at variable wholesale prices, while their operating costs may be fixed. On the other hand, distributors/retailers face price risks as well in that they supply most of their costumers at an annual fixed tariff, when at the same time have to purchase electricity at SMP. As a result, accurate electricity price forecasting is an extremely important task for all electricity market players and namely for proper risk management [1]. Selling and buying physically electricity is carried out in the power exchanges mainly via the Spot Market (Day-Ahead, Intra-Day). Accurate Day-Ahead price forecast in the spot market helps power suppliers and generators and other market participants to adjust their bidding strategies in order to achieve maximum benefit.

An integrated electricity wholesale market was envisioned by the EU, with the aim of optimizing the overall welfare of market participants in the European electricity market. National Regulatory Authorities (NRAs) led a work stream on the EU Electricity Market Target Model in 2008 and came to an end in 2010. This work was preceded by a project, started in 2007 and led by Transmission System Operators (TSOs) and power exchanges, on how to organise the electricity market at different timeframes. The final outcome of such efforts was the adoption of the 'EU Target Model'. The EU Target Model foresees a specific cross-border market design for each of timeframe of the market as those move closer to the time electricity has to be actually delivered. These markets are: Forward, Day-Ahead, Intra-Day, and Balancing. For the Day-Ahead timeframe, the implicit allocation of cross-border transmission capacities through a single European price coupling process is provided, replacing explicit auctions. Implicit market coupling implies that all order books from power exchanges (all bids and offers) are to be aggregated and optimized in one algorithm that calculates prices and flows, subject to the available transmission capacity between market areas, leading to a single price. However, price differences can still occur due to bottlenecks between different market areas [3].

In recent years a number of studies have been conducted with regard to the prediction of the Day-Ahead SMP for different European countries. These studies could be classified into autoregressive models and neural network models. The vast majority of them don't take into account the SMP of neighbouring countries in respect to the under-examination country. Considering the EU Target Model implications, coupling between countries has led to higher interrelation among them making imperative for any new price forecasting attempt to take into account the respective countries' SMP. For example, Italy is a very significant energy market in the southern area of Europe, affecting to a great extent neighbouring countries' SMP such as, Greece, Malta, Montenegro, Slovenia, Austria, Switzerland and France, while being also affected by them. The Italian Day-Ahead Market (MGP) hosts most of the electricity sale and purchase transactions in Gestore Mercati Energetici (GME). In the MGP, hourly energy blocks are traded for the next day. Participants submit bids/asks, where they specify the quantity and the minimum/maximum price at which they are willing to sell/purchase. The MGP sitting opens at 8:00 a.m. of the ninth day before the day of delivery and closes at 12:00 p.m. of the day before the day of delivery. The results of the MGP are made known within 12:58 p.m. of the day before the day of delivery. Bids/asks are accepted after the closure of the market, based on the economic merit-order criterion and taking into account transmission capacity limits between zones. The price is determined, for each hour, by the intersection of the demand and supply curves and is differentiated from zone to zone when transmission capacity limits are saturated. The accepted demand bids pertaining to consuming units belonging to Italian geographical zones are valued at the "Prezzo Unico Nazionale" (PUN – national single price); this price is equal to the average of the prices of geographical zones, weighted for the quantities purchased in these zones [4]. Thus, this research will focus on the implementation of different machine learning models and their respective evaluation with the view to accurately forecasting the Italian Day-Ahead SMP.

2 Scientific Related Work

2.1 Neural Networks

Raquel Gareta, Luis M. Romeo and Antonia Gil demonstrated that the Artificial Neural Network (ANN) approach can be used to forecast short term (for the next day and two or three days after) SMP. Historical SMP prices were solely used as features to train the model. The ANN architecture and design of the model are thoroughly described in their paper. The results were tested with extensive data sets, and good agreement was found between actual data and ANN results [5].

Ioannis P. Panapakidis and Marios N. Moschakis provided a comparative analysis of various machine learning models, such as artificial neural networks and neuro-fuzzy models for the prediction of the Greek SMP. Namely, they studied and compared the performances of a Feed Forward Neural Network (FFNN) and an Adaptive Neuro-Fuzzy Inference System (ANFIS). Historical SMP prices were solely used as features to train the model. Validation was carried out based on the Mean Absolute Range Normalized Error (MARNE) indicator. Their study showed that ANFIS leads to lower errors and therefore, it is more suitable for this specific problem [6].

Nima Amjady and Farshid Keynia propose a combination of a feature selection technique and a cascaded neuro-evolutionary algorithm (CNEA). The feature selection method is an improved version of the mutual information (MI) technique. The CNEA is composed of cascaded forecasters, where each forecaster consists of an ANN and an evolutionary algorithm. An iterative search procedure was also incorporated in their solution to fine-tune the adjustable parameters of both the MI technique and CNEA. The price forecast accuracy of the proposed method was evaluated by means of real data from the Pennsylvania-New Jersey-Maryland and Spanish electricity markets. The candidate inputs used for this work included lagged values of price, load and available generation [7].

J. P. S. Catalão, S. J. P. S. Mariano, V. M. F. Mendes, and L. A. F. M. Ferreira deploy an ANN approach for short-term electricity prices forecasting. A three-layered feedforward ANN was trained by the Levenberg-Marquardt algorithm, and was used for forecasting the next 168-hour power prices. Historical SMP prices were solely used as features to train the model. The ANN architecture and design of the model are thoroughly described in their paper [8].

Pavlos S. Georgilakis in his paper analyses the use of two ANNs: the first to predict the Day-Ahead load and the second to forecast the Day-Ahead market-clearing prices. The methodology is applied on the California power market. After determining the optimal ANN architecture with the minimum mean absolute percentage error (MAPE) on the test set, this architecture was used for price forecasting in periods with price spikes, for weekends, and for week-ahead SMP forecasting during the four seasons of the year. The forecasting accuracy of the ANN model was compared with the accuracy of the persistence method (it is described in the paper) and the results proved the efficiency and practicality of the proposed technique [2].

Adam Marszałek and Tadeusz Burczyński in their paper present a novel approach to forecast hourly Day-Ahead electricity prices. They created a model combining techniques such as, long short-term memory (LSTM) recurrent neural network, attention mechanism, and clustering. The proposed model's main feature is that the attention weights for LSTM hidden states are calculated considering a context vector given for each sample individually as the cluster centre, to which the sample belongs. In training mode, the samples are iteratively clustered based on representation vectors given by the attention mechanism. Historical SMP prices were solely used as features to train the model. In the empirical study, the proposed model was applied and evaluated on the Nord Pool market data. To confirm that the model decreases categorical bias, the obtained results were compared with results of similar LSTM models but without the proposed attention mechanism [9].

Simon Schnürcha and Andreas Wagnera employ machine learning algorithms to forecast German electricity spot market prices. The forecasts utilize in particular bid and ask order book data from EPEX but also fundamental market data like renewable infeed and expected total demand. Using cross validation to optimize hyperparameters, ANNs and random forests were tuned and applied on the data. Their in-sample and out-of-sample performance was compared to statistical reference models. The investigated machine learning models outperformed traditional approaches [10].

S. Anbazhagan and N. Kumarappan in their paper propose a recurrent neural network (RNN) model for Day-Ahead SMP forecasting of the Spanish and the New York electricity markets respectively, that could be realized using the Elman network. The proposed Elman network is a single compact and robust architecture and was compared with many other machine learning methods. The RNN architecture and design of the model are thoroughly described in their paper. Historical SMP prices were solely used as features to train the model [11].

2.2 Autoregression Approaches

Angelica Gianfreda and Luigi Grossi investigate in their paper the Italian Spot Power market with emphasis on price dynamics accounting for technologies, market concentration and congestions. They aimed to understand how technologies, concentration and congestions affect the zonal prices since these ones combine to bring about the single national price (prezzo unico d'acquisto, PUN). They implemented Reg–ARFIMA– GARCH models and assessed the forecasting performance of selected models showing that they perform better when these factors are considered [12].

Zhongfu Tan, Jinliang Zhang, Jianhui Wangb and Jun Xu present a novel price forecasting method based on wavelet transform combined with ARIMA and GARCH models. By wavelet transform, the historical price series were decomposed and reconstructed into one approximation series and some detail series. As a next step, each subseries could be separately predicted by a suitable time series model. The final forecast was obtained by composing the forecasted results of each subseries. This proposed method was examined on Spanish and PJM electricity markets and compared with some other forecasting methods [13].

Cuaresma Jesus Crespo, Hlouskovab Jaroslava, Kossmeierc Stephan and Obersteinerd Michael study the forecasting abilities of a battery of univariate models on hourly electricity spot prices, using data from the Leipzig Power Exchange. The sample period begins at 1:00, June 16, 2000 (the opening of the market) and ends at 24:00, October 15, 2001. The specifications studied include autoregressive models, autoregressive-moving average models and unobserved component models. Historical SMP prices were solely used as features to train the model. The results showed that specifications, where each hour of the day is modelled separately, present uniformly better forecasting properties than specifications for the whole time-series. Furthermore, the inclusion of simple probabilistic processes for the arrival of extreme price events proved to be beneficial for the forecasting abilities of univariate models pertaining to electricity spot prices [14].

Angelica Gianfreda, Francesco Ravazzolo and Luca Rossini compare alternative univariate versus multivariate models and frequentist versus Bayesian autoregressive and vector autoregressive specifications for hourly day-ahead electricity prices, both with and without renewable energy sources. The accuracy of point and density forecasts was inspected in four main European markets (Germany, Denmark, Italy, and Spain) characterized by different levels of renewable energy power generation. The results showed that the Bayesian vector autoregressive specifications with exogenous variables dominate other multivariate and univariate specifications [15].

2.3 Other Approaches

Filipe Azevedo and Zita A. Vale present a use of ANNs in order to find the market price for a given period, with a certain confidence level. Historical information was used to train the ANNs and the number of ANNs used is dependent on the number of clusters found on that data. K-Means clustering method was used to find clusters. The main concern of the method was not to get a single value but a price range forecast with a desired confidence level. A study case with real data from the mainland Spanish market is presented and discussed in detail in their paper [1].

Shu Fan, James R. Liao, Kazuhiro Kaneko, and Luonan Chen proposes a novel model for short-term electricity price forecasting based on an integration of two machine learning technologies: Bayesian Clustering by Dynamics (BCD) and Support Vector Machine (SVM). The proposed forecasting system adopts an integrated architecture. Firstly, a BCD classifier is applied to cluster the input data set into several subsets in an unsupervised manner. Then, groups of 24 SVMs for the next day's electricity price profile are used to fit the training data of each subset in a supervised way. To demonstrate the

effectiveness, the proposed model was trained and tested on the data of the historical energy prices from the New England electricity market [16].

Marco G. Flammini, Giuseppe Prettico, Andrea Mazza and Gianfranco Chicco present in their paper a methodology that allows simulating future electricity wholesale's prices, by considering hourly electricity generation offers datasets. This was accomplished by taking into account new generation units and the dismissing of old (coal-based) ones, according to the demand and generation forecasts in the European Ten-Year Network Development Plan (TYNDP) 2030 scenarios. Machine learning, clustering and distribution sampling techniques were used to ultimately estimate prices distribution in 2030 in the biggest bidding zone of the Italian market. The results suggested that the prices obtained in the different scenarios do converge to those estimated by the TYNDP [17].

2.4 Research Gap

Up to now, a number of studies addressing short term forecasting on power prices for different countries have been carried out. Many of them such as Raquel Gareta et al., Ioannis P. Panapakidis and Marios N. Moschakis, J. P. S. Catalão et al., Adam Marszałek and Tadeusz Burczyński, S. Anbazhagan and N. Kumarappan, Cuaresma Jesus Crespo et al., utilized exclusively historical SMP data and calendar characteristics as inputs, in order to build their models [5] - [6], [8] - [9], [11], [14]. There is another group of studies that have been held regarding again short term forecasting on power prices for different countries, such as Nima Amjady and Farshid Keynia, Pavlos S. Georgilakis, Simon Schnürcha and Andreas Wagnera, Angelica Gianfreda and Luigi Grossi, Zhongfu Tan et al. and Angelica Gianfreda et al. [7], [2], [10], [12] - [13], [15], only that this group utilized among others as inputs fundamental data (e.g., load, renewable energy generation, electricity generation units, fuel prices).

However, none of the above studies have considered the SMP of neighbouring countries in respect to the under-examination country or bidding zone in each occasion. This is a research gap that has to be filled, given the enhanced interrelations among European countries due to the EU Target Model. Furthermore, few studies have been conducted specifically on the forecasting of the Italian SMP. Each country as an energy system exhibits unique characteristics and hence should be examined carefully before proceeding in any price prediction. Italy is a very significant energy market in southern Europe, importing and exporting considerable amounts of electricity on a daily basis.

Thus, the main goal of this research is to select the most relevant and important features for the forecasting of the Italian Day-Ahead SMP and in turn build robust machine learning models which will provide accurate predictions. The most prominent, based on related literature, machine learning algorithms will be applied and evaluated.

3 Main Algorithms

3.1 MLP

The core of neural networks is the node (or neuron). A node takes in one or more inputs, multiplies each input by a weight, sums the weighted input's values along with some bias value, and then feeds the value into an activation function. This output is then sent forward to the other nodes, if any, lying deeper in the network. Feedforward neural networks, also called MLPs, are the simplest ANN used in any real-world setting. Neural networks can be visualized as a series of connected layers that form a network connecting an observation's feature values at one end, and the target value at the other end. The name feedforward comes from the fact that an observation's feature values are fed forward through the network, with each layer successively transforming the feature values with the goal that the output at the end is the same as the target's value. Specifically, feedforward neural networks contain three types of layers of units. At the start of the neural network is an input layer where each unit contains an observation's value for a single feature. At the end of the neural network is the output layer, which transforms the output of the hidden layers into values useful for the task at hand. Between the input and output layers are the hidden layers. These hidden layers successively transform the feature values from the input layer to something that, once processed by the output layer, resembles the target class. Neural

networks with many hidden layers are considered deep networks and their application is called deep learning [18].

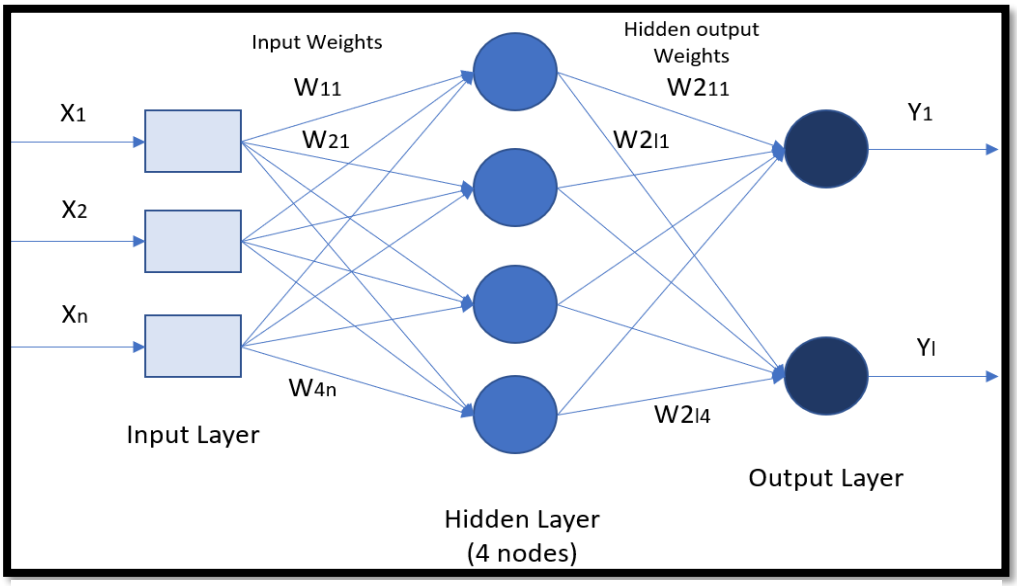

Figure 3-1: Visualization of MLP function.

With regard to neural networks' processing, once an observation is fed through the network, the output is compared with the observation's true value using a loss function. This process is called "forward propagation''. As a next step, an algorithm goes backwards through the network identifying how much each parameter contributed to the error between the predicted and true values, a process called "backpropagation''. At each parameter, the optimization algorithm determines how much each weight should be adjusted to improve the output. MLPs learn by repeating this process of forward propagation and backpropagation for every observation in the training data multiple times, iteratively updating the values of the parameters. In most cases features have to be normalized to the same scale, so that our neural network does not underperform.

To construct and train an MPL, there is a number of choices to be made about both the network architecture and training process [18]:

1. Enable sequences of layers over time.
2. For each layer in the hidden and output layers we must define the number of units to include in the respective layers and the activation functions.
3. Define the number of hidden layers to use in the network. More layers allow the network to learn more complex relationships, but with a computational cost.
4. Define the structure of the activation function (if any) of the output layer.
5. Define a loss function, that is the function that measures how well a predicted value matches the true value.
6. Define an optimizer, which intuitively can be thought of as our strategy "walking around" the loss function to find the parameter values that produce the lowest error (e.g., stochastic gradient descent, stochastic gradient descent with momentum, root mean square propagation and adaptive moment estimation).
7. Select one or more metrics to evaluate model's performance, such as accuracy.
8. Separate training and test data properly.
9. Provide number of epochs, each time all observations have been sent through the network is called an epoch.
10. Provide batch size, the number of observations utilized in one iteration.

All of the above steps are used to initiate (1), construct (2,3,4), compile (5,6,7) and fit (8,9,10) the MLP model.

3.2 RNN

The central problem in machine learning and deep learning is to meaningfully transform data, in order to learn useful representations of the original input data, that is representations that get us closer to the expected output. Deep learning is a subfield of machine learning, giving an emphasis on learning successive layers of increasingly meaningful representations. The term deep in deep learning stands for the concept of successive layers of representations and the number of layers contributing to a model of the data is called the depth of the model [19].

