
Bergische Universität Wuppertal

Fakultät für Mathematik und Naturwissenschaften

Institute of Mathematical Modelling, Analysis and Computational Mathematics (IMACM)

Preprint BUW-IMACM 22/22

Aurora Poggi, Luca Di Persio and Matthias Ehrhardt

# Electricity Price Forecasting via Statistical and Deep Learning Approaches: the German case

December 7, 2022

http://www.imacm.uni-wuppertal.de

MDPI

*Article*

# Electricity Price Forecasting via Statistical and Deep Learning Approaches: the German case

**Aurora Poggi [1,*,†], Luca Di Persio [2,†] and Matthias Ehrhardt [3,†]**

[1] College of Mathematics, Department of Computer Science, University of Verona, Verona, Italy; aurora.poggi@studenti.univr.it
[2] College of Mathematics, Department of Computer Science, University of Verona, Verona, Italy; luca.dipersio@univr.it
[3] Chair of Applied and Computational Mathematics, University of Wuppertal, Wuppertal, Germany; ehrhardt@uni-wuppertal.de
* Correspondence: aurora.poggi@studenti.univr.it
‡ These authors contributed equally to this work.

Academic Editor: Chaman Verma, Maria Simona Raboaca, Zoltán Illés
Version December 7, 2022 submitted to Journal Not Specified

**Abstract:** We study state-of-the-art models for Electricity Price Forecasting ranging from standard inferential statistical methods to deep learning based ones. Combining multiple weekday dummies with historical data, we propose an innovative forecast solution where electricity spot prices series are decomposed into a seasonal trend component plus a stochastic one. The latter allows us to provide a highly performing predictive solution in all considered time windows.

**Keywords:** Electricity price forecasting; univariate model; statistical method; autoregressive; machine learning; deep learning; neural network

## 1. Introduction

The study of *Electricity Price Forecasting (EPF)* has attracted an increasing attention within Europe energy markets at least starting from 1996, as result of liberalization protocol adopted by the European Authority for Energy. The latter concertized in an augmented complexity of the European electricity market, as a whole, as well as w.r.t. each single production/consumption of its components.

The most studied aspect of electricity price forecasting concerns short time horizons, i.e. from hours to a day, underlying dynamics being different from those of other commodities. Indeed, electricity can not be stored and a constant balance between generation and consumption is required. Demand, in turn, is subject to hourly, daily and seasonal fluctuations, being also influenced by economic activities of participating countries, political changes, weather variables' behavior (temperature, solar radiation, wind speed/direction, etc.), interconnected markets' dynamics.

Focusing on the German market, from 2017 renewable energy sources start acquiring more and more relevance, with an increment of 38.5 %, leading to an adjustment in short-term electricity trading also because of the implicit volatility due to weather conditions variance. The latter implied the need to develop models able to simultaneously consider point forecasts, probabilistic forecasts or path forecasts. We focus on the study of electricity prices first applying standard statistical models, to then switch to machine learning based solutions. Predictions of both daily (average) and hourly prices are carried out w.r.t. different time windows to be predicted: in the first case medium and long time horizons, medium and short ones for the second one.

The rest of the paper is organized as follows. Section 2 and Section 3 present the statistical models for the prediction and the machine and deep learning models. Section 4 contains the analysis of our data and shows the results obtained. Finally, Section 5 discusses the results and possible future work.

## 2. Benchmark Data and Statistical Models

We focus on forecasting electricity prices behavior within the German market w.r.t. to both large time window, months, and short time ones, day. We implement statistical as well as deep learning models, exploiting German market hourly prices from 2020 and until mid-2022. For the sake of simplicity, we have assumed that the year 2020 is a non leap year, in order to have all the years with 365 days. It is worth mentioning that our datasets are characterized by not so regular behavior. Indeed, we had an increasing trend, starting in July 2021 then culminating in March 2022, probably incorporating effects due to the Russia-Ukraine conflict. Moreover, we underline how years analyzed have seen the huge impact of the COVID-19 pandemic as well as energy crisis that, in Germany, implied supply contracts with delivery in 2023 to be over 1000 € per Mw/h and to 800 € per Mw/h, in August 2022. Finally, 2022 saw significant changes since German renewable electricity covering 49 % of demand during first six months. In what follows, we denote the price $P$ at time $t$, by $P_t$, then discretizing the period of interest, let's say $[0, T]$, with $T$ positive, but finite, in regular $N$, $N \in \mathbb{N}^+$ and finite, sub-intervals with equally spaced extremes $t_i$, $i = 1, \dots, N$, i.e. observations are collected at fixed time intervals: hourly intervals. We used the most recent prices along considered time windows, dividing the time series into segments with the 'same' price level, and exploiting algorithms such as K-Nearest Neighbourhood (K-NN) [1] or more recent Narrowest Over Threshold (NOT) [2], to select the calibration sample based on similarities with respect to a subset of explanatory variables, leading to a remarkable improvement in forecasting.

As naive benchmark we used a method belonging to the class of similar-day technique. The idea is the following: the electricity price forecast for hour $h$ on Tuesday, Wednesday, Thursday and Friday is set equal to the price for the same hour on the previous day, i.e. $\hat{P}_{d,h} = P_{d-1,h}$, while the forecast for hour $h$ on Saturday, Sunday and Monday is set equal to the price for the same hour a week ago, i.e. $\hat{P}_{d,h} = P_{d-7,h}$.

### 2.1. Statistical Models

Among statistical models usually exploited for time series prediction, let us recall the Autoregressive Moving Average (ARMA) one along with its extensions, and Generalized Autoregressive Conditional Heteroskedasticy (GARCH) [3]. For the sake of completeness, let us recall the definitions of Autoregressive (AR) and Moving Average (MA), to then analyze Autoregressive Moving Average and its extensions.