RNNs are a class of ANNs that can process a sequence of inputs in deep learning and retain its state, while processing the next sequence of inputs. Traditional neural networks will process an input and move onto the next one disregarding its sequence. Namely, a RNN is different from a traditional neural network because it introduces a transition weight to transfer information between time. The introduction of the transition weight means that the next state is now dependent on the previous model, as well as the previous state [20].

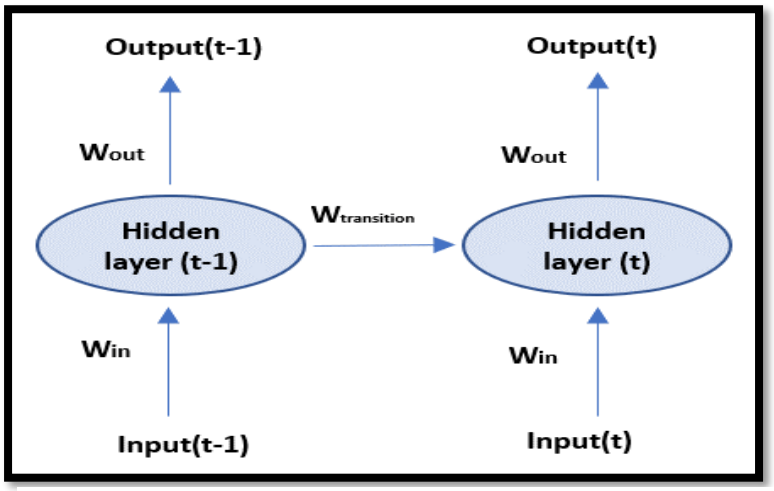

Figure 3-2: Visualization of RNN function.

There is a number of steps/processes that have to be followed in order to create and fit a RNN model to a time series:

1. Initialize RNN – Enable sequences of layers over time
2. Determine shape of input data.
    a. Samples. One sequence is one sample. A batch is comprised of one or more samples.
    b. Time Steps. One time step is one point of observation in the sample. One sample is comprised of multiple time steps.
    c. Features. One feature is one observation at a time step. One time step is comprised of one or more features.
3. Tune model hyperparameters.
    a. Number of nodes and hidden layers
    b. Learning rate
    c. Decay Rate
    d. Weight initialization
    e. Activation function
4. Create output layer
    a. Number of units in a dense layer
5. Compile model
    a. Optimizer
    b. Loss function
6. Fit model
    a. Training data (independent and dependant)

b. Epochs
c. Batch size

A popular type of RNN is the LSTM network. Experiments show how difficult is to effectively train a RNN, due to the weight update process leading to weight changes that quickly become too small and hence have no effect, the so called ‘‘vanishing gradient problem’’, or so large as to result in sharp changes experiencing exploding gradients. LSTM is an RNN architecture specifically designed to address the vanishing gradient problem. An LSTM layer consists of a set of recurrently connected blocks, known as memory blocks. These blocks can be thought of as a differentiable version of the memory chips in a digital computer. Each one contains one or more recurrently connected memory cells and three multiplicative units - the input, output and forget gates, that provide continuous analogues of write, read and reset operations for the cells.

A memory cell has weight parameters for the input, output, as well as an internal state that is built up through exposure to input time steps:

- Input Weights. Weight input for the current time step.
- Output Weights. Weight the output from the last time step.
- Internal State. Applied in the calculation of the output for the current time step.

The key to the memory cell is the gates. These are also weighted functions that further govern the information flow in the cell:

- Forget Gate: Certain information is discarded from the cell.
- Input Gate: Certain values from the input are used to update the memory state.
- Output Gate: The output calculated based on input and the memory of the cell.

The forget gate and input gate are used in the updating of the internal state. The output gate determines what the cell actually produces. It is these gates and the consistent data flow called the constant error carrousel that keep each cell from neither exploding or vanishing [21].

3.3 Autoregressive Algorithms

Autoregression is a time series method that uses observations from the past as input to a regression equation to predict a value in the future. It is a simple idea that can produce accurate forecasts on a range of time series problems. A regression model, such as linear regression, models an output value based on a linear combination of independent input variables:

$$Y = C0 + (C1 * X1) + \ldots + (\mathrm{Cn} * \mathrm{Xn}) \quad (1)$$

Where, Y is the target variable, C0 and Cn are coefficients found by fitting the model on training data, and X1,..., Xn are input variables. The same idea can be applied on time series, where observations of the target variable at previous time steps are used as input variables, called lag variables:

$$Y(t,p) = e + bo + b1 * X(t-1) + \cdots + bp * X(t-p) \quad (2)$$

Where, Y(t,p) is the target variable for time t, e is white noise, b0, b1 and bp are coefficients found by fitting the model on training data and X(t-1) and X(t-p) are the lag variables. Due to the fact that the regression model uses data from the same input variable at previous time steps, it is referred to as an autoregression.

The deviation of the forecast from the actual/expected values (residual errors) from on a time series data set constitutes another source of information, which can be modelled. Residual errors themselves form a time series that can have temporal structure. A simple autoregression model of this structure can be used to predict the forecast error, which in turn can be used to correct forecasts. An effective modelling, of residual errors, of this type is considered an autoregression as well:

$$Y'(t,q) = e1 + z0 + z1 * e(t-1) + \cdots + zq * e(t-q) \quad (3)$$

Where, Y’(t,q) is the target variable for time t, e1 is white noise, z0, z1 and zq are coefficients found by fitting the model on training data and e(t-1) and e(t-q) are the lag values of the residual errors [22].

Thus, combining all of the above we end up to AutoRegressive Moving Average (ARMA) which is a method of statistical models for analysing and forecasting time series and can be expressed as:

$$ARMA(p,q) = Y(t,p) + Y'(t,q) \ \ from\ (2)\ and\ (3)$$

Where, p and q is the number of lags for Y(t,p) and Y'(t,q) respectively. ARMA model can be applied only on stationary time series. If the time series are not stationary AutoRegressive Integrated Moving Average (ARIMA) has to be applied. ARIMA is a generalization of ARMA and adds the notion of integration in order to make time series stationary.

- AR: A model that uses the dependent relationship between an observation and some number of lagged observations.
- I: The use of differencing of raw observations in time, subtracting an observation from an observation at the previous time step, in order to make the time series stationary.
- MA: A model that uses the dependency between an observation and residual errors from a moving average model applied to lagged observations.

ARIMA(p,d,q) receives the same parameters as inputs with the ARMA model plus d, which is the number of differencing between lag observations. However, ARIMA is still not able to identify seasonality in the data. Thus, in case there is seasonality in the data the use of Seasonal AutoRegressive Moving Average (SARIMA) is necessary. SARIMA is a generalization of ARIMA and has a seasonal component through which seasonality can be offset. SARIMA requires selecting hyperparameters for both the trend and seasonal elements of the series, SARIMA(p,d,q)(P,D,Q)S [22].

Trend elements:

- p: Trend AR order.
- d: Trend difference order.
- q: Trend MA order.

Seasonal elements:

- P: Seasonal AR order.
- D: Seasonal difference order.
- Q: Seasonal MA order.
- S: The number of time steps for a single seasonal period.

4 Problem Description

Italy presents particular interest as an energy market due to its unique characteristics and the significant amounts of electricity exchanged through cross border trading, while also being one of the biggest electricity consumers in Europe [23]. That is, Italy is separated into a number of zones of which North, Central North, Central South, Sardinia, South and Sicily are the most important. The SMP or PUN of Italy is formed by the weighted average of zonal prices, whereby when a congestion event occurs between zones has a direct effect on the country's overall SMP. Italy is also bordering with a number of countries such as France, Switzerland, Austria, Slovenia, Malta, Montenegro and Greece, exchanging considerable amounts of electricity through the respective interconnections. As a consequence, the outlook of the neighbouring countries' SMP has a direct impact on the Italian SMP and vice versa. Next, Italy is diverse in weather conditions across the country leading to different load profiles for each area with the Northern zone dominating overall. Finally in the electricity mix, thermal energy (predominantly generated from gas) lies in the first place with hydroelectric power, wind power, solar power and geothermal power combined in the second place [24].

Thus, the challenge of this thesis is to properly analyse the Italian energy system in relation to wholesale electricity prices and then proceed with the construction and evaluation of three robust and consistent machine learning models with a view to forecasting as much accurate as possible the Italian Day-Ahead SMP. Namely, the challenges to overcome are,

- Initial attributes selection.
- Find reliable sources for data collection.

- Data pre-processing if required.
- Attributes determination – filter out redundant attributes if any.
- Determine optimal time periods for both training and test data sets.
- Construct one MLP, one LSTM and one Autoregressive model in a consistent manner.
- Tune the respective hyperparameters and parameters of each model.
- Perform SMP Day-Ahead predictions and evaluate the results of each machine learning model accordingly.

5 Methodology

5.1 Initial Attributes Selection

Attribute selection will be based on the degree of relevance pertaining to SMP and based on the degree to which attributes can be utilized for future forecasting purposes.

As has already been discussed, SMP of neighbouring countries are impacting the Italian wholesale energy market and have to be considered as attributes, which is also the main research gap that has to be filled. However, not all of the neighbouring countries' SMP will be taken into account since not all of them are as much impactful. In fact, only French wholesale prices will be considered, the German (GER) SMP and the Hungarian (HU) SMP. That is, France and Switzerland as depicted in the below graph exchange by far the largest amounts of electricity with Italy, something that is also supported by their corresponding available transmission capacities [4]. However, Switzerland's SMP is highly influenced by the German, Hungarian and French (FR) SMP respectively. In addition, there is not much crucial information readily available online for Switzerland's electricity market, in order for its SMP to be reliably forecasted for future use as an attribute and hence is discarded from any further investigation. Germany is the central energy hub of Europe and Hungary is the major energy market of the eastern

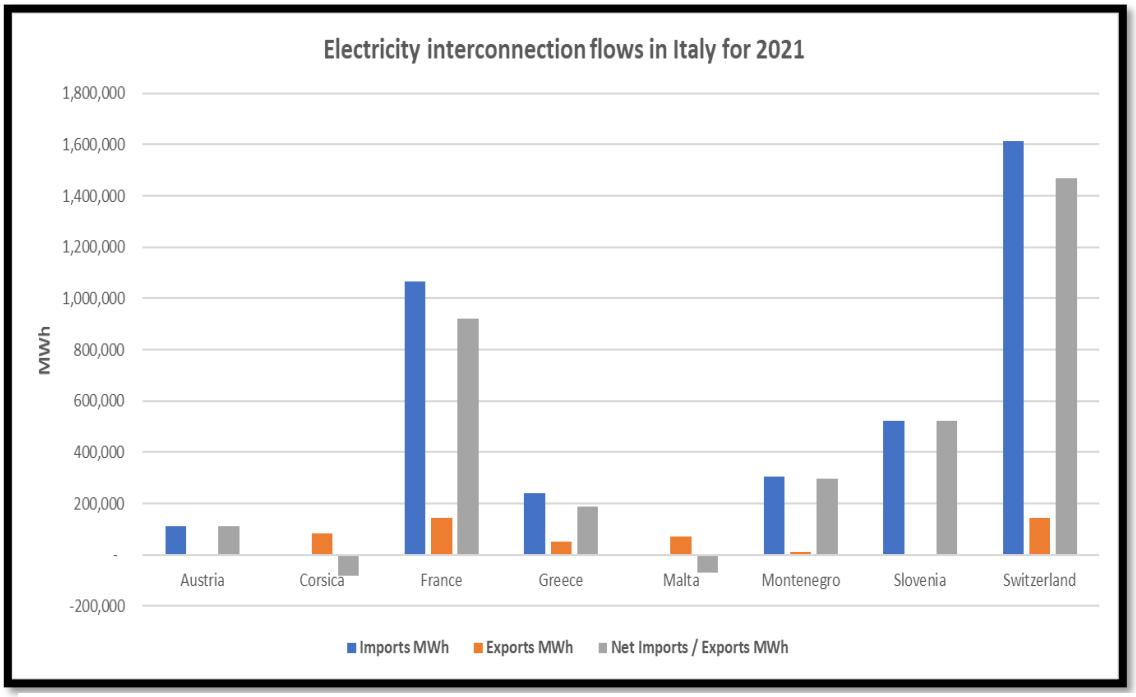

Figure 5-1: Electricity exchange in the Italian border for 2021 as provided by Terna [24].

European area affecting many countries adjacent to Italy such as Slovenia, Austria and Switzerland. Thus, French, German and Hungarian SMPs are selected as attributed for further investigation.

PUN is determined based on the economic merit-order criterion, while taking into account transmission capacity limits between zones. The general notion of the merit-order criterion is that cheap

electricity production technologies are the first to be brought online in order to meet demand for load, as opposed to expensive technologies which are the last to be brought online. As a consequence, load and power generation per technology are essential drivers of wholesale electricity prices and should be used as attributes. Instead of using separately load and power generation per technology as attributes, an expression of them is used.

$$Residual = Load - (Hydro + Wind + Solar) \quad (4)$$

Electricity produced by hydro power plants (run-of-river), wind turbines and solar panels is cheap and usually these power plants offer their energy at minimum prices due to their participation in ‘‘green’’ supporting mechanisms. There are also other renewable energy technologies producing electricity in Italy like geothermal and biomass, nevertheless no respective electricity generation forecasts could be found for future use and hence only Hydro, Wind and Solar power data are considered. Thus, residual is the remainder amount of load demand that has to be matched by more expensive technologies and is selected as an attribute.

As has already been mentioned PUN is formed by the weighted average of zonal prices and subsequently any congestion between zones has a direct effect on the country’s overall SMP. Consequently, congestion is selected as one more attribute and for this, historical zonal data from the most important zones of Italy will be collected.

Finally, many of the aforementioned studies in previous sections utilize lag values of the SMP itself (target variable) and calendar characteristics as attributes showing promising results. Thus, the most relevant lag values of the SMP will be determined and in turn will be selected as attributes, together with the calendar characteristics of the target variable. As a result, the initial selection of the attributes summarizes in,

- FR SMP.
- GER SMP.
- HU SMP.
- Residual.
- Congestion.
- SMP lag values.
- Calendar Characteristics.

5.2 Data Preparation

According to the initial attribute selection as described in the previous section, a number of data were collected from multiple sources, and all of them spanning from 01/01/2017 to 22/10/2021. Namely, historical hourly SMP data regarding Germany and Hungary were collected from the official site of Entso-e [23]. Continuing, historical hourly SMP data for France were obtained as well from Nord Pool [25]. GME was another source, where actual data on zonal (NORD, CNOR, CSUD, SARD, SUD and SICI) hourly SMPs were acquired [4]. Finally, historical data of the Italian electricity market on actual load and actual electricity generation per technology were collected from Entso-e and Terna [23], [24]. In total 10 csv and 2 xlsx files, not exceeding 1.7 MB each, of data were used and processed to create one Master csv file, so that all useful input data are stored properly in order and in one place ready for use. All data pre-processing was carried out in the programming language python.

At first, 5 xlsx files containing all zonal SMP prices were merged into one dataframe. Redundant columns were dropped, while date data were processed and converted from string format to datetime format and subsequently set as index. All of the remaining data were converted from string format to numeric. Pandas library, os and datetime modules were used accordingly. A final file was created containing datetimes from 01/01/2017 to 22/10/2021 and the respective PUN and zonal SMP prices. The same process was followed for data pertaining to respective amounts of electricity purchased and sold.

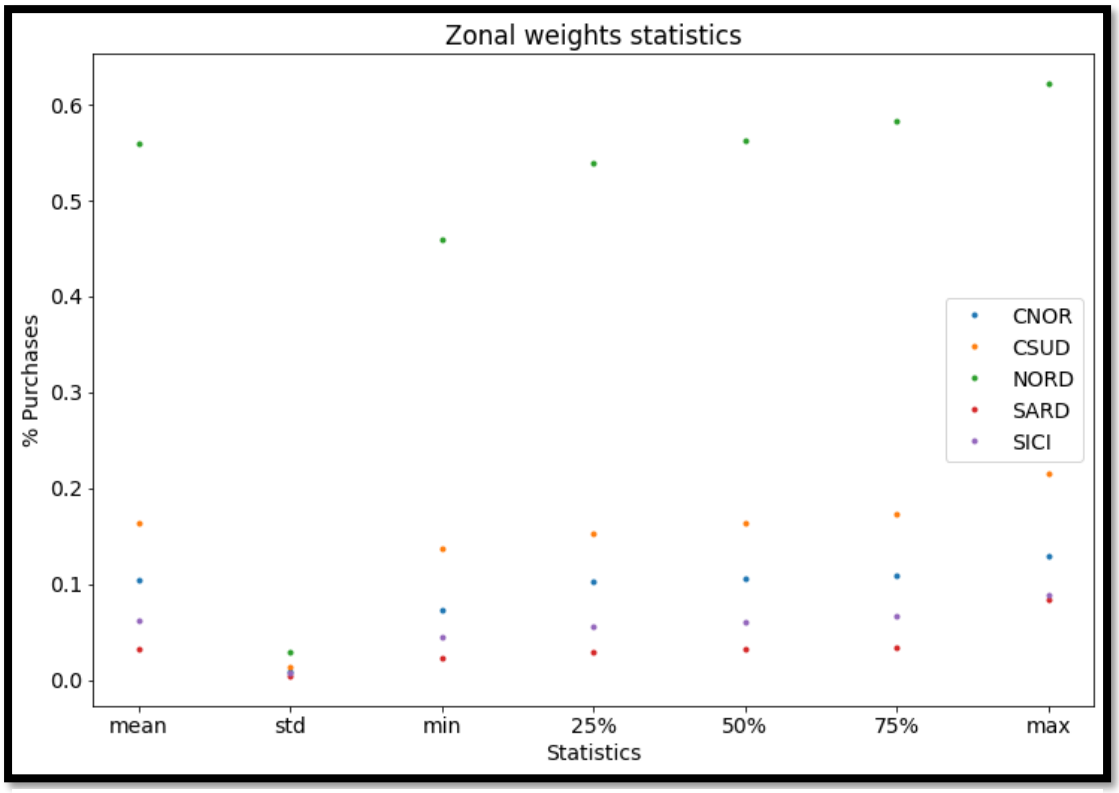

Figure 5-2: Zonal contribution to PUN formation.

In the above graph, the contribution of each zone to the formation of PUN price separately is depicted. Due to the fact that standard deviation is very low, it can be safely deduced that NORD is the most influential zone of Italy, which together with CSUD account for more than 70 per cent of PUN

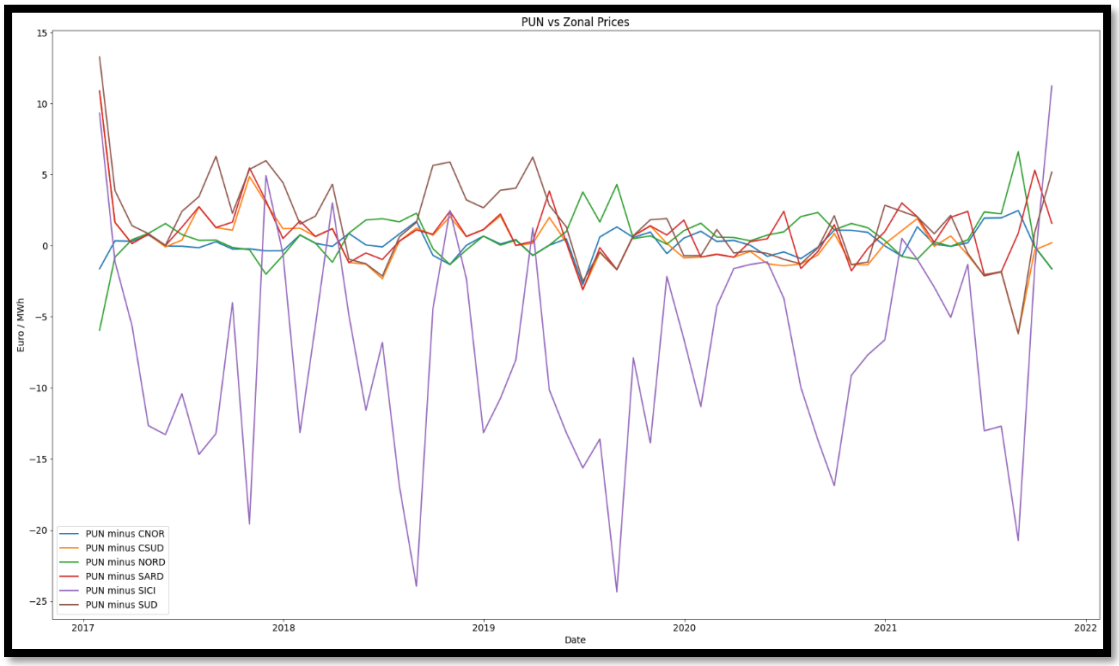

Figure 5-3: Monthly PUN and Zonal prices comparison.

price. The above graph shows the relationship between PUN and the different zonal prices.
It can be clearly seen that NORD price is not deviating significantly from PUN. Furthermore, there is a looming reverse relationship between NORD and CSUD, meaning that when congestion events occur

those two zonal prices are opposed with CSUD's price being higher than NORD's price in most cases. Data were resampled from hourly to monthly owe to clarity reasons. As a next step, a different code was developed through which load, renewable energy production (RES) per technology, HU SMP, GER SMP and FR SMP data are loaded together with the data from the already created file for PUN, and all of them are stored separately in different dataframes. Again, Pandas library, os and datetime modules were used in order to concatenate the different files and change the date string format to datetime format. The initial date format and file layout was not the same in all of the files, as a result list(), map(), lambda() strptime() and read_csv() functions are applied combined with new created functions to handle properly each occasion. With regard to the Italian zonal prices a new feature is created representing congestion. That is, a new column is inserted named ''Congestion'', whose prices are 0, if there is no congestion, -1 if NORD's price is higher than PUN and 1 if PUN is higher than NORD's price. Thus, six different dataframes for load, RES, HU, GER, FR and Italian data are used, where a date filter is applied in order to make data consistent. Data from 2019-07-04 and backwards are filtered out due to the fact that there were no data available for FR during this time period. Missing values are present in rows where time changes to 1 hour less in each March of every year, and hence those rows are removed. Finally, the six dataframes are concatenated into one dataframe ready for further analysis. The correlation between lag

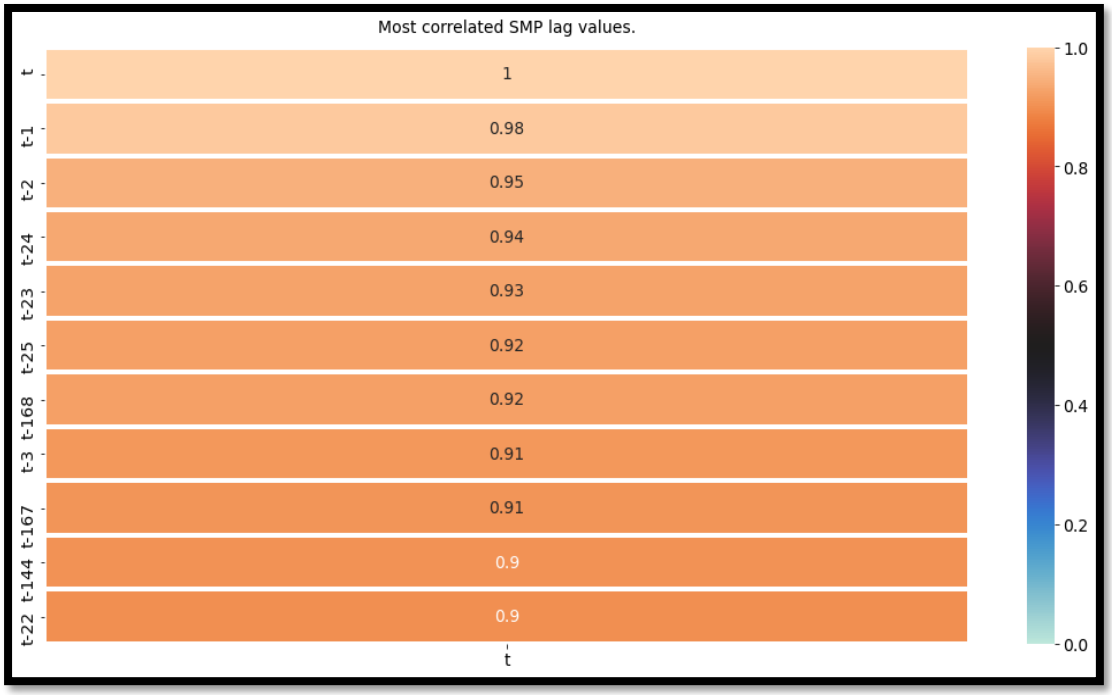

Figure 5-4: PUN lag values correlation.

values of PUN up to 168 hours (one week) in the past is investigated. For this, seaborn module corr() and heatmap() functions are used. From the above picture can be observed that t-24 lag is the most correlated one which could be used as a feature (t-1 and t-2 won't be known when forecasting). Thus, a new column named ''SMP-24'', with shifted 24 hours ahead PUN data is inserted as a new feature. The resulting dataframe is consisted of 20208 rows and 7 columns (Congestion, HU, FR, GER, Residual, SMP-24 and SMP), where the Residual column is derived from the difference between the load and the aggregated RES production and SMP is PUN price. The first 6 columns represent the features and the last one, SMP, the target variable. All features are then normalized by applying the MinMaxScaler function and using a (0,1) feature range. This dataframe is used thereinafter for filtering out any redundant attributes / features.

5.3 Attributes Determination

In this section, the methods applied to determine the most important attributes among Congestion, HU, FR, GER, Residual, SMP-24 and their respective results are discussed. These methods are the Sequential Forward Selection, the Random Forest Regressor and the f_regression.

5.3.1 Sequential Forward Selection (SFS)

SFS is a sequential feature selection algorithm that adds features from the dataset sequentially until the addition of extra features does not improve the performance of the model. Occasionally they evaluate each feature separately and select K features from L features, K < L, on the basis of individual scores. That is, the aim is to improve the computational efficiency and reduce the generalization error of the model by removing irrelevant features or noise. SFS has two main components:

1. An objective function: It is the criterion, mean squared error for regression models, based on which the optimal subset of features can be selected.
2. A sequential search algorithm: This searching algorithm adds sequentially variant features to an empty set of features, while evaluating the objective function.

SFS has been executed in python, where mlxtend.feature_selection, sklearn.linear_model libraries and SequentialFeatureSelector and Lasso modules are used. Specifically,

$$SequentialFeatureSelector(Lasso(), k\_features = 6, forward = True, floating = False, scoring = 'neg\_mean\_squared\_error', cv = 10, n\_jobs = -1)$$

Where, Lasso() is the estimator or the sequential searching algorithm, k_features is the maximum number of features that could be selected, neg_mean_squared_error is the criterion, forward = True indicating a forward selection method and cv=10 which is a 10 kfold cross validation.

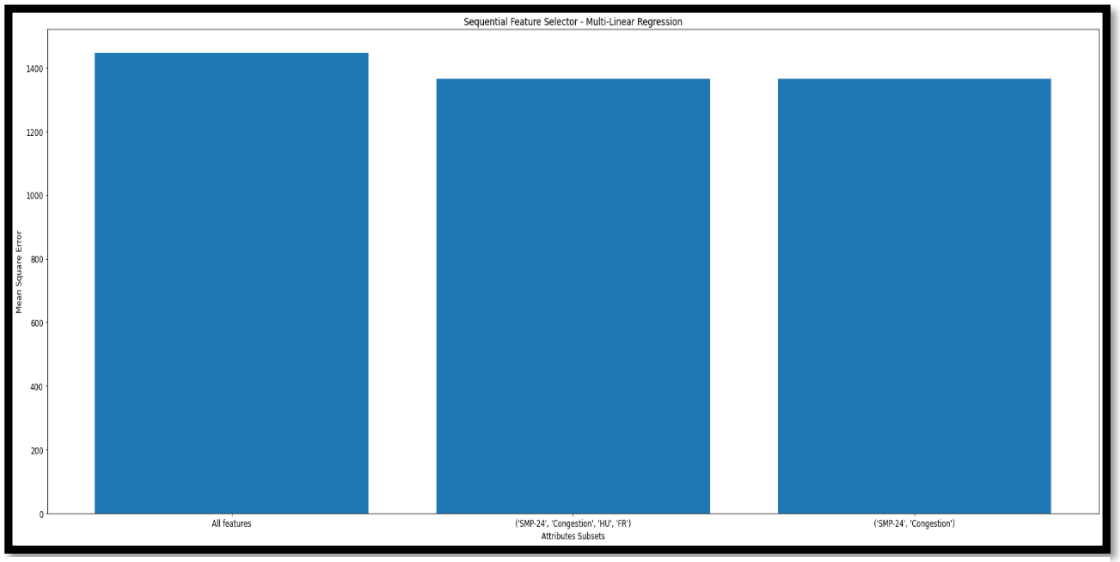

Figure 5-5: SFS feature subset indicative results.

In the above graph indicative results from the SFS algorithm are shown. There is an increase in the mean square error of the model when Residual is added to the subset of features. Model's performance remains unaffected for the remaining subsets.

5.3.2 Random Forest Regressor

Every decision tree has high variance, but in a random forest many trees are trained in parallel resulting in a lower variance, as each decision tree gets perfectly trained on that particular sample of data and consequently the output depends on multiple decision trees and not only one. A Random Forest has each tree only receiving a random sample of observations with replacement that matches the original number of observations and each node only considers a subset of features when determining the best split. In the case of a regression problem, the final output is the mean of all the outputs.

Feature selection through Random Forest Regressor method is based on importance measures, which for classification tasks can be gini impurity or infomation gain, and for regression is variance reduction. Random Forest Regressor's structure is a set of decision trees, where each decision tree is a set of internal nodes and leaves. Measuring how each feature decreases the variance of the split, the feature with highest decrease is selected for internal node. The average of variance reduction over all trees in the forest is the feature's importance [18]. For this method sklearn.ensemble library and RandomForestRegressor module are used. Specifically,

$$RandomForestRegressor(n\_estimators = 500, random\_state = 1, n\_jobs = -1)$$

Where, n_estimators is the number of trees in the forest. In the below figure the importance scores of each feature are depicted.

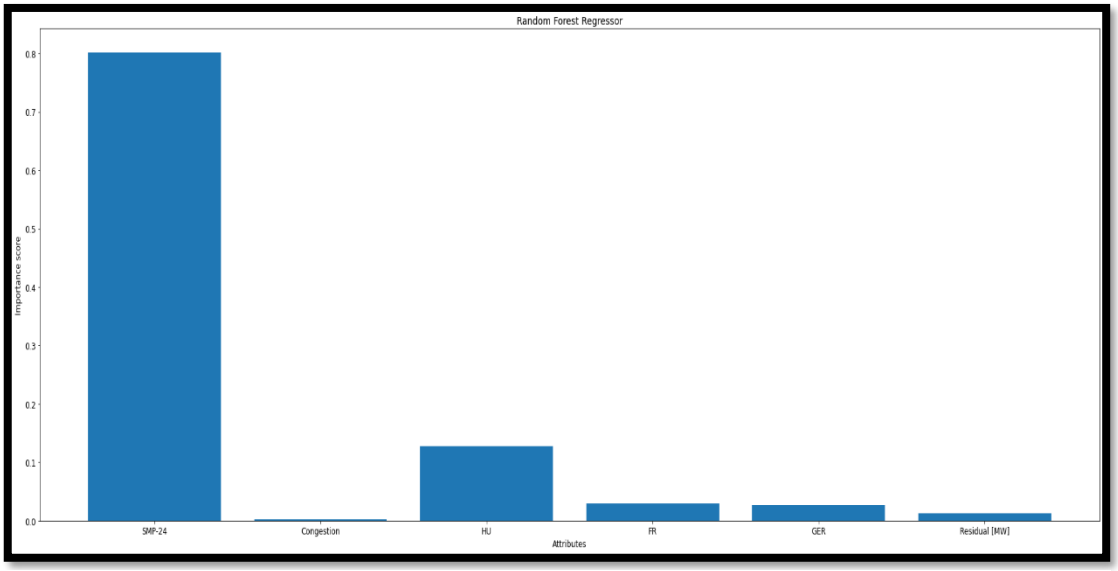

Figure 5-6: Individual feature importance scores.

As can be seen from the figure, SMP-24 has by distance the highest importance score and Congestion the lowest. HU comes second with FR, GER and Residual following in that order.

5.3.3 F_regression

F_regression is a method for univariate linear regression tests returning F-statistic and p-values. What is more, it is a linear model for testing the effect of a single predictor, sequentially for as many different predictors. F_regression is recommended as a feature selection criterion for a downstream classifier, irrespective of the sign of the association with the dependent variable. The f-test in regression compares your model with zero predictor variables, and determines if model's coefficients improved performance. If you get a significant result, then the coefficients included in the model resulted in a better fit. In general, the higher the F-value of a feature together with a small p-value (<0.05) the more significant the feature is. For this method, sklearn.feature_selection library and f_regression module are used.

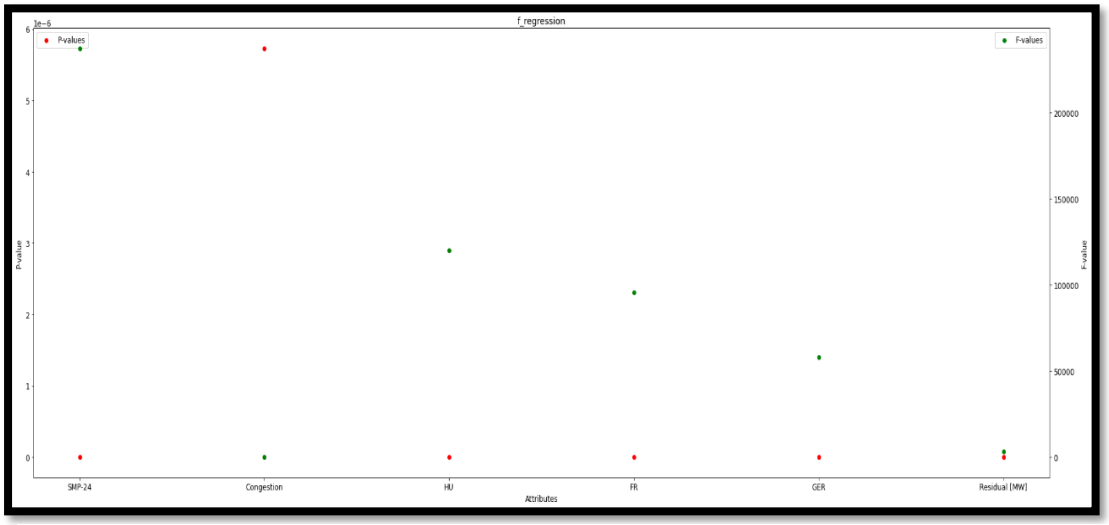

Figure 5-7: Features F-values and p-values.

All features have small p-values much lower than the typical level of 0.05. SMP-24 feature has the highest F-value and Congestion feature the lowest. HU has the second highest with FR, GER and Residual following in that order. Red dots represent p-values and green dots F-values.

5.3.4 Feature Selection Discussion

In this sub section the different methods used for the determination of the most important features were presented and their respective results. Through the first method, SFS, is implied that regression would work better with less than the full set of the features under investigation. Namely, it is indicated that the addition of the Residual would worsen the model's performance. In the second method, Random Forest Regression, SMP-24 is found to be the most important feature with Congestion being least important. Finally, via F_regression method SMP-24 is found to be again the most significant feature with Congestion being the least significant, nevertheless not insignificant.

From all of the examined features only SMP-24 is historical, the remaining features need to be forecasted as much accurately as possible in order to contribute to a reliable forecast. Thus, it is of great importance to reduce this uncertainty and to do so some of HU, FR, GER, Residual and Congestion will be filtered out. Below a heatmap with the respective correlations is presented providing further insights.