An *Autoregressive Model*, of order $p$ is indicated as AR$(p)$, predicts a variable of interest using a linear combination of past values of that variable. This model takes into account the random nature and time correlations of the phenomenon studied, starting with previous prices. The AR model reads

$$P_t = \alpha + \phi_1 P_{t-1} + \phi_2 P_{t-2} + \cdots + \phi_p P_{t-p} + \epsilon_t, \tag{1}$$

where $\epsilon_t$ takes into account the randomness component, being modeled by a white noise stochastic process.

A *Moving Average Model* predicts a variable using a linear combination of past forecast errors: this model considers $q$ previous values of the noise, thus performing a totally different procedure compared to the AR models. We denote by MA$(q)$ a Moving Average model of order $q$:

$$P_t = \alpha + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}, \tag{2}$$

$\epsilon_t$ being defined as above.

The *Autoregressive Moving Average Model* is denoted by ARMA$(p, q)$, where $p$ and $q$ are the coefficients of AR and MA, respectively. In the ARMA$(p, q)$ model, the price $P_t$ is defined as

$$\phi(B) P_t = \theta(B)\, \epsilon_t, \tag{3}$$

where $B$ is the backward shift operator, i.e. $BP_t = P_{t-1}$. In detail, the $\phi(B)$ and $\theta(B)$ terms are:

$$\phi(B) = 1 - \phi_1 B - \ldots - \phi_p B^p \quad \text{and} \quad \theta(B) = 1 + \theta_1 B + \ldots + \theta_q B^q,$$

where $\phi_1, \ldots, \phi_p$ and $\theta_1, \ldots, \theta_q$ represent the coefficients of the AR and MA polynomials, while $\epsilon_t$ can be seen as a collection of i.i.d. Gaussian random variables, i.e. $\epsilon_t \sim WN(0, \sigma^2)$, hence specifying the white noise component cited before at each time of interest.

Since the ARMA model can be applied only to stationary time series, we applied unit root tests (ADF, KPSS and PP), cf. [4].

Previous point can be overcame by *Autoregressive Integrated Moving Average Model*, ARIMA, introduced by Box and Jenkins (1976), to consider non-stationary time series by exploiting a *differencing technique*. The latter allows to remove both trend and seasonality, to obtain a stationary time series, from data with a period $d$. Firstly, we have to introduce the $lag - d$ differencing operator $\nabla_d$ defined as

$$\nabla_d P_t = P_t - P_{t-d} = (1 - B^d)\, P_t. \tag{4}$$

The ARIMA$(p, d, q)$ model can be written as:

$$\phi(B)\nabla^d P_t = \theta(B)\epsilon_t, \tag{5}$$

where $p$ and $q$ are the order for the AR and MA models, respectively. Here $d$ indicates the number of differencing passes at lag $d$.

When time series is also characterized by *seasonality*, a standard approach is to used the *Seasonal Autoregressive Integrated Moving Average Model*, SARIMA, an extension of ARIMA. The SARIMA$(p, d, q) \times (P, D, Q)_s$ model is defined as

$$\phi(B)\Phi(B^s)\nabla^d \nabla_s^D P_t = \theta(B)\Theta(B^s)\epsilon_t, \tag{6}$$

where $(p, d, q)$ refers to the non-seasonal component, while $(P, D, Q)$ refers to the seasonal ones, and $s$ indicates the number of observations in a season. Since every SARIMA model can be transformed into an ARMA model using the variable $\tilde{P}_t = \nabla^d \nabla_s^D P_t$, prediction is accomplished in two steps: model identification and estimation of the parameters, see Section 4.3.2.

Next, we introduce an Autoregressive Model for hourly price prediction where dummies are considered. The model uses the ARX model as starting point, but does not include exogenous variables:

$$P_{d,h} = \beta_1 P_{d-1,h} + \beta_2 P_{d-2,h} + \beta_3 P_{d-7,h} + \beta_4 P_{d-1,24} + \beta_5 P_{d-1}^{\max} + \beta_6 P_{d-1}^{\min} + \sum_{j=1}^{7} \beta_{h,j+8} D_j + \epsilon_{d,h}, \tag{7}$$

where $P_{d-1,h}$, $P_{d-2,h}$ and $P_{d-7,h}$ account for the autoregressive effects corresponding to prices from the same hour $h$ of the previous day, two days before and a week before. The coefficient $P_{d-1,24}$ is the last known price at the time when the prediction is made, then providing information about the end of day price level. Coefficients $P_{d-1}^{\min}$ and $P_{d-1}^{\max}$ are the previous day's minimum and maximum prices, respectively. Lastly, $D_1, \ldots, D_7$ are weekday dummies and $\epsilon_{d,h}$ is the noise term, which is assumed to be i.i.d. and with finite variance, then we estimate the $\beta_i$'s using *Least Absolute Shrinkage and Selection Operator*.

## 3. Machine Learning based Models

For the sake of completeness, in what follows we recall basics about Extreme Gradient Boosting and Neural Network models.

### 3.1. The XGBoost Model

The *Extreme Gradient Boosting* (XGBoost), improves gradient booster performances by considering new trees correcting errors of those trees that are already part of the model. Trees are added until no further improvements can be made to the model, hence implementing a *walk forward validation* [5] scheme. In particular, we used the XGBoost library [6].