Figure 5-8: Pearson correlation scores between features and target.

Figure 5-8 indicates a strong correlation between SMP-24 and target variable (SMP) of 0.96. In addition, HU, FR and GER are significantly correlated to SMP with 0.93, 0.91 and 0.89 respectively. Residual shows a small correlation to the target variable and Congestion practically no correlation at all. It has to be noted that HU, FR and GER are strongly intercorrelated implying redundancy. Taking into account all of the above, SMP-24, HU and Residual are selected as the most prominent features and will be used at a later stage to train certain machine learning algorithms. Despite the fact that Residual does not appear to be as much promising as SMP-24 and HU, is selected due to accurate available forecasts online and due to its by definition fundamentals' nature, that can combine properly with the other 2 features. Following the determination of the most prominent features, a dataframe with 4 columns (SMP-24, HU, Residual and SMP) is finalized and subsequently saved in a csv format, in order to be used later as input to the respective machine learning algorithms. Below the function created in python for feature selection is provided.

```
def attribute_s(features,target,mode):
    from matplotlib import pyplot as plt
    import pandas as pd
    if mode == 'LR':
        from mlxtend.feature_selection import SequentialFeatureSelector as SFS
        from sklearn.linear_model import Lasso
        sfs=
SFS(Lasso(),k_features=6,forward=True,floating=False,scoring='neg_mean_squared_error',cv=10,n_jobs=-1)
        sfs=sfs.fit(features,target)
        results=pd.DataFrame(sfs.subsets_).transpose()
        groups=[i for i in results.feature_names]
        groups[-1]='All features'
        values=[]
        [values.append(-i) for i in results['avg_score']]
        ticks=[i for i in range(1,len(results)+1)]
        plt.figure(figsize=(33,12))
        plt.bar(ticks[0:3],values[-1:0:-2])
        plt.xticks(ticks[0:3],groups[-1:0:-2])
        plt.xlabel('Attributes Subsets')
        plt.ylabel('Mean Square Error')
        plt.title('Sequential Feature Selector - Multi-Linear Regression')
        plt.show()
        return print(results[results.avg_score==results.avg_score.max()][['avg_score','feature_names']])
    elif mode == 'RF':
        from sklearn.ensemble import RandomForestRegressor
        model = RandomForestRegressor(n_estimators=500, random_state=1,n_jobs=-1)
```

```
        model.fit(features, target)
        # plot importance scores
        names = features.columns
        ticks = [i for i in range(len(names))]
        plt.figure(figsize=(33,12))
        plt.bar(ticks, model.feature_importances_)
        plt.xticks(ticks, names)
        plt.xlabel('Attributes')
        plt.ylabel('Importance score')
        plt.title('Random Forest Regressor')
        plt.show()
        return print(model.feature_importances_)
    elif mode == 'FR' :
        from sklearn.feature_selection import f_regression
        # Select features with highest F-values
        fval,pval = f_regression(features, target)
        results=pd.DataFrame(data=pval,columns=['P-value'],index=features.columns)
        results['F-value']=fval
        fig,ax=plt.subplots(figsize=(33,12))
        ax1=ax.twinx()
        ax.scatter(results.index,results['P-value'],color='r',label='P-values')
        ax1.scatter(results.index,results['F-value'],color='g',label='F-values')
        ax.legend(loc='upper left')
        ax1.legend()
        ax.set_xlabel('Attributes')
        ax.set_ylabel('P-value')
        ax1.set_ylabel('F-value')
        ax.set_title('f_regression')
        plt.show()
        return print(results.sort_values(by='P-value'))
```

The function accepts as inputs the features to be investigated (should be 6 in total), the target variable and the feature selection method (LR, RF, FR).

5.4 Development of Machine Learning Algorithms

With regard to electricity spot price forecasting, most of the studies use a training sample of approximately 2 months [2], [7]. What is more, the electricity market in Europe is facing an unprecedented sharp increase in wholesale power prices since June 2021. As a result, the file created as discussed in the previous section is filtered and split into two samples, one for a period extending from 2021/06/01 to 2021/10/08 (training sample) and another one for a period extending from 2021/10/09 to 2021/10/22 (test sample). Namely, the training sample and the test sample consist of 3120 and 336 records respectively and are used to build and evaluate the models.

Datetime Index	SMP-24	HU	Residual [MW]	SMP
01/06/2021 00:00	59.60	66.37	19578	73.72
01/06/2021 01:00	57.30	62.00	18025	71.21
01/06/2021 02:00	56.22	59.82	17477	66.92
01/06/2021 03:00	55.02	58.13	17199	64.76
01/06/2021 04:00	55.90	59.73	16996	65.31
…	…	…	…	…
22/10/2021 19:00	271.16	310.01	34444	312.66
22/10/2021 20:00	250.00	251.84	32708	265.00
22/10/2021 21:00	218.00	202.43	30895	246.43
22/10/2021 22:00	213.05	234.74	27995	218.11
22/10/2021 23:00	198.16	183.18	25546	210.45

Table 1: Dataset including features and target variable.

Input data are hourly and the target variable has inherent multiple seasonalities which have to be handled. Thus, a function called fourier_con() is created through which Fourier cyclical series for week, weekday and hour are added as extra features in the dataset. The function and the corresponding calculations are provided below.

```
def fourier_con(dataframe, wk, wdk, hk):

    for k in range(1, wk+1):

        # week has a period of 53
        dataframe['week_sin'+str(k)] = np.sin(2 *k* np.pi * dataframe.index.week/53)
        dataframe['week_cos'+str(k)] = np.cos(2 *k* np.pi * dataframe.index.week/53)

    for k in range(1, wdk+1):

        # weekday has a period of 7
        dataframe['weekday_sin'+str(k)] = np.sin(2 *k* np.pi * dataframe.index.dayofweek/7)
        dataframe['weekday_cos'+str(k)] = np.cos(2 *k* np.pi * dataframe.index.dayofweek/7)


    for k in range(1, hk+1):

        # hour has period of 24
        dataframe['hour_sin'+str(k)] = np.sin(2 *k* np.pi * dataframe.index.hour/24)
        dataframe['hour_cos'+str(k)] = np.cos(2 *k* np.pi * dataframe.index.hour/24)
```

For the realization of all of the above pandas and numpy libraries are used. The fourier_con() function accepts as inputs a dataframe and the number of terms to be created for week, weekday and hour

respectively, which will be used in turn as features to handle seasonality. Finally, all features are normalized on the training sample, using MinMaxscaler at a (0,1) feature range.

In order to assess the prediction capacity of the models, 2 evaluation metrices are applied, the Mean Absolute Percentage Error (MAPE) and the Root Mean Square Error (RMSE). MAPE is an evaluation metric used to assess the performance of a model. It is unitless and expresses errors as a percentage of the actual data. It is calculated as per the below equation,

$$MAPE = \frac{1}{N} * \sum_{i=1}^{N} \left| \frac{Actual(i) - Forecast(i)}{Actual(i)} \right| * 100\% \quad (5)$$

Where, N is the total number of observations in the test sample. There is a drawback in the usage of MAPE as defined above and this is when actual values are zero. These cases are rare in the original data and when an actual price is zero, is subsequently replaced by the average of the respective 24-hourly values. The mean squared error is calculated as the average of the squared forecast error values, where forecast error is the difference between the predicted values and the actual values. Very large or outlier forecast errors are squared, which in turn leads to dragging the mean of the squared forecast errors out having the effect of a larger mean squared error score. In effect, the score gives worse performance to those models that make large wrong forecasts. The mean squared error metric as described above, has the squared units of the predictions. It can be transformed back into the original units of the predictions by taking the square root of the mean squared error score [22]. In short, RMSE is the square root of the mean squared error and is defined as follows,

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Forecast(i) - Actual(i))^2}{N}} \quad (6)$$

Where, N is the total number of observations in the test sample. For the calculation of MAPE and RMSE pandas and numpy libraries, sklearn.metrics and mean_squared function are used.

5.4.1 Autoregression - SARIMAX

A time series is a sequence where a metric is recorded over regular time intervals. In this research the target variable is a time series of hourly frequency. In case of utilizing predictors or alternatively exogenous variables other than the series itself to make a forecast, the method is called multi variate time series forecasting. As has already been discussed, ARIMA is a forecasting algorithm based on the idea that information in lag values of the time series can be solely used to make predictions. SARIMAX is a general extension of ARIMA, which can handle seasonality and supports the use of exogenous variables. Thus, is selected to perform autoregression and ultimately make hourly predictions from one Day-Ahead up to two weeks ahead.

Time series differs from more regression predictive modelling problems. The temporal structure adds an order to data. This order implies that important assumptions taken concerning the consistency of these data needs to be handled. For instance, when modelling time series, the summary statistics of observations is assumed to be consistent. This expectation is referred as the time series being stationary. Stationarity is a statistical property in which time series show constant mean and variance over time. Stationarity could be easily violated through the addition of a trend, seasonality, and other time-dependent structures. One of the common methods to perform a stationarity check on a time series data is the Augmented Dickey-Fuller (ADF) test. Namely, there is a unit root test and a null hypothesis that

the time series is considered non-stationary and whether the p-value of the test is less than a significance level (e.g. 0.05), then it rejects the null hypothesis and considers the time series to be stationary [22].
In the below figure the seasonal decomposition of the dependent variable is depicted. In the first graph the original data of the SMP are shown, in the second graph the trend component is illustrated, the

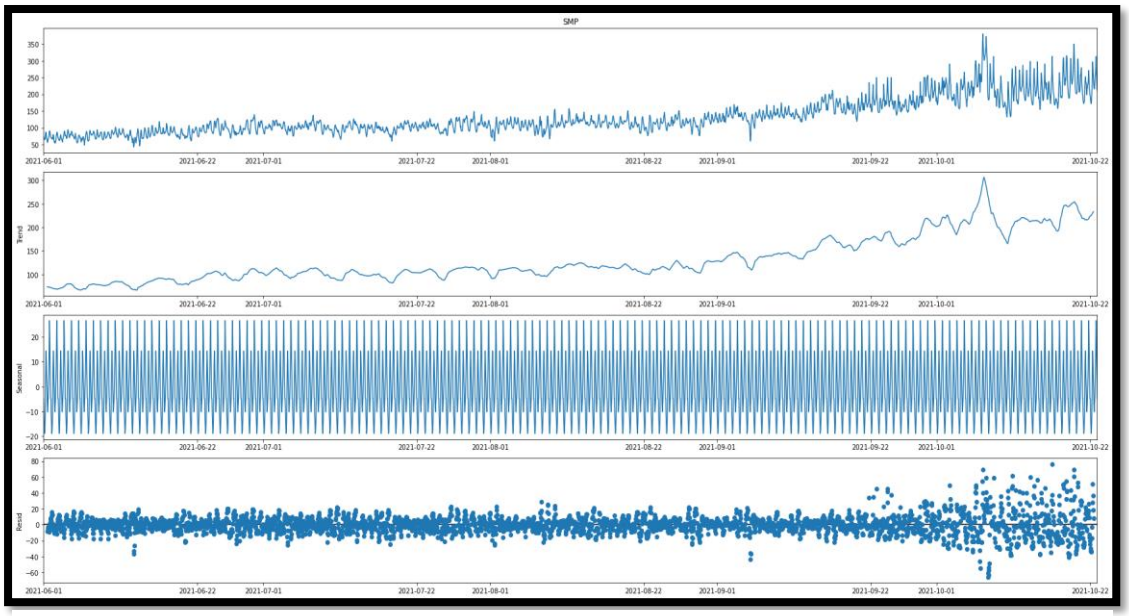

Figure 5-9: Seasonal decomposition of the time series.

seasonal component is shown in the third graph and in the last graph is white noise. It can be seen that there is an increasing trend in the data and that there is multiple seasonality.

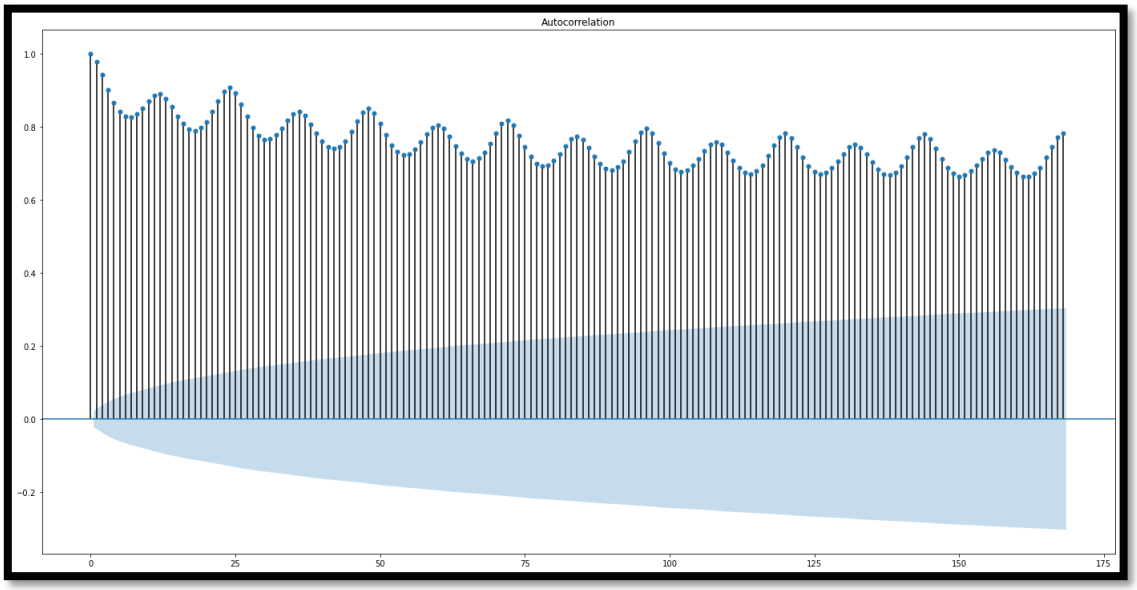

Figure 5-10: Autocorrelation plot.

In the above graph the autocorrelation between the SMP and its lag values up to 168, one week in the past, is depicted. As can be observed there is a repeated pattern in the lag values indicating seasonality.

Likewise in Figure 5-11, where the partial autocorrelation between the SMP and its lag values up to 168 is presented, there are lag values up to 168 showing a repeated pattern while being significantly correlated

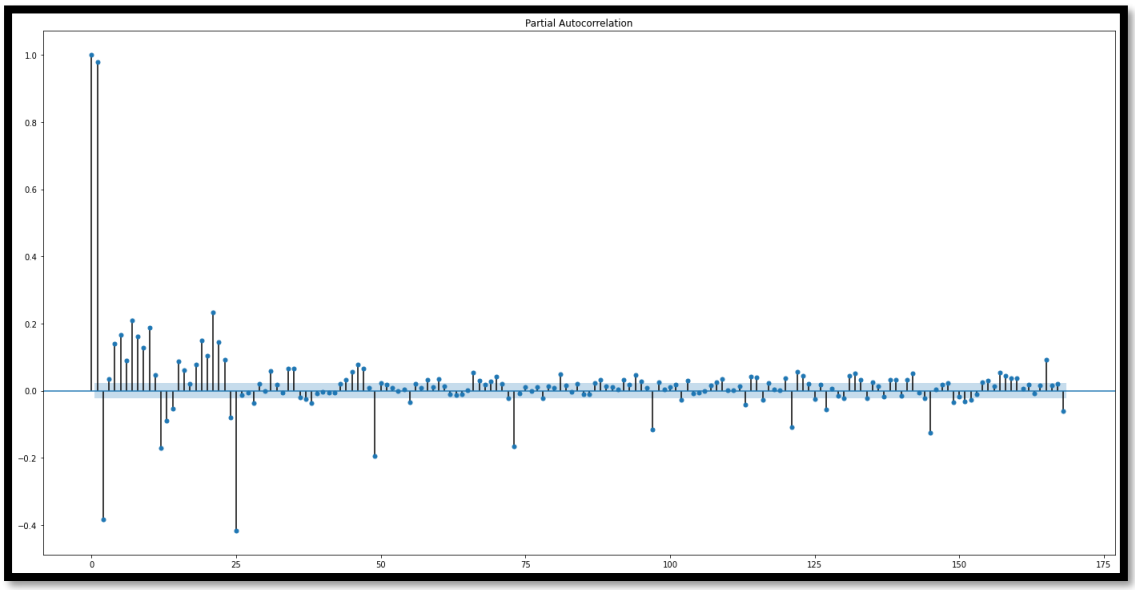

Figure 5-11: Partial Autocorrelation plot.

with the SMP, and hence implying the presence of seasonality in the data. Consequently, there is a strong indication for the time series to be non-stationary. In order to confirm the non-stationarity in the data the ADF test is performed. The statsmodels.tsa.stattools package and adfuller function are used. The results are provided below,

$$Stationarity\ check\ for\ original\ SMP\ data.$$
$$ADF\ test\ statistic: -1.562$$
$$ADF\ p-values: 0.502$$
$$ADF\ number\ of\ lags\ used: 30$$
$$ADF\ number\ of\ observations: 3425$$
$$ADF\ critical\ values: \{'1\%': -3.432,\ '5\%': -2.862,\ '10\%': -2.567$$
$$ADF\ best\ information\ criterion: 23242.157$$

P-value is higher than the 0.05 significance level and the test statistic is higher than the critical values leading to the conclusion that SMP data are non-stationary. With the aim of making the time series stationary a first order of difference is applied on the data and the ADF test is performed again to check the stationarity. The results are provided below,

$$Stationarity\ check\ for\ 1st\ order\ differenced\ SMP.$$
$$ADF\ test\ statistic: -12.670$$
$$ADF\ p-values: 1.246e-23$$
$$ADF\ number\ of\ lags\ used: 29$$
$$ADF\ number\ of\ observations: 3425$$
$$ADF\ critical\ values: \{'1\%': -3.432, '5\%': -2.862, '10\%': -2.567\}$$
$$ADF\ best\ information\ criterion: 23235.773$$

The results show that a 1st order of difference is sufficient to transform the time series from non-stationary to stationary and hence SMP is differenced accordingly.

As a next step, the optimal number of Fourier terms to be used as additional features in the model is determined. To achieve this, two performance evaluation metrices apart from MAPE and RMSE are used. Namely, the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) are estimated and compared. AIC, is a method predominantly applied in autoregression problems and is used for scoring and selecting a model. AIC is defined as follows,

$$AIC = -2 * LL + 2 * n \quad (7)$$

Where, LL is the log-likelihood of the model (measure of fit) and n is the number of parameters in the model. The model with the lowest AIC score is preferred [26]. BIC, is a similar method to AIC and is also used for scoring and selecting a model. BIC is defined as follows,

$$BIC = -2 * LL + log(S) * n \quad (8)$$

Where, LL is again the log-likelihood of the model, S is the number of observations in the training sample, and n is the number of parameters in the model. Likewise, the model with the lowest BIC score is preferred. Continuing, a simple SARIMAX model of order (AR=1, Difference=1, MA=0) is used as a base to test and find the best combination of Fourier terms. A for loop is used wherein the SARIMAX model is created and trained on the training sample as defined in the beginning of sub section 5.4, in each iteration. Before the creation of the SARIMAX model, the dataset is inserted in the fourier_con() function, where the Fourier terms are produced and added in the original dataset as extra features. From the resulting dataset the features are inserted as exogenous variables in the SARIMAX model. The number of iterations range from 0 to 6 and they represent the order of each Fourier term (week, weekday, hour). The model is fit using 1000 max iterations to reach convergence and the optimization algorithm applied is by default the limited memory BFGS. MAPE and RMSE are calculated based on a test sample of two weeks and more precisely the last two weeks of the entire dataset. The statsmodels.api module is used to create the SARIMAX model.

$$model = sm.tsa.statespace.SARIMAX(target, exog = features, order = (1,1,0), enforce_{stationarity} = False, enforce_{invertibility} = False, freq = 'H')$$

The results are stored in a new dataframe with five columns, order (of Fourier terms), MAPE, RMSE, AIC and BIC. Next, a code is developed through which the optimal order of the SARIMAX model is determined. Knowing the optimal order of the Fourier terms from previous step, the fourier_con() function is applied on the dataset. The order(p,d,q) is tested for p,d and q values ranging from 0 to 3. The order values are not tested for higher prices due to the fact that the model would become too complex and result potentially to overfitting. The same training and test samples as in the previous step are used. Below the code developed in python to assess the optimal order is provided.

```
check=input('Do you want to check for optimal order in SARIMAX yes or no ? ')
if check == 'yes':
    MAPE=[]
    RMSE=[]
    AIC=[]
    BIC=[]
    order=[]
    OR=dataset.copy()
    fourier_con(OR, n, n, n)
    endog=pd.DataFrame(data=OR['SMP'],columns=['SMP'])
    exog=OR[OR.columns[-(2*n*3+3):]]
    tr,tst=endog[:'2021-10-08'],endog['2021-10-09':]
    tre,tste=exog[:'2021-10-08'],exog['2021-10-09':]
    tst.replace(to_replace=0,value=np.average(tst),inplace=True)
    tr=tr.resample('H').sum()
```

```
tst=tst.resample('H').sum()
tre=tre.resample('H').sum()
tste=tste.resample('H').sum()
p = d = q = range(0, 4)
pdq = list(itertools.product(p, d, q))
for param in pdq:
    try:
        model = sm.tsa.statespace.SARIMAX(tr,exog=tre,order=param,enforce_stationarity=False,enforce_invertibility=False,freq='H')
        res = model.fit(maxiter=1000,disp=True)
        print('ARIMA{} - AIC:{}'.format(param,res.aic))
        predictions=pd.DataFrame(data=res.forecast(steps=len(tste),exog=tste),columns=['SMP'])
        predictions.set_index(pd.DatetimeIndex(tst.index),inplace=True)
        MAPE.append(np.mean(abs((tst.values-predictions.values))/abs(tst.values))*100)
        RMSE.append(np.sqrt(mean_squared_error(tst, predictions)))
        AIC.append(res.aic)
        BIC.append(res.bic)
        order.append(param)
    except:
        continue

SM=pd.DataFrame(data=list(zip(order,MAPE,RMSE,AIC,BIC)),columns=['order','MAPE','RMSE','AIC','BIC'])
else:
    print('No check.')
```

Finally, after having determined the optimal order of both the Fourier terms and the SARIMAX p, d and q terms the optimal model is created. Two forecasts are made, one extending from 2021/10/09 to 2021/10/22 with a training sample ranging from 2021/06/01 to 2021/10/08 and the other concerning only the last day 2021/10/22 of the dataset, with a training sample ranging from 2021/06/01 to 2021/10/21.

5.4.2 MLP Regressor

As has already been mentioned MLPs can be visualized as a series of connected layers that form a network connecting features' values at one end, and the target variable's value at the other end. MLPs can be applied to time series forecasting and when they have many hidden layers (e.g., 10, 100, 1,000) are deemed as deep networks and their application is called deep learning [18]. Class MLPRegressor implements an MLP, which is trained via backpropagation and with no activation function in the output layer. It uses the square error as the loss function and the output is a set of continuous prices. In

MLPRegressor the squared error is optimized using either the ‘‘LBFGS’’ or the ‘‘stochastic gradient descent’’ or the ‘‘adam’’ solver.

In this approach a simple neural network is designed and applied in the dataset, sklearn.metrics, mean_squared_error, sklearn.neural_network, MLPRegressor, sklearn.preprocessing, MinMaxScaler, datetime, pandas and numpy are used. One of the most important components of a neural network is the number of hidden layers and the corresponding nodes or neurons in each layer. Thus, a function named tune(dataset,layers) is created through which the optimal number of hidden layers and their respective nodes is determined. The normalization of the dataset is performed within the function, early_stopping is set to True so that training terminates when validation score is not improving and all of the remaining parameters are set to default values. The fourier_con() function is applied on the dataset before tuning the MLP. The tune() function evaluates the MLP’s performance through MAPE and RMSE metrices. The number of layers used to optimize the model ranges from 1 to 4 and the number of nodes in each layer is tested from a set of five values [20,50,100,150,200].

```
def tune(dataset,layers):
   if layers==1:
      MAPE=[]
      RMSE=[]
      number=[]
      for i in [20,50,100,150,200]:
         train=data[:'2021-10-8']
         test=data['2021-10-9':]
         #Normalize data
         mnm=MinMaxScaler(feature_range=(0,1)).fit(train.loc[:,train.columns[:-1]])
         train.loc[:,train.columns[:-1]]=mnm.transform(train.loc[:,train.columns[:-1]])
         test.loc[:,test.columns[:-1]]=mnm.transform(test.loc[:,test.columns[:-1]])
         trainx,trainy=train[train.columns[:-1]],train[train.columns[-1]]
         testx,testy=test[test.columns[:-1]],test[test.columns[-1]]
         testy.replace(to_replace=0,value=np.average(testy),inplace=True)
         #Design model's architecture
         model = MLPRegressor(hidden_layer_sizes=(i),
                              activation='relu',
                              solver='adam',
                              alpha=0.0001,
                              batch_size=auto,
                              learning_rate='constant',
                              learning_rate_init=0.001,
                              random_state=7,
                              early_stopping=True
                              )

         #Fit model
         model.fit(trainx,trainy)
         #Prediction and evaluation
         predictions=model.predict(testx)
         MAPE.append(np.mean(abs((testy-predictions))/abs(testy))*100)
         RMSE.append(np.sqrt(mean_squared_error(testy, predictions)))
         number.append(i)
      df=pd.DataFrame(data=list(zip(number,MAPE,RMSE)),columns=['Layers','MAPE','RMSE'])
      return  print(df.loc[df['MAPE']==df['MAPE'].min()])
   elif layers==2:
      MAPE=[]
      RMSE=[]
      number=[]
      for i in [20,50,100,150,200]:
```

```
        for j in [20,50,100,150,200]:
            train=data[:'2021-10-8']
            test=data['2021-10-9':]
            #Normalize data
            mnm=MinMaxScaler(feature_range=(0,1)).fit(train.loc[:,train.columns[:-1]])
            train.loc[:,train.columns[:-1]]=mnm.transform(train.loc[:,train.columns[:-1]])
            test.loc[:,test.columns[:-1]]=mnm.transform(test.loc[:,test.columns[:-1]])
            trainx,trainy=train[train.columns[:-1]],train[train.columns[-1]]
            testx,testy=test[test.columns[:-1]],test[test.columns[-1]]
            testy.replace(to_replace=0,value=np.average(testy),inplace=True)
            #Design model's architecture
            model = MLPRegressor(hidden_layer_sizes=(i,j),
                                 activation='relu',
                                 solver='adam',
                                 alpha=0.0001,
                                 batch_size=auto,
                                 learning_rate='constant',
                                 learning_rate_init=0.001,
                                 random_state=7,
                                 early_stopping=True
                                 )

            #Fit model
            model.fit(trainx,trainy)
            #Prediction and evaluation
            predictions=model.predict(testx)
            MAPE.append(np.mean(abs((testy-predictions))/abs(testy))*100)
            RMSE.append(np.sqrt(mean_squared_error(testy, predictions)))
            number.append((i,j))
    df=pd.DataFrame(data=list(zip(number,MAPE,RMSE)),columns=['Layers','MAPE','RMSE'])
    return  print(df.loc[df['MAPE']==df['MAPE'].min()])
elif layers==3:
    MAPE=[]
    RMSE=[]
    number=[]
    for i in [20,50,100,150,200]:
        for j in [20,50,100,150,200]:
            for k in [20,50,100,150,200]:
                train=data[:'2021-10-8']
                test=data['2021-10-9':]
                #Normalize data
                mnm=MinMaxScaler(feature_range=(0,1)).fit(train.loc[:,train.columns[:-1]])
                train.loc[:,train.columns[:-1]]=mnm.transform(train.loc[:,train.columns[:-1]])
                test.loc[:,test.columns[:-1]]=mnm.transform(test.loc[:,test.columns[:-1]])
                trainx,trainy=train[train.columns[:-1]],train[train.columns[-1]]
                testx,testy=test[test.columns[:-1]],test[test.columns[-1]]
                testy.replace(to_replace=0,value=np.average(testy),inplace=True)
                #Design model's architecture
                model = MLPRegressor(hidden_layer_sizes=(i,j,k),
                                     activation='relu',
                                     solver='adam',
                                     alpha=0.0001,
                                     batch_size=auto,
                                     learning_rate='constant',
                                     learning_rate_init=0.001,
```

```
                        random_state=7,
                        early_stopping=True
                        )

            #Fit model
            model.fit(trainx,trainy)
            #Prediction and evaluation
            predictions=model.predict(testx)
            MAPE.append(np.mean(abs((testy-predictions))/abs(testy))*100)
            RMSE.append(np.sqrt(mean_squared_error(testy, predictions)))
            number.append((i,j,k))
    df=pd.DataFrame(data=list(zip(number,MAPE,RMSE)),columns=['Layers','MAPE','RMSE'])
    return  print(df.loc[df['MAPE']==df['MAPE'].min()])
elif layers==4:
    MAPE=[]
    RMSE=[]
    number=[]
    for i in [20,50,100,150,200]:
        for j in [20,50,100,150,200]:
            for k in [20,50,100,150,200]:
                for l in [20,50,100,150,200]:
                    train=data[:'2021-10-8']
                    test=data['2021-10-9':]
                    #Normalize data
                    mnm=MinMaxScaler(feature_range=(0,1)).fit(train.loc[:,train.columns[:-1]])
                    train.loc[:,train.columns[:-1]]=mnm.transform(train.loc[:,train.columns[:-1]])
                    test.loc[:,test.columns[:-1]]=mnm.transform(test.loc[:,test.columns[:-1]])
                    trainx,trainy=train[train.columns[:-1]],train[train.columns[-1]]
                    testx,testy=test[test.columns[:-1]],test[test.columns[-1]]
                    testy.replace(to_replace=0,value=np.average(testy),inplace=True)
                    #Design model's architecture
                    model = MLPRegressor(hidden_layer_sizes=(i,j,k,l),
                                         activation='relu',
                                         solver='adam',
                                         alpha=0.0001,
                                         batch_size=auto,
                                         learning_rate='constant',
                                         learning_rate_init=0.001,
                                         random_state=7,
                                         early_stopping=True
                                         )

                    #Fit model
                    model.fit(trainx,trainy)
                    #Prediction and evaluation
                    predictions=model.predict(testx)
                    MAPE.append(np.mean(abs((testy-predictions))/abs(testy))*100)
                    RMSE.append(np.sqrt(mean_squared_error(testy, predictions)))
                    number.append((i,j,k,l))
    df=pd.DataFrame(data=list(zip(number,MAPE,RMSE)),columns=['Layers','MAPE','RMSE'])
    return  print(df.loc[df['MAPE']==df['MAPE'].min()])
else:
    print('Give layers number between 1 and 4.')
```

The above function accepts as input a dataframe, with the last column being the target variable, and the number of hidden layers. After having determined the optimal number of hidden layers and their respective nodes, the MLP is further optimized and tested for different values of alpha, batch_size and learning_rate_init. Two forecasts are made, one extending from 2021/10/09 to 2021/10/22 with a training sample ranging from 2021/06/01 to 2021/10/08 and the other concerning only the last day 2021/10/22 of the dataset, with a training sample ranging from 2021/06/01 to 2021/10/21.

5.4.3 RNN - LSTM

The main feature of a RNN is that information loops back in the network. This provides RNNs with a type of memory that enables them to process sequential data in a more efficient way. A popular type of a RNN is the LSTM. Sequence data, stored in 3D tensors of shape (samples, timesteps, features), is typically processed by recurrent layers such as an LSTM layer. Like an MLP, LSTM is comprised of different layers of neurons and input data is propagated through the network in order to make a prediction. Like RNNs, LSTMs possess recurrent connections, so that the state of the neuron from a previous time step is used as context for formulating an output at the next time step. Unlike other RNNs, the LSTM has a unique design which allows it to avoid problems preventing the training and scaling that other RNNs face. As has already been discussed in 3.2 sub section, experiments show how difficult is to effectively train a RNN, due to the weight update process leading to weight changes that quickly become too small and hence have no effect, the so called ‘‘vanishing gradient problem’’, or so large as to result in sharp changes experiencing exploding gradients. LSTMs overcome this problem by design [21].

In this approach three types of LSTM networks are applied and evaluated, the Vanilla LSTM, the Stacked LSTM and the Bidirectional LSTM. A Vanilla LSTM is a model that has a single hidden layer of LSTM units and an output layer used to make predictions. A Stacked LSTM is a model where multiple hidden LSTM layers are stacked one on top of another. An LSTM layer requires a three-dimensional input and LSTMs by default produce a two-dimensional output as an interpretation from the end of the sequence. This is addressed by setting the return sequences argument to True. This allows us to have a 3D output from the hidden LSTM layer as input to the next. A Bidirectional LSTM model learns the input sequence both forward and backwards and concatenates both interpretations, which could prove beneficial for a number of sequence prediction problems. For the implementation of the aforementioned LSTM models, tensorflow and keras libraries are required. Tensorflow is an open-source end-to-end platform, a library for multiple machine learning tasks, while keras runs on top of tensorflow and is a deep learning framework for Python that provides a convenient way to define and train almost any kind of deep learning models.

A function named split_data(dataframe,timesteps) is created with the aim of splitting the dataset into samples based on a given timestep and ultimately return the data as numpy arrays ready to be used as input in the LSTM model.

```
def split_data(dataframe, timesteps):
  Train=[]
  Target=[]
  for i in range(len(dataframe)):
    end = i + steps
    if end > len(dataframe):
      break
    seq_tr, seq_ta = dataframe.iloc[i:end, :-1], dataframe.iloc[end-1, -1]
    Train.append(seq_tr)
    Target.append(seq_ta)
  return np.array(Train), np.array(Target)
```

The function accepts as inputs a dataframe and the number of timesteps and returns as output two numpy arrays, one comprised of samples concerning features and the other comprised of samples concerning the target variable (SMP). The time series is fed in the fourier_con() function to add week, weekday and hour features. The order of Fourier terms is the same as the order used in SARIMAX and MLP Regressor

approaches. As a next step the time series is split into a training and a test sample and subsequently fed in the split_data() function. Training and test samples are the same as the ones used in the two previous approaches. The dataset is then ready to train and validate the LSTM models.

Vanilla LSTM model architecture

$$model = Sequential()$$
$$model.add(LSTM(number\ of\ LSTM\ units, activation = 'relu', input\_shape = (timesteps, number\ of\ features)))$$
$$model.add(Dense(1),)$$
$$model.compile(optimizer = 'adam', loss = 'mse')$$

Stacked LSTM model architecture

$$model = Sequential()$$
$$model.add(LSTM(number\ of\ LSTM\ units, activation = 'relu', return\_sequences = True, input\_shape = (timesteps, number\ of\ features)))$$
$$model.add(LSTM(number\ of\ LSTM\ units, activation = 'relu', return\_sequences = True, input\_shape = (timesteps, number\ of\ features)))$$
.
.
.
$$model.add(LSTM(number\ of\ LSTM\ units, activation = 'relu'))$$
$$model.add(Dense(1))$$
$$model.compile(optimizer = 'adam', loss = 'mse')$$

Bidirectional LSTM model architecture

$$model = Sequential()$$
$$model.add(Bidirectional(LSTM(number\ of\ LSTM\ units, activation = 'relu'), input\_shape = (timesteps, number\ of\ features)))$$
$$model.add(Dense(1))$$
$$model.compile(optimizer = 'adam', loss = 'mse')$$

All the models are tested for different LSTM units, batch_size and epochs while Stacked LSTM is also tested for a number of stacked LSTM layers. The number of timesteps is set to 1, in order to maintain consistency in the implementation of the different machine learning algorithms. These tests are evaluated based on the respective MAPE and RMSE results, where the architecture of the model producing the lowest MAPE and RMSE values (at the same time achieving low training and validation loss), is selected. Two forecasts are made, one extending from 2021/10/09 to 2021/10/22 with a training sample ranging from 2021/06/01 to 2021/10/08 and the other concerning only the last day 2021/10/22 of the dataset, with a training sample ranging from 2021/06/01 to 2021/10/21.