Given a training set $\{(x_t, p_i)\}_{i=1}^{N}$, a differentiable loss function $L(p, F(x))$, a number of weak learners $M$ and a learning rate $\alpha$ [7], the algorithm is defined as follow:

1. Initialization of the model with a constant value:

$$\hat{f}_{(0)}(x) = arg\min_{\theta} \sum_{i=1}^{N} L(p_i, \theta).$$

2. For $m = 1, \dots M$:

   (a) Compute the gradients and Hessians:

$$\hat{g}_m(x_i) = \left[\frac{\partial L(p_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}_{(m-1)}(x)}, \qquad \hat{h}_m(x_i) = \left[\frac{\partial^2 L(p_i, f(x_i))}{\partial f(x_i)^2}\right]_{f(x)=\hat{f}_{(m-1)}(x)}.$$

   (b) Fit a base learner using the training dataset $\left\{x_i, -\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}\right\}_{i=1}^{N}$ by solving the optimization problem below:

$$\hat{\phi}_m = \arg\min_{\phi \in \Phi} \sum_{i=1}^{N} \frac{1}{2}\hat{h}_m(x_i)\left[-\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)} - \phi(x_i)\right]^2, \qquad \hat{f}_m(x) = \alpha\hat{\phi}_m(x).$$

   (c) Update the model $\hat{f}_{(m)}(x) = \hat{f}_{(m-1)}(x) + \hat{f}_m(x)$.

3. Output: $\hat{f}(x) = \hat{f}_{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x)$.

### 3.2. Neural Network Models

Since the mid-2010s, research on EPF shifted to consider an increasing number of inputs as, e.g., in Deep Learning models. The latter has been made possible by augmented computational capacities (mainly based on GPUs) at lower costs, also exploiting on cloud solutions. As a result, it has been possible to gain better representations of hidden data, while maintaining reasonable work-times. Neural Networks can be equipped to provide from single-valued forecast, to a complete interval of possible values.

The first Neural Networks used for Electricity Price Forecasting were mainly simple NNs with one hidden layer such as Multilayer Perceptron (MLP), Radial Basis Function (RBF) networks or at most very simple Recurrent Neural Networks. The most common MLP is described as follows: every neuron in the previous layer is fully connected to every neuron in the next layer. In the EPF literature, the input is the past energy load and the output is the future energy load.

The *Deep Neural Network* (DNN) is the natural extension of the traditional MLP using multiple hidden layers. Here, our DNN is a Deep Feedforward Neural Network [8] with 4 layers within the multivariate framework and exploiting Adam optimizer. The variables defining a DNN with two hidden layers are the following: the input vector $\mathbf{X} = [x_1, \dots, x_n]^\top$, the vector of day-ahead prices that we want to predict $\mathbf{P} = [p_1, \dots, p_{24}]^\top$ and the number of neurons for hidden layer $n_1$, $n_2$.

*Recurrent Neural Network* (RNN) are a specialized class of Neural Networks allowing cyclical connections. Their structure allows to record past information that can be then used to forecast future values. It is worth mentioning that RNNs only take into account inputs at time $t-1$ hence facing

the *long-term dependencies* problem, that can be tackled by Long Short Term Memorys, introduced by Hochreiter and Schmidhuber in 1997 [9].

*Long Short Term Memory* (LSTM) Neural Networks are special variants of RNNs, in which information can be stored, updated or forgot by a choice of the state of the cell. This allows such NNs to use large input spaces and understand long-term dependencies. The computational graph of LSTMs contains five basic elements: input gate, forget gate, output gate cell and state output. Gate operations are performed in the cell memory state and are of the following types: reading, writing and erasing.

Defining as $x_t$ the input value recorded at time $t$ and by $h_t$ the associated LSTM output, main steps of this particular NN-solution read as follows:

- decide which information is going to be removed from the cell state by using the sigmoid layer, called the *forget gate layer*. It looks at $x_t$ and $h_{t-1}$ and returns a value between 0 and 1 for each value in the cell state $C_{t-1}$.
- decide which new information will be stored in the cell state. This step is divided into two substeps:
  1. use a *input gate layer* implemented by a sigmoid layer deciding which values will be updated, $i_t = \sigma(W_i \times [h_{t-1}, x_t] + b_i)$;
  2. use a tanh layer to provide a vector of new candidate values $\tilde{C}_t = \tanh(W_C \times [h_{t-1,x_t}] + b_C)$ that can be added to the state.

  As a result, old cell state $C_{t-1}$ is updated with $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$, where $f_t \cdot C_{t-1}$ indicates what we have already chosen to forget and the other term indicates the new candidates.
- The output will be defined by a filtered version of the cell state, via a sigmoid layer deciding which parts will be included in the output $o_t = \sigma(W_o \cdot [h_{t-1,x_t}] + b_o)$, and then a tanh is carried out which is itself multiplied by the output of the sigmoid gate $h_t = o_t \cdot \tanh(C_t)$.