6 Experimental Results

In this section the evaluation results from the implementation of the already discussed in the methodology machine learning algorithms are analysed. In the below table the final dataset pertaining to the training sample is presented, after having added the optimal Fourier terms and after having performed the MinMaxscaler normalization.

Datetime Index	SMP-24	HU	Residual	weekday _sin1	weekday _cos1	...	hour_sin 6	hour_cos 6	SMP
01/06/20 21 00:00	0.050	0.154	0.380	0.901	0.802	...	0.500	1.000	73.724
01/06/20 21 01:00	0.043	0.144	0.325	0.901	0.802	...	1.000	0.500	71.208
01/06/20 21 02:00	0.040	0.139	0.306	0.901	0.802	...	0.500	0.000	66.924
01/06/20 21 03:00	0.036	0.135	0.296	0.901	0.802	...	0.000	0.500	64.761
01/06/20 21 04:00	0.039	0.139	0.289	0.901	0.802	...	0.500	1.000	65.310
…	...	...	...	...	...	...	...	...	…
08/10/20 21 19:00	0.977	0.629	0.813	0.277	0.000	...	0.000	0.500	312.700
08/10/20 21 20:00	0.868	0.550	0.767	0.277	0.000	...	0.500	1.000	254.620
08/10/20 21 21:00	0.815	0.453	0.692	0.277	0.000	...	1.000	0.500	230.640
08/10/20 21 22:00	0.763	0.423	0.602	0.277	0.000	...	0.500	0.000	205.040
08/10/20 21 23:00	0.716	0.350	0.510	0.277	0.000	...	0.000	0.500	190.950

Table 2: Final form of the training sample.

The optimal order of the Fourier terms was found to be 6 for weekday and 6 for hour, the week term was excluded as redundant and hence 24 Fourier terms are added as extra features in the dataset, resulting to 27 features in total.

6.1 SARIMAX Results

The optimal order of the SARIMAX model was found to be p=2, d=1, q=3 and the autoregression model was created as shown below,

$$model = sm.tsa.statespace.SARIMAX(SMP, exog = features, order = (2,1,3), enforce\_stationarity = False, enforce\_invertibility = False, freq = 'H')$$

As is discussed in the methodology section the ADF test showed that a 1st order of difference is sufficient to detrend the time series and make them stationary. The AR and MA terms were chosen with values of 2 and 3 respectively as the model scored the lowest AIC while producing relatively low levels of BIC, MAPE and RMSE. An alternative order option would be order(1,1,1), where BIC in this case has the lowest score among all different combinations, nevertheless order(2,1,3) was chosen due to better diagnostic results. In the below graph the diagnostics of the residuals are presented.

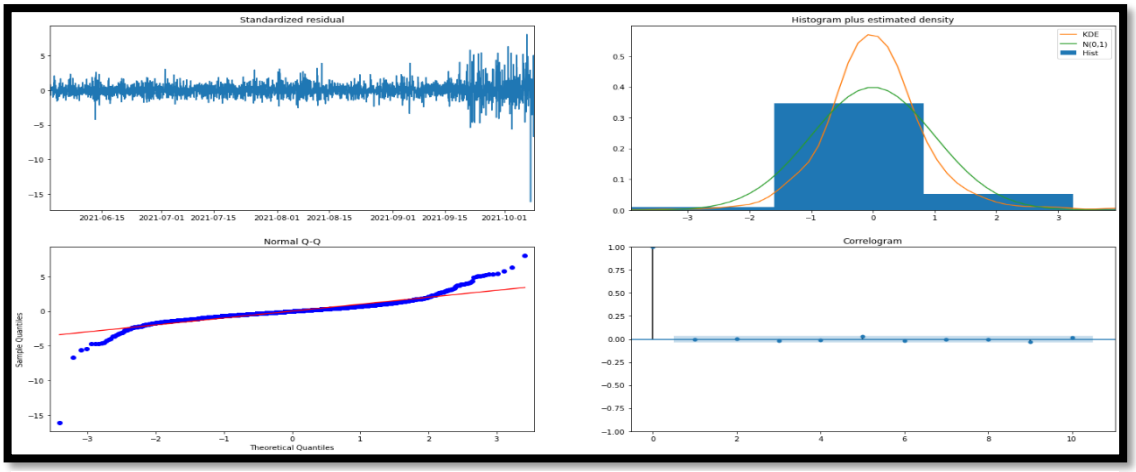

Figure 6-1: Diagnostics plot of SARIMAX model.

In the top left graph of Figure 6-1 the residuals don't display over time any obvious seasonality and appear to be white noise. This is confirmed by the Correlogram plot on the bottom right, which shows a low correlation between the time series residuals and the lagged versions of itself. In the Q-Q plot all the dots, apart from one extreme point on the bottom left, fall in line with the red line indicating that the residuals are normally distributed. Thus, it could be concluded that the chosen model overall produces a satisfactory fit.

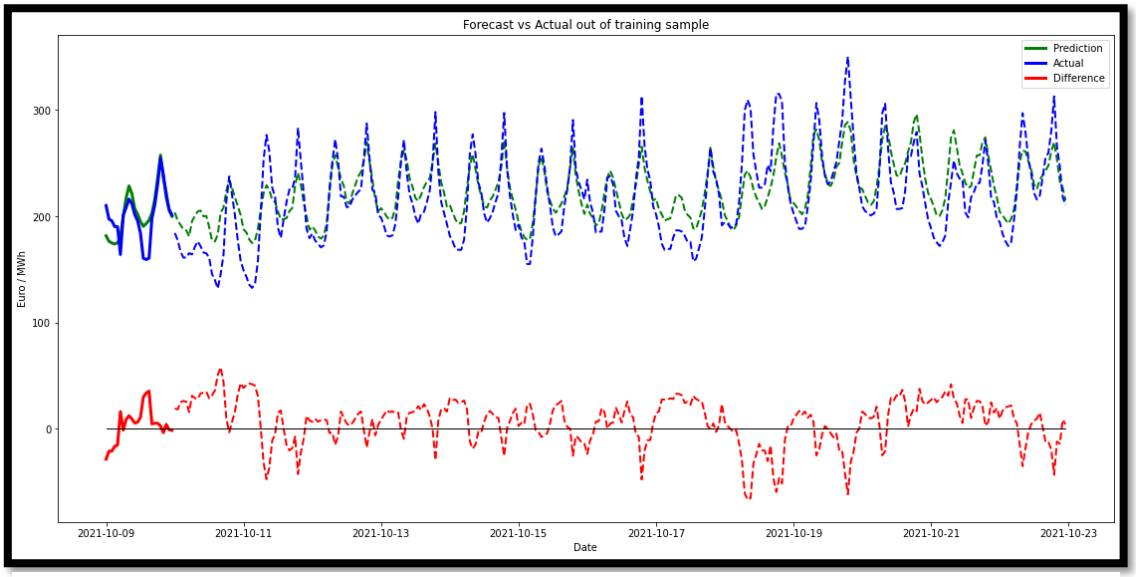

Figure 6-2: SARIMAX two-week ahead forecast against actual prices.

In the above graph the forecast results against the actual prices are depicted. The forecast covers a two-week period from 2021/10/09 to 2021/10/22, and the solid lines represent the Day-Ahead time horizon. Blue line reflects actual SMP prices, green line reflects forecast SMP prices and the red line indicates the difference between forecast and actual prices. MAPE equals 8.631 per cent and RMSE 21.852 Euro / MWh. With regard to Day-Ahead (2021-10-09), MAPE equals 6.797 per cent and RMSE 16.307 Euro / MWh. Finally, a training sample ranging from 2021/06/01 to 2021/10/21 is used to forecast

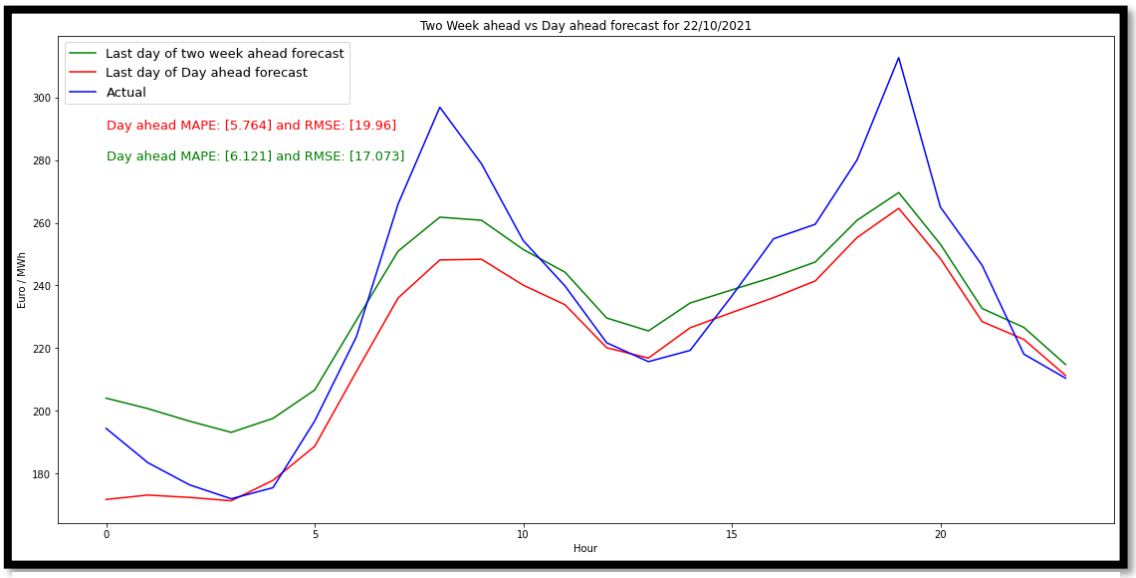

Figure 6-3:SARIMAX forecasts comparison for 2021/10/22.

the 24 SMP hourly values of the last day in the entire dataset. As can be seen from Figure 6-3 there is a slight improvement in the MAPE value concerning the last forecast, when comparing this with the two-week forecast. However, the RMSE worsens by approximately 3 Euro / MWh.

6.2 MLP Regressor Results

As has been already described in the methodology section through the tune(dataframe, number of hidden layers) function the optimal number of hidden layers was determined to four with 20, 100, 20 and 100 nodes in each layer accordingly. Subsequently, the model's parameters were tested for a number of different values and the architecture of the model final used to perform forecasts is given as per below,

$$
\begin{aligned}
model = MLPRegressor(&hidden\_layer\_sizes = (20,100,20,100),\\
&activation = 'relu',\\
&solver = 'adam',\\
&alpha = 0.002,\\
&batch\_size = 20,\\
&learning\_rate = 'constant',\\
&learning\_rate\_init = 0.0008,\\
&random\_state = 7,\\
&early\_stopping = True)
\end{aligned}
$$

Adam solver, relu activation function, 0.002 alpha, 20 batch_size and a learning_rate_init of 0.0008 were used. Thus, based the training sample as presented in Table 2 a two-week forecast was performed.

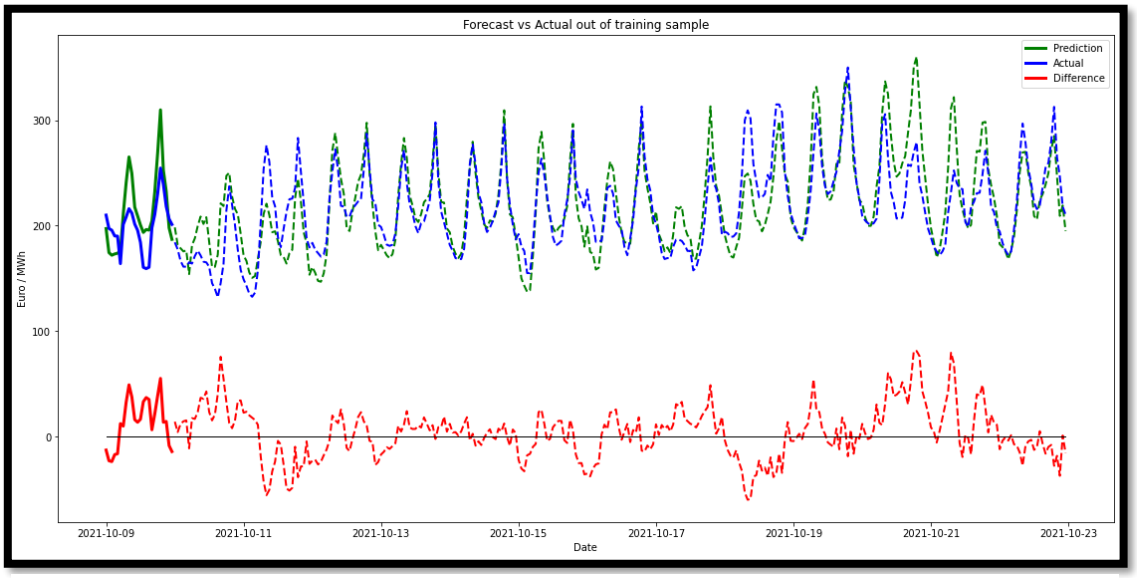

Figure 6-4: MLP two-week ahead forecast against actual prices.

In the above graph the forecast results against the actual prices are depicted. The forecast covers a two-week period from 2021/10/09 to 2021/10/22, and the solid lines represent the Day-Ahead time horizon. Blue line reflects actual SMP prices, green line reflects forecast SMP prices and the red line indicates the difference between forecast and actual prices. MAPE equals 8.888 per cent and RMSE 24.644 Euro / MWh. With regard to Day-Ahead (2021/10/09), MAPE equals 11.706 per cent and RMSE 26.669 Euro / MWh. Finally, a training sample ranging from 2021/06/01 to 2021/10/21 is used to forecast the 24 SMP hourly values of the last day in the entire dataset. As can be seen from Figure 6-5 in the next page, there is a drop in performance concerning the last forecast when comparing this with the two-week forecast, which is concluded from both MAPE and RMSE values. This drop in performance could be attributed to overfitting.

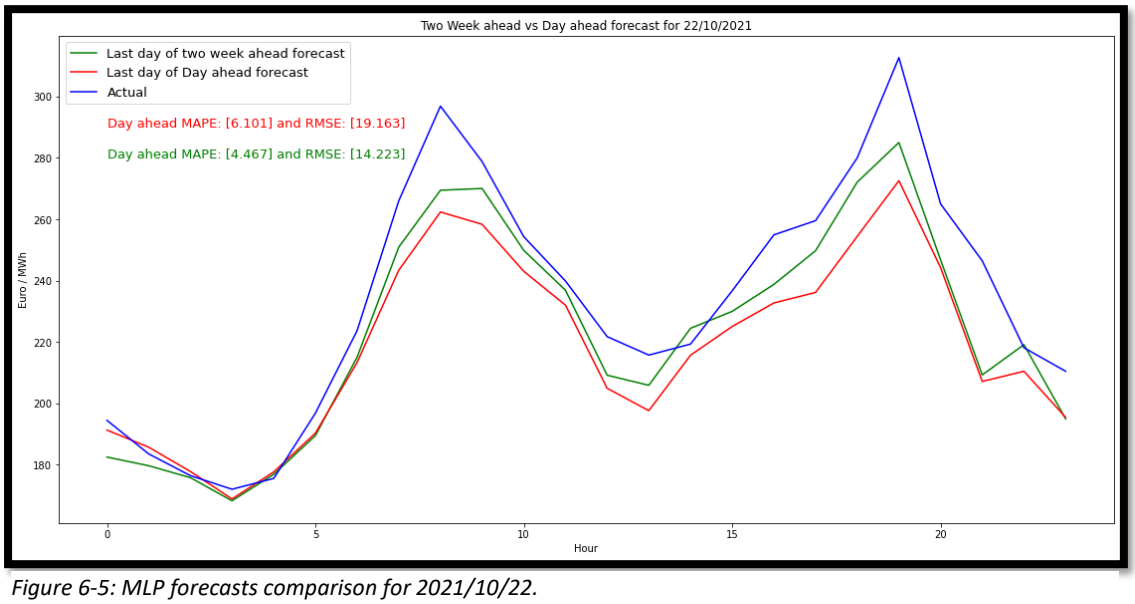

Figure 6-5: MLP forecasts comparison for 2021/10/22.

6.3 LSTM Results

In this sub section the results of the three LSTM models applied are presented. The training sample used is the same with the training samples used in both the SARIMAX and the MLP Regressor methods.

Vanilla LSTM

After having tested the model's architecture for a number of different LSTM units and subsequently fit it with a number of different batch sizes and epochs, the ultimate model used and its parameters are given,

$$model = Sequential()$$
$$model.add(LSTM(100, activation = 'relu', input\_shape = (1, 27)))$$
$$model.add(Dense(1))$$
$$model.compile(optimizer = 'adam', loss = 'mse')$$
$$M = model.fit(features, target, batch\_size = 20, epochs = 50, validation\_data = (testx, testy), verbose = 0, shuffle = False)$$

Thus, 100 LSTM units, relu activation function, 1 timestep, 27 features, adam optimizer, mean square error loss function, 20 batch_size and 50 epochs were used. The model was validated based on a test sample of data extending from 2021/10/09 to 2021/10/22. The graph in the next page shows the forecast results against the actual prices. The forecast covers a two-week period from 2021/10/09 to 2021/10/22, and the solid lines represent the Day-Ahead time horizon. Blue line reflects actual SMP prices, green line reflects forecast SMP prices and the red line indicates the difference between forecast and actual prices. MAPE equals 7.484 per cent and RMSE 19.873 Euro / MWh. With regard to Day-Ahead (2021/10/09), MAPE equals 8.113 per cent and RMSE 17.761 Euro / MWh.