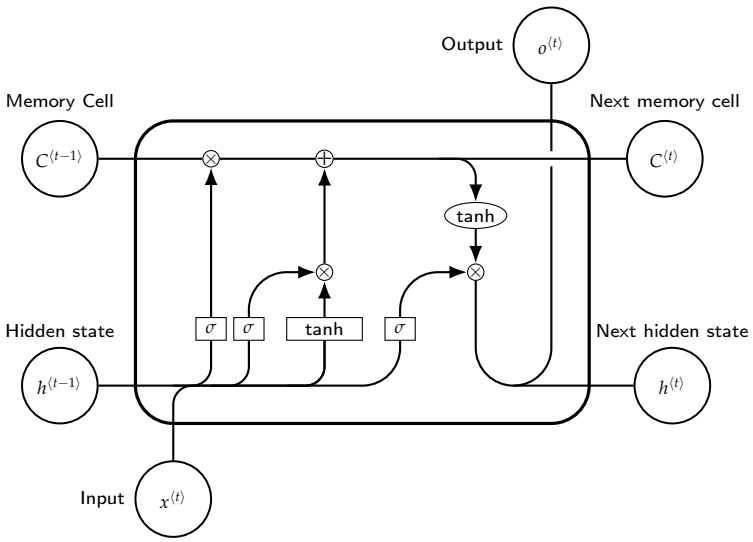


**Figure 1.** Block of Long Short Term Memory at any timestamp $t$.

Another NN that can be used to predict day-ahead prices is the *Convolutional Neural Network* (CNN), which uses the concept of weight sharing and provides better accuracy in highly non-linear problems. The inputs are divided between those modeling sequential past data $\mathbf{X_S} = [x_{S_1}, \ldots, x_{S_N}]^\top$ and those modeling information about next 24 hours day-ahead $\mathbf{X_F} = [x_{F_1}, \ldots, x_{F_N}]^\top$. With previous two inputs, the model uses two parallel CNNs to model the electricity price behavior. Specifically, the convolution process starts with inputs and transforms them into the feature maps. Then the pooling process is performed, wherein the feature map of convolution layer is sampled and its dimension is reduced. After both networks perform a series of convolution and pooling operations. Then, we have a fully connected layer that models the day-ahead prices $\mathbf{P}$.

## 4. Data Analysis

We implemented the above mentioned models considering hourly electricity prices in Germany, ranging from 2020 to mid-2022, then providing a forecast exploiting both daily average and hourly prices on different time windows. Concerning daily average prices, we first carried out a long term forecast taking holidays. The latter could be useful in view of investments planning them, e.g., when dealing with mid to long-term power plant energy demand programming, see, e.g. [10].

The accuracy of a forecast model is defined considering realized errors, e.g., analyzing Mean Absolute Percentage Error, Mean Absolute Deviation and Median Relative Absolute Error. For the day-ahead forecasting those errors are not a good choice, since the economic decision might lead to economic benefits or damage, depending on the forecast precision. Rather, according to literature, a better choice is represented bny quadratic errors, i.e. $L_2$, such as the *Mean Squared Error* (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{t=1}^{n}(\hat{P}_t - P_t)^2$$

or the related *Root Mean Squared Error* (RMSE), defined as: $\text{RMSE} = \sqrt{\text{MSE}}$.

### *4.1. Naive Benchmark Results*

We first show results obtained from the naive benchmark, cf. Section 2.

Daily Average Price:

In the analysis of average daily prices, hours were not taken into account, i.e. the forecast is equivalent to the price of the previous day except for Saturday, Sunday and Monday, considered to be those of the previous week. These results, in Table 1, show that this model does not accurately identify future prices, in fact the errors are considerable.

| **Naive Benchmark** | | |
|---|---|---|
| **Training period** | **Test period** | **RMSE** |
| 2020 | 2021 | 91.0102 |
| 2020-2021 | 2022 | 150.4029 |
| two weeks in 2020 | two weeks in 2021 | 56.0675 |

**Table 1.** RMSE of Naive Benchmark on different training and test periods for daily average prices.

Hourly Daily Price:

We show the errors of the Naive Benchmark concerning the prediction of daily hourly prices over different time windows. From Table 2, showing the RMSEs, we deduce that, as in the previous case concerning daily average prices, this model does not perform well on our data. The predictions of daily hourly prices obtained by the Naive Benchmark model are distant in terms of values to the current price, but as we show in Figure 2, the behavior is predicted very accurately, which is due to the fact that the year 2020 and 2021 during the first months of the year had similar daily behavior.

| **Naive Benchmark** | | |
|---|---|---|
| **Training Period** | **Test Period** | **RMSE** |
| one month in 2020 | one month in 2021 | 123.8230 |
| one week in 2020 | one week in 2021 | 51.5851 |
| one day in 2020 | one day in 2021 | 31.6256 |

**Table 2.** RMSE of Naive Benchmark model on different training and test periods for daily hourly prices.

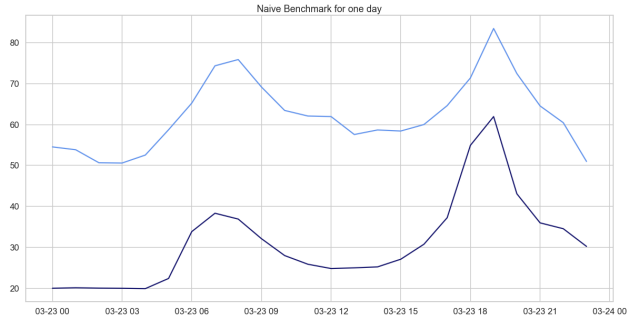


**Figure 2.** Comparison of target and forecast values for one day with Naive Benchmark.

### *4.2. SARIMA*

In what follows we consider SARIMA model, after observing seasonality of our time series. We first take into account Autocorrelation Function (ACF) and the Partial Autocorrelation Function (PACF), as to focus on parameter $q$ and $p$, respectively. While to identify $m$ it is sufficient to analyze the time series provided. It is crucial to note that the choice of $m$ affects the seasonal parameters $P, D, Q$. Parameters $d$ and $D$ values can be determined by unit root test, in our study case the ADF, or by looking at the rolling mean and the rolling standard deviation.

Daily Average Price:

In the context of daily average prices of electricity the seasonality is defined as $m = 7$, since we observed a periodical weekly behaviour. The parameters of the model SARIMA, for the forecasts of the daily average prices, have been chosen observing ACF and PACF plots and using the library *pmdarima*.

We rely on this results, as we checked at the $p$-value of each individual parameter which are less than 0.05, i.e. they are less statistically significant. Then we checked at the $p$-value of the Ljung-Box test and we cannot reject the null hypothesis: in conclusion, the residuals are independently distributed, i.e. they are white noise.

| $\mathbf{SARIMA}(3,1,3) \times (1,1,1)_7$ | |
|---|---|
| **Training Period** | **RMSE** |
| 2020 | 85.3509 |

**Table 3.** RMSE of SARIMA model of 2020 for testing 2021 for the daily average prices.

| $\mathbf{SARIMA}(2,0,2) \times (0,1,1)_7$ | |
|---|---|
| **Training Period** | **RMSE** |
| 2020-2021 | 77.6600 |

**Table 4.** RMSE of SARIMA model of 2020-2021 for testing mid-2022 for the daily average prices.

In Table 4 we observe errors obtained from the SARIMA model trained on data from 2020 to 2021 to predict mid-2022 which are much lower than the ones for the model previously observed. Also we can consider the results obtained because the $p$-value of the Ljung-Box test is 0.92 which is much greater than 0.05, the residuals are independent.

Although we can consider such results as they satisfying the model's assumptions and although they fit better than previous model, we observe in Figure 3 that SARIMA predicts regular and periodic behavior which is not reflected in our data. Is also interesting to observe a significant peak in March, an unexpected datum probably due to the war between Russia and Ukraine that started one week before. As seen for the other SARIMA models, Table 5 provides errors obtained using different time windows as a training period and a two weeks test set.

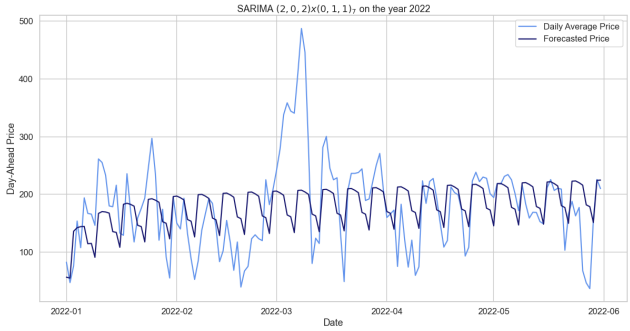


**Figure 3.** Comparison of target and forecast values for mid-2022 using 2020-2021 as training set with SARIMA$(2,0,2) \times (0,1,1)_7$ for daily average prices.

| RMSE of two weeks of testing in 2021 | | |
|---|---|---|
| **Training Period** | **SARIMA$(p,d,q) \times (P,D,Q)_7$** | **RMSE** |
| 2020 | SARIMA$(1,1,2) \times (1,0,2)_7$ | 23.3416 |
| Spring-Summer | SARIMA$(2,1,1) \times (0,1,1)_7$ | 17.1818 |
| Autumn-Winter | SARIMA$(1,1,3) \times (4,1,3)_7$ | 12.2135 |

**Table 5.** RMSE of the SARIMA model trained on different time windows for testing two weeks.

In Table 6 we show the $p$-values for the Ljung-Box test, which allows us to state that the three models are reliable. In Figure 4 we observe the forecast made by the SARIMA model for a fortnight, the model predicts a more regular behavior than the current one and fails to predict peaks.

| Ljung-Box Test | | |
|---|---|---|
| **Training Period** | **$p$-value** | **Null Hypothesis** |
| 2020 | 0.93 | not rejected |
| Spring-Summer | 0.87 | not rejected |
| Autumn-Winter | 0.69 | not rejected |

**Table 6.** The $p$-values of the Ljung-Box test for the SARIMA model on 2 weeks of test.

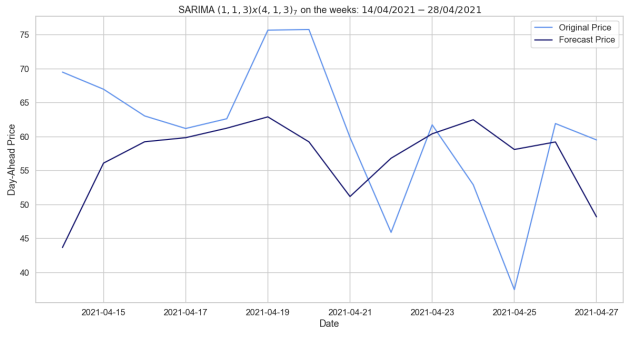


**Figure 4.** Comparison of target and forecast values for two weeks in 2021 using autumn-winter 2020-2021 as training set with SARIMA$(1,1,3) \times (4,1,3)_7$ on daily average prices.

Hourly Daily Price:

We considered daily hourly prices, starting by predicting a month using different time windows and then predicting a week and a day using the same time windows. The SARIMA models explained above were tested performing the Ljung-Box test as shown in Table 8. In Table 9 we see the results

obtained from SARIMA models to predict a week. In the case of the autumn-winter training set it turns out to be a good prediction, in fact the lowest RMSE obtained in these SARIMAs.

| **RMSE of one month of testing in 2021** | | |
|---|---|---|
| **Training Period** | **SARIMA**$(p,d,q) \times (P,D,Q)_{24}$ | **RMSE** |
| 2020 | SARIMA$(2,1,1) \times (2,1,0)_{24}$ | 118.8870 |
| Spring-Summer | SARIMA$(2,0,2) \times (2,1,0)_{24}$ | 107.9992 |
| Autumn-Winter | SARIMA$(2,0,0) \times (2,1,0)_{24}$ | 110.7636 |

**Table 7.** RMSE of the SARIMA model of different time windows for testing one month in 2021.

| **Ljung-Box Test** | | |
|---|---|---|
| **Training Period** | $p$**-value** | **Null Hypothesis** |
| 2020 | 0.33 | not rejected |
| Spring-Summer | 0.92 | not rejected |
| Autumn-Winter | 0.96 | not rejected |