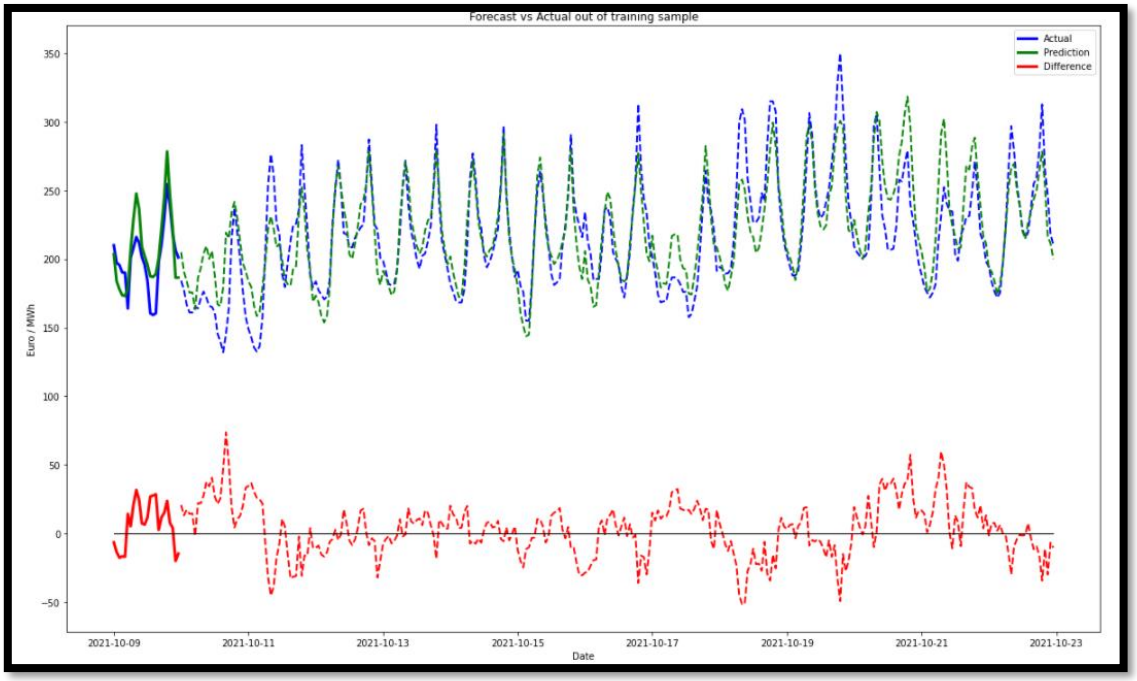

Figure 6-6: Vanilla LSTM two-week ahead forecast against actual prices.

So far Vanilla LSTM seem to fit better the data and provide more robust forecasts. Next, a training sample ranging from 2021/06/01 to 2021/10/21 is used to forecast the 24 SMP hourly values of the last day in the entire dataset. As can be seen from Figure 6-7 there is no significant change in the performance of the two forecasts. The highest deviation between forecast and actual values appears to be during the peak periods, where SMP values reach their highest levels.

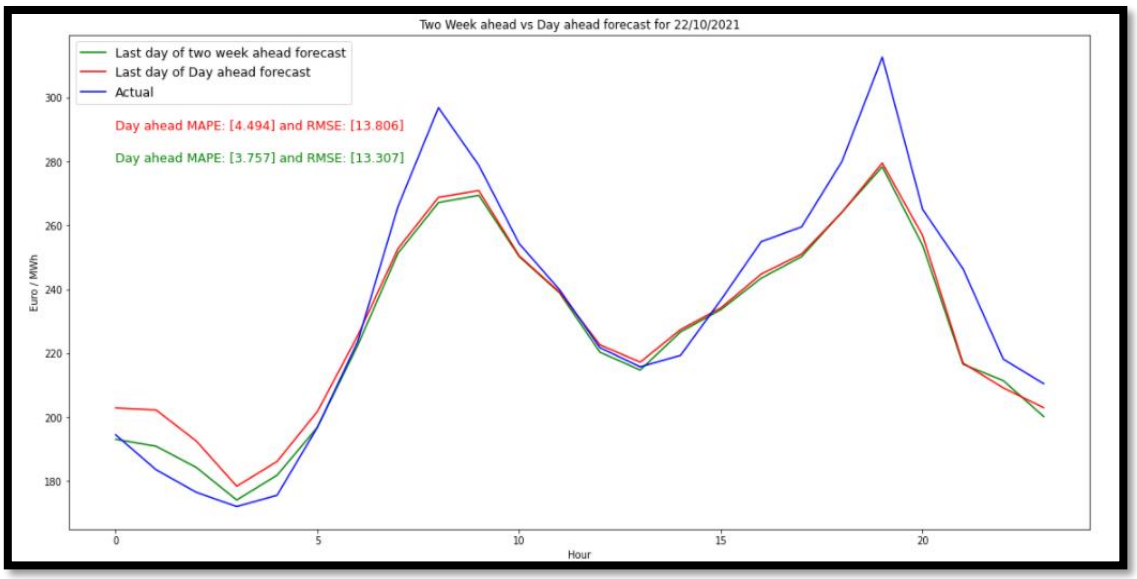

Figure 6-7: Vanilla LSTM forecasts comparison for 2021/10/22.

Stacked LSTM

After having tested the model's architecture for a number of different LSTM units and layers and subsequently fit it with a number of different batch sizes and epochs, the ultimate model used and its parameters are given,

$$model = Sequential()$$
$$model.add(LSTM(50, activation = 'relu', return\_sequences = True, input\_shape = (1, 27)))$$
$$model.add(LSTM(30, activation = 'relu', return\_sequences = True, input\_shape = (1, 27)))$$
$$model.add(LSTM(20, activation = 'relu'))$$
$$model.add(Dense(1))$$
$$model.compile(optimizer = 'adam', loss = 'mse')$$
$$M = model.fit(features, target, batch\_size = 35, epochs = 35, validation\_data = (testx, testy), verbose = 0, shuffle = False)$$

Thus, 3 Stacked LSTM layers were used with 50, 30 and 20 LSTM units respectively, 1 timestep and 27 features. The relu activation function was applied in each LSTM layer. The model was compiled with the adam optimizer and the mean square error loss function. In contrast to Vanilla LSTM, Stacked LSTM model was fit with 35 batch_size and 35 epochs. The model was validated based on a test sample of data extending from 2021/10/09 to 2021/10/22.

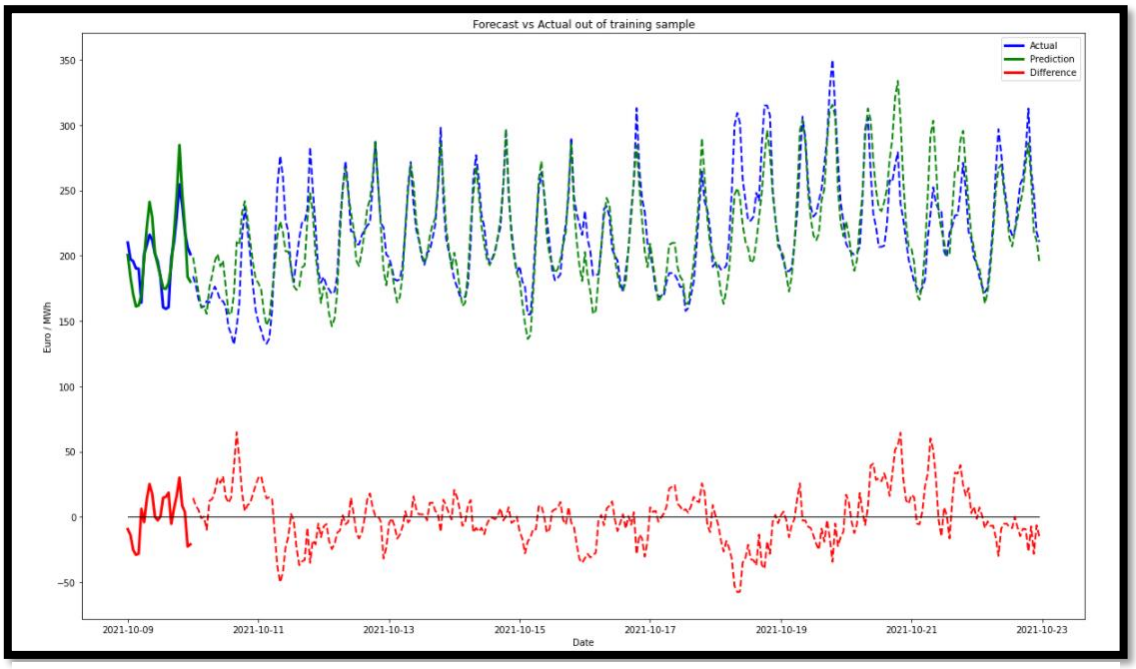

Figure 6-8: Stacked LSTM two-week ahead forecast against actual prices.

The above graph shows the forecast results against the actual prices. The forecast covers a two-week period from 2021/10/09 to 2021/10/22, and the solid lines represent the Day-Ahead time horizon. Blue line reflects actual SMP prices, green line reflects forecast SMP prices and the red line indicates the difference between forecast and actual prices. MAPE equals 6.965 per cent and RMSE 19.628 Euro / MWh. With regard to Day-Ahead (2021/10/09), MAPE equals 7.085 per cent and RMSE 16.869 Euro / MWh. As can be seen, the Stacked LSTM model outperforms all the aforementioned in this section models pertaining to the 2 week-ahead forecast.

Finally, a training sample ranging from 2021/06/01 to 2021/10/21 is used to forecast the 24 SMP hourly values of the last day in the entire dataset. As can be seen from Figure 6-9 below, there is an improvement in performance concerning the last forecast when comparing this with the two-week forecast, which is concluded from both MAPE and RMSE values. RMSE specifically drops by approximately 3 Euro / MWh. It can be clearly observed that this improvement in performance is predominantly owe to a better

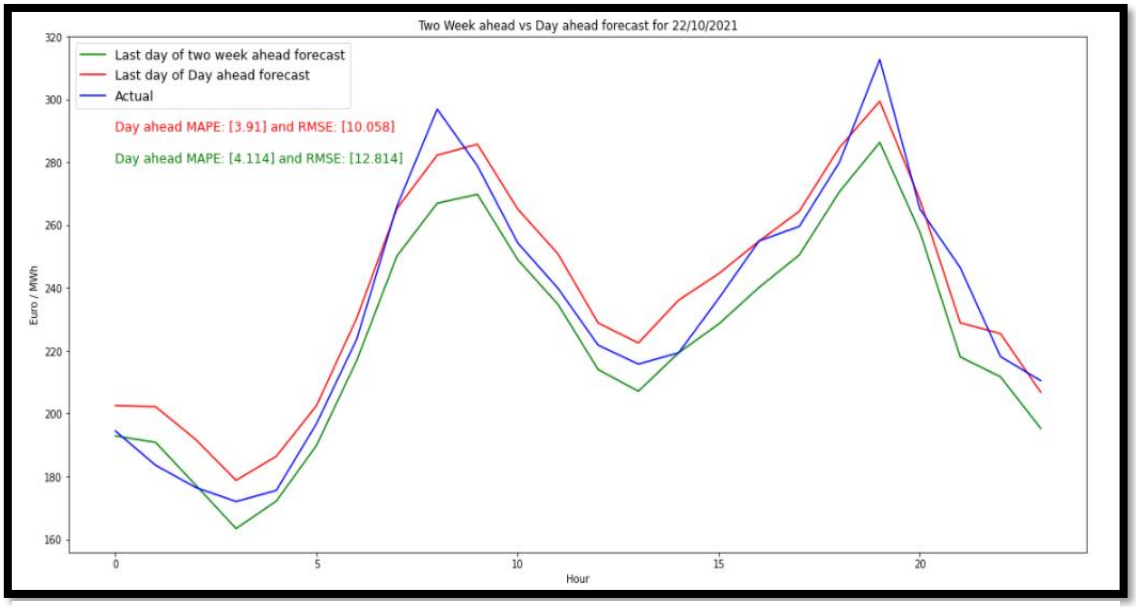

Figure 6-9: Stacked LSTM forecasts comparison for 2021/10/22.

forecast in SMP hourly values during the peak period.

Bidirectional LSTM

After having tested the model's architecture for a number of different LSTM units and subsequently fit it with a number of different batch sizes and epochs, the ultimate model used and its parameters are given,

$$model = Sequential()$$
$$model.add(Bidirectional(LSTM(150, activation = 'relu'), input\_shape = (1, 27)))$$
$$model.add(Dense(1))$$
$$model.compile(optimizer = 'adam', loss = 'mse')$$
$$M = model.fit(features, target, batch\_size = 30, epochs = 55, validation\_data = (testx, testy), verbose = 0, shuffle = False)$$

Thus, 50 LSTM units, relu activation function, 1 timestep, 27 features, adam optimizer, mean square error loss function, 30 batch_size and 55 epochs were used. The model was validated based on a test sample of data extending from 2021/10/09 to 2021/10/22. The graph in the next page shows the forecast results against the actual prices. The forecast covers a two-week period from 2021/10/09 to 2021/10/22, and the solid lines represent the Day-Ahead time horizon. Blue line reflects actual SMP prices, green line reflects forecast SMP prices and the red line indicates the difference between forecast and actual prices. MAPE equals 7.249 per cent and RMSE 20.237 Euro / MWh. With regard to Day-Ahead (2021/10/09), MAPE equals 7.363 per cent and RMSE 17.104 Euro / MWh.

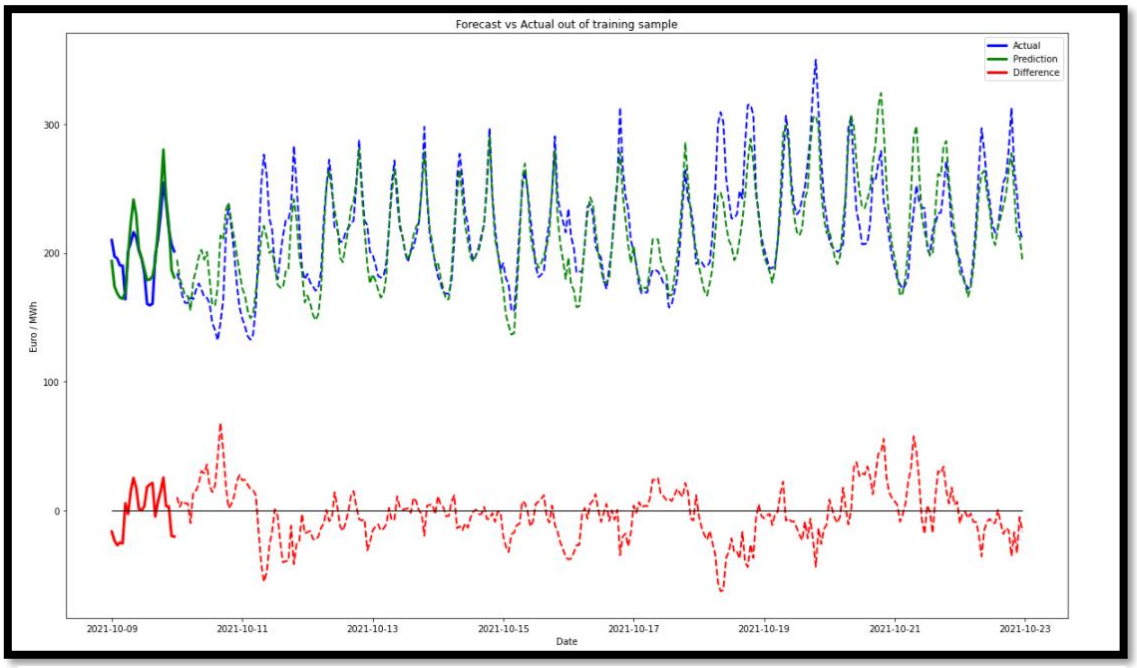

Figure 6-10: Bidirectional LSTM two-week ahead forecast against actual prices.

Next, a training sample extending from 2021/06/01 to 2021/10/21 is used to forecast the 24 SMP hourly values of the last day in the entire dataset. As can be noticed in the Figure below, there is an improvement in performance concerning the last forecast when comparing this with the two-week forecast, which is concluded from both MAPE and RMSE values. RMSE metric specifically presents a significant drop by approximately 6 Euro / MWh. Likewise with Stacked LSTM, it can be clearly observed that this

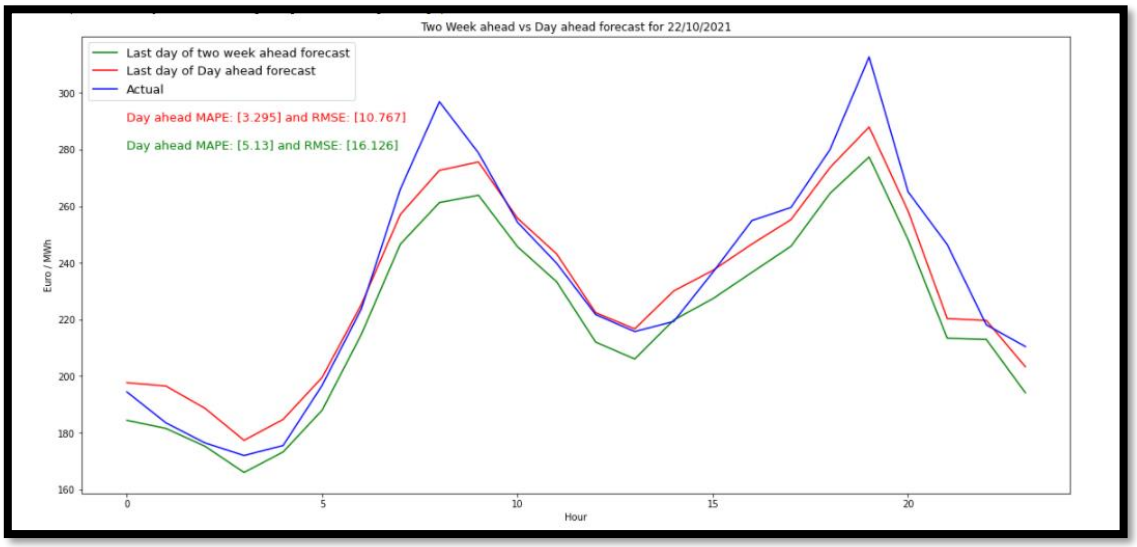

Figure 6-11: Bidirectional LSTM forecasts comparison for 2021/10/22.

improvement in performance is predominantly owe to a better forecast in SMP hourly values during the peak period.