**Table 8.** $p$-values of the Ljung-Box test for the SARIMA model on one month test.

| **RMSE of one week of testing in 2021** | | |
|---|---|---|
| **Training Period** | **SARIMA**$(\mathbf{p},\mathbf{d},\mathbf{q}) \times (\mathbf{P},\mathbf{D},\mathbf{Q})_{24}$ | **RMSE** |
| 2020 | $SARIMA(2,1,0) \times (2,1,0)_{24}$ | 41.7355 |
| Spring-Summer | $SARIMA(2,0,2) \times (2,1,0)_{24}$ | 19.6915 |
| Autumn-Winter | $SARIMA(2,0,0) \times (2,1,0)_{24}$ | 16.5124 |

**Table 9.** RMSE of the SARIMA model of different time windows for testing one week.

Figure 5 presents the forecasts made provided by the SARIMA model, using autumn-winter as training set and testing one week in 2021, which predicts positive peaks quite well while is not detecting prices approaching zero. Table 10 shows the errors obtained by SARIMA models on one day using different time series as training sets. Also for these models, $p$-values were measured for the Ljung-Box test, all of which allows us to state the models considered are defined by independent residuals.

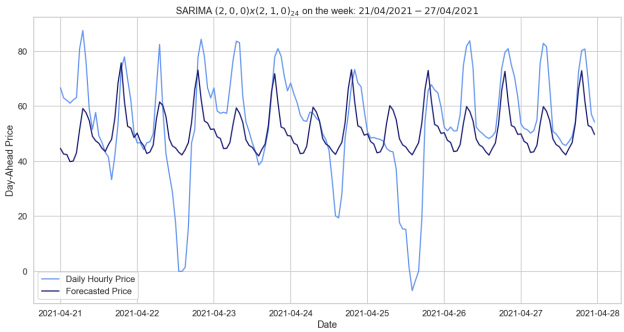


**Figure 5.** Comparison of target and forecast values for one week in 2021 using autumn-winter as training set with SARIMA$(2,0,0) \times (2,1,0)_{24}$.

| **RMSE of one day of testing in 2021** | | |
|---|---|---|
| **Training Period** | **SARIMA**$(\mathbf{p},\mathbf{d},\mathbf{q}) \times (\mathbf{P},\mathbf{D},\mathbf{Q}_{24}$ | **RMSE** |
| 2020 | $SARIMA(2,1,0) \times (2,1,0)_{24}$ | 7.8986 |
| Spring-Summer | $SARIMA(2,0,2) \times (2,1,0)_{24}$ | 16.9000 |
| Autumn-Winter | $SARIMA(2,0,0) \times (2,1,0)_{24}$ | 12.1786 |

**Table 10.** RMSE of SARIMA model of different time windows for testing one day for the hourly prices.

Figure 6 displays the 24-hour period forecast by the model previously stated. The above model does not predict very well since it predicts only one peak, although our time series of daily hourly prices are defined by two peaks. We can state that the SARIMA model better performs on both daily average and hourly prices, when compared with the XGBoost and Naive Benchmark. However it is not really appropriate as evidenced by the RMSEs.

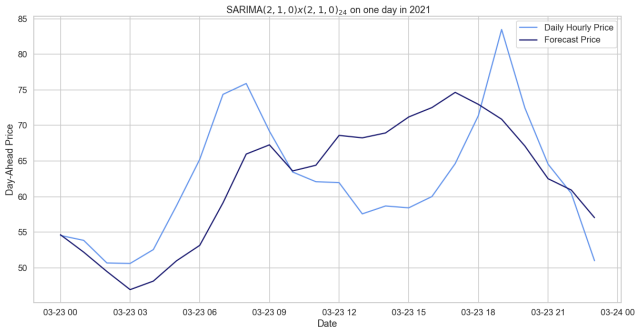


**Figure 6.** Comparison of target and forecast values for one day in 2021 using 2020 as training set with SARIMA$(2,1,0) \times (2,1,0)_{24}$.

### *4.3. Deseasonalization*

In this section we are going to deseasonalize our time series, applying the wavelet decomposition. Using the time series with a seasonal adjustment clearly improves the accuracy obtained on both simple autoregression and structured models with automated variable selection via LASSO. Moreover, such seasonal decomposition has also turned out to work well for deep learning based models for both point forecast and probabilistic forecast.

Seasonal decomposition refers to the representation of a signal as sum and/or product of a periodic component, the remaining variability being typically described by the action of a stochastic process tha could allow for jumps. We referred to the *wavelets approach*. Because after numerical implementations, the RMSEs obtained from forecasts made with the same models on time series with a seasonal adjustment done with the HP filter are similar.

#### 4.3.1. Wavelet Decomposition

Wavelet transform is based on a series of functions called wavelets, each with a different scale. A wavelet family consists of pairs composed by a father, also called scaling function, and a mother, respectively indicated with $\phi$ and $\psi$. In detail, the father wavelet is about the low frequency smooth components while the mother captures the higher frequency components. Every wavelet family is defined with an order, indicating the number of vanishing moments which is related to the approximation order and smoothness of the wavelet.

Additionally, there are two assumptions on the wavelet, finite energy and zero mean. Finite energy means that it is localized in time and frequency; it is integrable and the inner product between the wavelet and the signal always exists. The admissibility condition implies a wavelet to have zero mean in the time domain, a zero at zero frequency in the time domain. This is necessary to ensure that it is integrable and the inverse of the wavelet transform can also be calculated.