6.4 Synopsis of Results

Three types of machine learning models were created, tested and evaluated accordingly. One autoregressive model, one simple MLP model and three types of LSTM model. A training sample extending from 2021/06/01 to 2021/10/08 having 3120 records and a test sample extending from 2021/10/09 to 2021/10/22 were used to produce one Day-Ahead forecast up to two week-ahead in order to be able to evaluate properly the forecast results. The results were evaluated based on the respective MAPE and RMSE values.

D+1 (2021/10/09)	**MAPE (%)**	**RMSE (Euro / MWh)**
SARIMAX	6.797	16.307
MLP Regressor	11.706	26.669
Vanilla LSTM	8.113	17.761
Stacked LSTM	7.085	16.869
Bidirectional LSTM	7.363	17.104

Table 3: Machine Learning Algorithms evaluation results for Day-Ahead forecast.

D+14 (2021/10/09-2021/10/22)	**MAPE (%)**	**RMSE (Euro / MWh)**
SARIMAX	8.631	21.852
MLP Regressor	8.888	24.644
Vanilla LSTM	7.484	19.873
Stacked LSTM	6.965	19.628
Bidirectional LSTM	7.249	20.237

Table 4: Machine Learning Algorithms evaluation results for two week-ahead forecast.

In Table 3, SARIMAX and Stacked LSTM models indicate possessing the greatest potential having the lowest MAPE and RMSE values concerning the Day-Ahead forecast, nevertheless when taking into account a two week-ahead forecast all types of the LSTM model seem to outperform SARIMAX and MLP Regressor, with the Stacked LSTM showing the lowest MAPE and RMSE values among all. Furthermore, as has been discussed in the previous sub sections LSTM models indicate signs of improvement with more data becoming available for training. That is, either Stacked LSTM or Bidirectional LSTM present a drop in MAPE and RMSE prices when using a training sample from 2021/06/01 to 2021/10/21 with the view to forecasting the last day, 2021/10/22, of the dataset and whether comparing these results with the respective ones produced by the two-week ahead forecast. In general, the most significant forecast deviations are present during peak periods and for a few hours. Overall, all models' results seem to be promising when considering a range of 130-350 Euro / MWh in the training sample pertaining to the actual SMP prices, with the Stacked LSTM's results though being the most prominent.

As has already been mentioned two of the features used in our models, HU and Residual, must be forecasted for future use. As a result, apart from the forecast error discussed earlier there is an additional uncertainty stemming from HU and Residual features that should be investigated through a sensitivity analysis. Due to the fact that Stacked LSTM produced the most prominent results, the sensitivity analysis

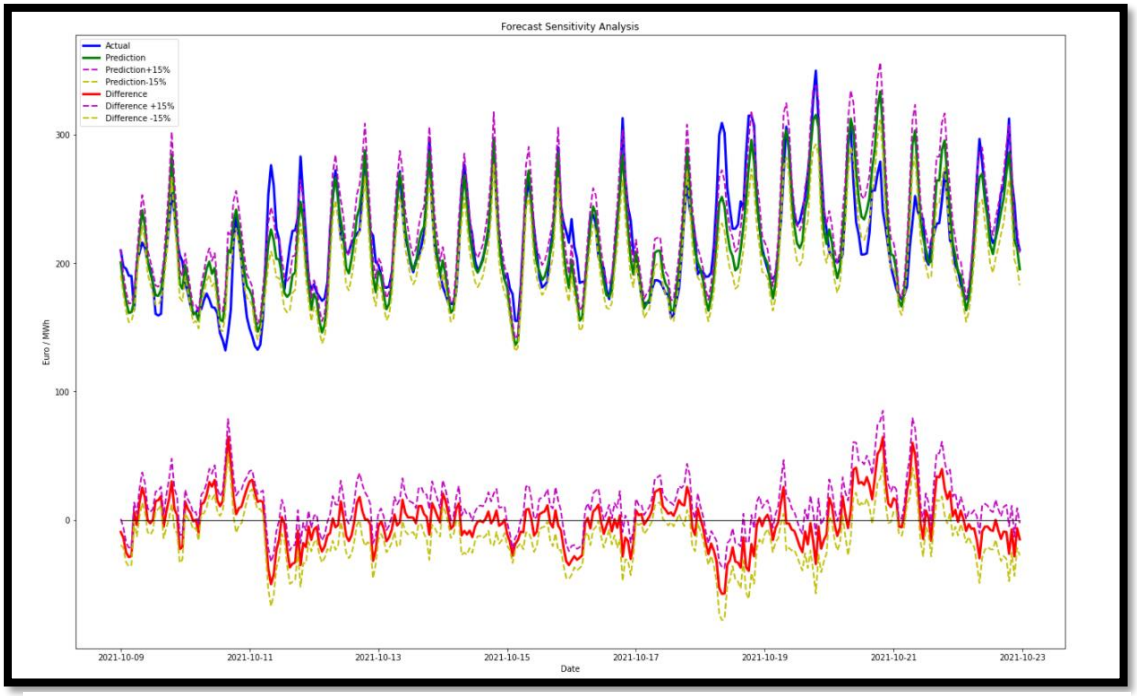

Figure 6-11: Sensitivity Analysis on Stacked LSTM.

was performed on it. It was decided to alter by +15 per cent and by -15 per cent the values of HU and Residual concerning the same test sample used for the two week-ahead forecast. As can be observed by the above graph, the performance of the model deteriorates. Specifically, for a +15 per cent change MAPE and RMSE become 8.720 per cent and 23.648 Euro /MWh correspondingly, while for a -15 per cent change MAPE and RMSE become 8.856 per cent and 24.561 Euro / MWh respectively.

7 Discussion

The aim of this research was to build suitable and robust machine learning models in order to provide accurate forecasts on the Italian Day Ahead SMP, moreover the methods applied to constitute a reference point for further studies in this area. In recent years, a lot of research has been carried out pertaining to the prediction of the Day-Ahead SMP for a number of European countries. This academic work could be classified, based on literature review, into autoregressive models and neural network models. The vast majority of them don't take into account the SMP of neighbouring countries in respect to the under-examination country. The energy conditions in major energy hubs in Europe such as Germany, France and Hungary, were always affecting the formation of other European countries' wholesale power prices. However, when considering the EU Target Model implications, coupling

Figure 7-1: Italian Transmission Network [27].

between countries has led to even higher interdependence among them evincing the need for any new price forecasting attempt to take into account the respective countries' SMP. Italy is one of the most prominent energy markets in southern Europe, yet very few studies on its electricity price prediction have been conducted. Furthermore, the Italian Day-Ahead Market hosts most of the electricity sale and purchase transactions in GME. These are the main reasons why Italy and the Day-Ahead market were selected in our investigation framework. As a next step, an initial set of attribute candidates were chosen,

- FR SMP.
- GER SMP.
- HU SMP.
- Residual.
- Congestion.
- SMP lag values.
- Calendar Characteristics.

For the purposes of this study, historical data of the first four attributes plus the Italian SMP were collected from various sources online. Congestion, SMP lag values and Calendar Characteristics were constructed. Three attribute selection methods were applied to eliminate any possible redundant features,

the SFS, the Random Forest Regressor and the F_regression. It was found that HU SMP, Residual and 24 lag values of the Italian SMP were the most important features. FR SMP and GER SMP were also important, nevertheless they were highly correlated with the Hungarian SMP and since the Hungarian SMP presented better results they were discarded. Italy has six zones and congestion events occur frequently, resulting in different zonal prices. The attribute selection method results showed though that Congestion was not contributing significantly to model's performance and hence was discarded as well. Calendar Characteristics were not included in this process due to the fact that were constructed to handle multiple seasonality in the time series. Data used in this thesis are essentially a multivariate time series which present multiple seasonality. Thus, the final attributes determined to train our models with were HU SMP, Residual, SMP-24 and 24 Fourier terms used to handle seasonality. One of the main challenges of this study was the determination of the training sample. With regard to electricity spot price forecasting, most of the studies use a training sample of approximately 2 months [2], [7]. What is more, the electricity market in Europe is facing an unprecedented sharp increase in wholesale power prices since June 2021. As a result, the training sample was chosen to range from 2021-06-01 to 2021-10-09 and it was applied to all the investigated machine learning algorithms in order to secure consistency. As mentioned above, most of the academic work on electricity price forecasting could be classified into two categories, autoregressive models and neural networks. Consequently, five models were built and tested one SARIMAX, one simple MLP, one Vanilla LSTM, one Stacked LSTM and one Bidirectional LSTM. In the SARIMAX model features were inserted as exogenous variables, apart from the SMP-24 which was excluded. All of the five models were evaluated on the basis of MAPE and RMSE metrices with the view to concluding for the best method. Five different forecasts were produced concerning hourly SMP data extending up to two weeks ahead. This period was decided so that to use a valid enough sample to test these forecasts. LSTM models and in particular the Stacked LSTM model presented the most promising results with MAPE and RMSE values for the two weeks ahead forecast of 6.965 aper cent and 19.628 Euro / MWh correspondingly. In addition, LSTM models (Stacked and Bidirectional) presented an improved performance as more data were fed for training. Overall, it could be suggested that LSTM networks combined with the proper attributes, like the selected ones in this research or other variations of them, may be the way forward regarding power spot price predictions.

However, there are certain limitations in this approach. HU SMP and Residual values used to train the different models were actual, something that will not be the case for future use. That is, HU SMP and Residual values must be forecasted, hence implying further uncertainty. The sensitivity analysis carried out showed that the forecasting performance drops, fortunately to acceptable levels, leading to the need for as much accurate as possible predictions pertaining to HU SMP and Residual. Furthermore, the applied dataset contained merely 3456 records and nonetheless LSTM networks required enough time to train and fit the data accordingly. In case of 20000 or 30000 records or even more, the use of LSTM networks won't be feasible without parallel processing.

8 Conclusions

This MSc thesis engaged with the creation of different machine learning models with a view to forecasting the Italian Day-Ahead SMP and evaluating of the respective results. Two main categories were investigated, autoregression and neural networks. Namely, five approaches were tested one SARIMAX, one MLP, one Vanilla LSTM, one Stacked LSTM and one Bidirectional LSTM. The problem could be classified as a multivariate time series problem, where from a number of attributes only three were selected (HU SMP, Residual and SMP-24). Only a small number of studies have been carried out on the Italian SMP forecasting, rendering this study relevant. What is more, among the attributes chosen to train the investigated machine models, HU SMP is included representing the wholesale electricity price of a major energy hub in Europe. The results of this research seem to be promising filling to some extend the current research gap. That is, new robust forecast methods are proposed concerning a very significant energy market, such as Italy, in the southern area of Europe while taking into account the even higher interdependence among European countries due to coupling. It may be concluded that all trained models are suitable for use without any update for at least two weeks. Stacked and Bidirectional LSTMs networks produced the most prominent results, entailing the implementation of recurrent neural networks for such problems preferable. The most significant forecast errors occurred on 2021/10/20 and on 2021/10/21, for a couple of hours lying in the peak periods, which could be attributed to unexpected events.

However, there is a number of recommendations that could further improve the performance of the proposed LSTM model. For the purposes of this study only one timestep was used to split the data into samples in order to train the LSTM model. Thus, the performance of the LSTM networks could be tested for more than one timestep. Stacked LSTM showed the best results, nevertheless other approaches could be more effective. For instance, A convolutional neural network (CNN) can be used in a hybrid CNN-LSTM model with an LSTM backend, where the CNN is used to interpret sub sequences of input data which combined are fed as a sequence to the LSTM model to interpret. Another example would be the ConvLSTM, where the convolutional reading of input is built directly into each LSTM unit [21]. With regard to HU SMP attribute, actual prices were used to train the different models, albeit predictions will be need for future use. For this reason, the development of similar to this study machine learning algorithms pertaining to HU SMP forecasting is highly recommended. Likewise, for the Residual attribute new accurate forecasting models should be developed or access to relevant forecasts released by the respective TSOs must be granted. Finally, different variations of the attributes selected might improve model's performance. Specifically, in this approach only the absolute values of the attributes were used to train the investigated machine learning models, whereas the corresponding daily changes or the corresponding moving averages with certain time windows could be applied instead.

9 References

[1] F. Azevedo and Z. A. Vale, "Forecasting Electricity Prices with Historical Statistical Information using Neural Networks and Clustering Techniques," IEEE, Porto, 2006.

[2] P. S. Georgilakis, "ARTIFICIAL INTELLIGENCE SOLUTION TO ELECTRICITY PRICE FORECASTING PROBLEM," *Applied Artificial Intelligence ,* vol. 21, no. 8, pp. 707-727, 2007.

[3] C. STAFF, "EUR-Lex.europa.eu," [Online]. Available: https://eur-lex.europa.eu/legal-content/bg/TXT/?uri=CELEX:52017SC0383. [Accessed 10 October 2021].

[4] GME, "mercatoelettrico.org," [Online]. Available: https://www.mercatoelettrico.org/En/Mercati/MercatoElettrico/MPE.aspx. [Accessed 11 October 2021].

[5] R. Gareta, L. M. Romeo and A. Gil, "Forecasting of electricity prices with neural networks," CIRCE, Zaragoza, 2005.

[6] I. P. Panapakidis and M. N. Moschakis, "Comparison of Machine Learning Models for the Prediction of System Marginal Price of Greek Energy Market," *International Journal of Energy and Environmental Engineering,* vol. 13, no. 3, pp. 148-152, 2019.

[7] N. Amjady and F. Keynia, "Day-Ahead Price Forecasting of Electricity Markets by Mutual Information Technique and Cascaded Neuro-Evolutionary Algorithm," *IEEE TRANSACTIONS ON POWER SYSTEMS,* vol. 24, no. 1, pp. 306-318, 2009.

[8] J. P. S. Catalão, S. J. P. S. Mariano, V. M. F. Mendes and L. A. F. M. Ferreira, "An Artificial Neural Network Approach for Short-Term Electricity Prices Forecasting," ISAP, Kaohsiung, 2007.

[9] M. Adam and B. Tadeusz, "Forecasting day-ahead spot electricity prices using deep neural networks with attention mechanism," *Smart Environments and Green Computing,* vol. 1, no. 1, pp. 21-31, 2021.

[10] S. Simon and W. Andreas, "Electricity Price Forecasting with Neural Networks on EPEX Order Books," *Applied Mathematical Finance,* vol. 27, no. 3, pp. 189-206, 2020.

[11] S. Anbazhagan and N. Kumarappan, "Day-Ahead Deregulated Electricity Market Price Forecasting Using Recurrent Neural Network," *IEEE SYSTEMS JOURNAL,* vol. 7, no. 4, pp. 866-872, 2013.

[12] A. Gianfreda and L. Grossi, "Forecasting Italian Electricity Zonal Prices with Exogenous Variables," University of Verona, Verona, 2011.

[13] Z. Tan, J. Zhang, J. Wangb and J. Xu, "Day-ahead electricity price forecasting using wavelet transform combined with ARIMA and GARCH models," *Applied Energy,* vol. 87, no. 11, pp. 3606-3610, 2010.

[14] J. C. Cuaresma, J. Hlouskovab, S. Kossmeierc and M. Obersteinerd, "Forecasting electricity spot-prices using linear univariate time-series models," *Applied Energy,* vol. 77, pp. 87-106, 2004.

[15] A. Gianfreda, F. Ravazzolo and L. Rossini, "Comparing the forecasting performances of linear models for electricity prices with high RES penetration," *International Journal of Forecasting,* vol. 36, pp. 974-986, 2020.

[16] S. Fan, J. R. Liao, K. Kaneko and L. Chen, "An Integrated Machine Learning Model for Day-Ahead Electricity Price Forecasting," IEEE, Osaka, 2006.

[17] M. G. Flammini, G. Prettico, A. Mazza and G. Chicco, "Reducing fossil fuel-based generation: Impact on wholesale electricity market prices in the North-Italy bidding zone," *Electric Power Systems Research,* vol. 194, no. 0378-7796, p. 107095, 2021.

[18] C. Albon, Machine Learning with Python Cookbook - Practical Solutions from Preprocessing to Deep Learning, Boston: O'Reilly Media, 2018.

[19] F. CHOLLET, Deep Learning with Python, Manning Publications Co., 2018.

[20] N. Shukla, Machine Learning With TensorFlow, Manning Publications Co., 2017.

[21] J. Brownlee, Long Short-Term Memory Networks with Python, Jason Brownlee, 2020.

[22] J. Brownlee, Introduction to Time Series Forecasting with Python, Jason Brownlee, 2020.

[23] Entso-e, "ENTSO-E Transparency Platform," [Online]. Available: https://transparency.entsoe.eu/dashboard/show. [Accessed 23 October 2021].

[24] Terna, "Terna Driving Energy," [Online]. Available: https://www.terna.it/en/electric-system/transparency-report. [Accessed 25 November 2021].

[25] N. Pool, "NORD POOL," [Online]. Available: https://www.nordpoolgroup.com/Market-data1/Dayahead/Area-Prices/fr/hourly/?view=table. [Accessed 23 October 2021].

[26] I. H.Witten and E. Frank, DATA MINING Practical Machine Learning Tools and Techniques, San Francisco: Elsevier, 2005.

[27] Platts, "GENI Global Energy Network Institute," POWERmap Europe, [Online]. Available: http://www.geni.org/globalenergy/library/national_energy_grid/italy/italiannationalelectricitygrid.shtml. [Accessed 22 March 2022].