We used the Daubechies family of order 24 (denoted by 'd24'), as suggested by Weron [11] in the *'deseasonalize.m'* MATLAB code, and smoothing level $k$ from 6 to 14, as suggested in [12]. The Daubechies wavelet family of order 24 is sufficiently regular and smooth for our datasets. To perform the wavelet decomposition in Python, the (open source) *PyWavelets* library has been exploited, particularly functions 'wavedec', 'waverec' and 'threshold' [13].

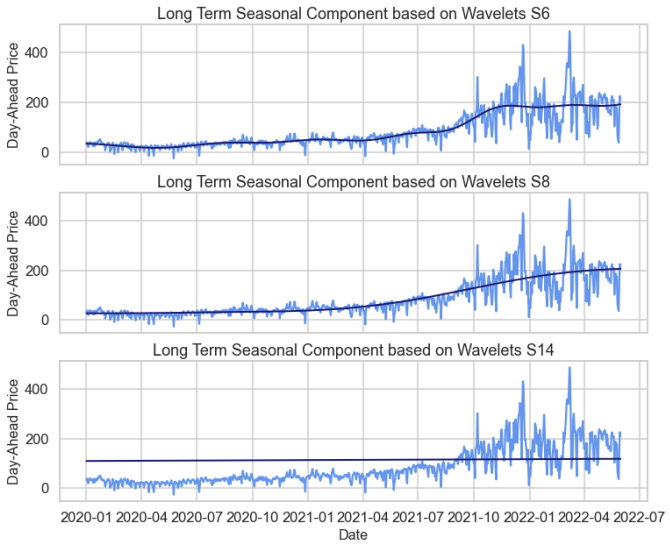


**Figure 7.** Figure of the Long Trend Seasonal Component (LTSC) based on wavelets $S_6$, $S_8$, $S_{14}$ for the daily average prices.

In wavelet smoothing, the time series is decomposed using the discrete wavelet transform into a sum of approximation series capturing the general trend, $S_J$, and a number of detailed series $D_J$ representing the high frequency components: $S_J + D_J + D_{J-1} + \cdots + D_1$, where $J$ is the smoothing level,

$$S_J = \sum_k s_{J,k}\phi_{J,k}(t) \quad \text{and} \quad D_j = \sum_k d_{j,k}\psi_{j,k}(t).$$

The terms $s_{J,k}, d_{J,k}, d_{J-1,k}, \ldots, d_{1,k}$ indicate the wavelet transform coefficients that measure the contribution of the corresponding wavelet function to the approximation sum. At the coarsest scale the LTSC term can be approximated by $S_J$ and more precisely by $S_{J-1} = S_J + D_J$. At each step, we obtain a better estimate of the original signal by adding a mother wavelet $D_j$ of a lower scale $j = J_1, J_2, \ldots$. The reconstruction process can always be interrupted, especially when we reach the desired accuracy.

Daily Average Price:

In Figure 8 we observe the daily average prices after removing the Long Trend Seasonal Component (LTSC) and after replacing negative values with null prices, see also *'deseasonalized.m'* MATLAB code [11]. The seasonal adjustment shown below was obtained with the wavelet family 'db24' on the wavelet $S_8$.

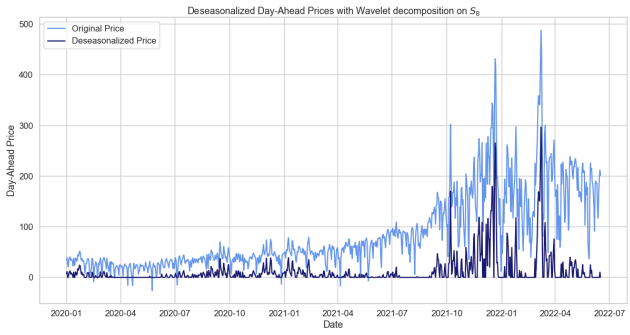


**Figure 8.** Figure of the deseasonalized prices based on wavelet $S_8$ for the daily average prices.

Hourly Daily Price:

For hourly prices, the LTSC decomposition is done with the same MATLAB code used for daily average prices and is performed on the same wavelet family 'db24' using the wavelet $S_{12}$, see Figure 9.

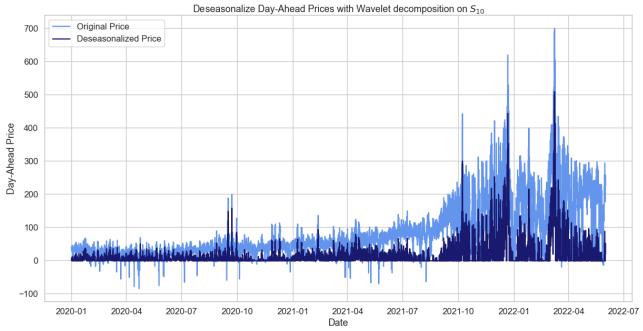


**Figure 9.** Figure of the deseasonalized prices based on wavelet $S_{12}$ for the daily hourly prices.

4.3.2. Box-Jenkins Model

The Box and Jenkins approach [14], ARIMA, consists of the following steps: model identification, parameter estimation, estimate the parameters for the model and model diagnostic [15].

Model Identification:

Since ARMA requires stationarity, standard Box and Jenkins approach suggests both a short and a seasonal differentiation to obtain stationarity of the mean, then performing a logarithmic/power transformation to achieve stationarity in the variance. Analogously, when dealing with seasonal components, we can consider seasonal multiplicative models coupled, when necessary, with long-term differencing to achieve mean-stationarity, see, e.g., [16].

Parameter Estimation

Dealing with a stationary and deseasonalized time series, we can move forward applying ARMA, hence choosing the order of the parameters $p$ and $q$, by exploiting Autocorrelation Function and Partial Autocorrelation Function plots. They, respectively, show correlations of an observation with lag values, a *summary* of correlations between observations and lag values that are not accounted for by prior lagged observations. Models accuracy is provided by the following information criteria:

- **Akaike Information Criteria** (AIC): goodness-of-fit measure of an estimated statistical model, $AIC = -2\log(\mathcal{L}) + 2(p+q+1)$.
- **Bayesian Information Criteria** (BIC): estimate of the Bayes factor for two competing models $BIC = -2\log(\mathcal{L}) + \log(N)(p+q+1)$.

$\mathcal{L}$ indicates the *maximum likelihood function*, while $N$ is the number of observations. Hence, the model providing the minor Information Criteria (IC) is the one to choose, since it identifies both the goodness of the fit and the number of parameters, see, e.g., [17,18].

Model Estimation:

To estimate the coefficients of the previously chosen ARMA$(p,q)$ model we can use criteria such as Maximum Likelihood Estimation and Least Squared Estimation.

Model Diagnostic:

The last step of the Box-Jenkins methodology concerns the diagnostic check to verify that the model is adequate. Box-Pierce and Ljung-Box tests are used to check the correlation of the residuals.

- The Box-Pierce test is defined as

$$Q(k) = N \sum_{i=1}^{k} r_i^2,$$

where $N$ represents the number of observations, $k$ is the length of coefficients to test autocorrelation and $r_i$ is the autocorrelation coefficient for the lag $i$.

The null hypothesis of the Box-Pierce test reads

$$H_0\text{: } \textit{none of the autocorrelation coefficients up to lag k is different from zero,}$$

i.e., the residuals are independently distributed, i.e. white noise, and the model is adequate.

- The Ljung-Box statistics follows this formula:

$$Q^*(k) = N(N+2) \sum_{i=1}^{k} \frac{r_i^2}{(N-i)},$$

where the variables are the same as the Box-Pierce test and the null hypothesis too.

Daily Average Price:

We continue the analysis with statistical models as ARIMA on the deseasonalized time series. Applying the steps required by the Box-Jenkins model we obtained the parameters $p = 2$, $d = 1$ and $q = 1$ using the ACF, PACF and the AIC as information criteria. The parameter $d = 1$ indicates the non-stationarity of our time series, see Section **??**, therefore it must be differentiated with $lag = 1$ to be stationary.

Additionally, the Ljung-Box test was carried out showing a $p$-value of 0.83 so we can not refuse the null hypothesis, i.e. the residuals are independently distributed and we can consider the results obtained by the model valid. In the first Table 11 is shown the RMSE of the ARIMA model trained on 2020 in order to predict 2021. We observe a clear improvement of the RMSE compared to what is seen in Table 3 with SARIMA. However, although the RMSE is not large, this is not a result of the model's ability but only of the seasonal adjustment of the LTSC component, as clearly observed in Figure 10.

| **ARIMA**$(2,1,1)$ | |
|---|---|
| **Training Period** | **RMSE** |
| 2020 | 43.4172 |

**Table 11.** RMSE of ARIMA model of 2020 for testing 2021 on the daily average deseasonalized prices.

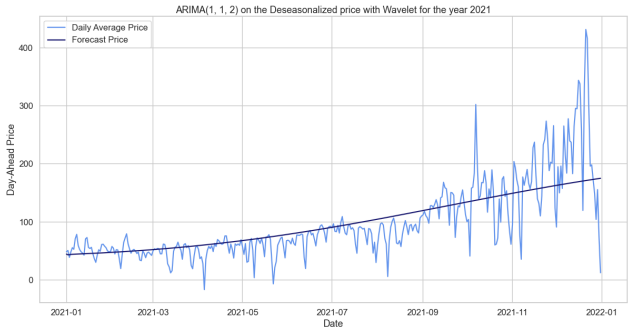


**Figure 10.** Comparison of target and forecast values for 2021 using 2020 as training set with ARIMA$(2,1,1)$ for daily average prices.

In the same way we evaluated and obtained the ARIMA's hyperparameters using the years 2020 and 2021 as training set and testing mid-2022, as shown in Table 12. However, this model in contrast to

the previous one, does not perform better than the SARIMA on the same training and test sets. The latest models designed to forecast two weeks starting from the daily average deseasonalized time series on different time windows. The results of these models are shown in Table 13 and Table 14.

| **ARIMA**$(7,1,1)$ | |
|---|---|
| **Training Period** | **RMSE** |
| 2020-2021 | 77.6933 |

**Table 12.** RMSE of ARIMA on 2020-2021 for testing mid-2022 on daily average deseasonalized prices.

| **RMSE of two weeks of testing in 2021** | | |
|---|---|---|
| **Training Period** | **ARIMA**$(p,d,q)$ | **RMSE** |
| 2020 | ARIMA$(2,1,1)$ | 11.1749 |
| Spring-Summer | ARIMA$(6,2,0)$ | 10.4634 |
| Autumn-Winter | ARIMA$(6,1,6)$ | 11.2119 |

**Table 13.** RMSE of ARIMA model for testing two weeks on daily average deseasonalized prices.

The RMSEs are lower than those obtained from the SARIMA model, see Table 5, trained on the same training and test sets. Figure 11 shows that the model can predict quite well, in truth the model predicts the weekly behaviour. This ARIMA model predicts the two upper peaks quite accurately.

| **Ljung-Box Test on two weeks of test** | | |
|---|---|---|
| **Training Period** | $p$**-value** | **Null Hypothesis** |
| 2020 | 0.83 | not rejected |
| Spring-Summer | 0.24 | not rejected |
| Autumn-Winter | 0.95 | not rejected |

**Table 14.** The $p$-values of the Ljung-Box test for the ARIMA model on two weeks test.

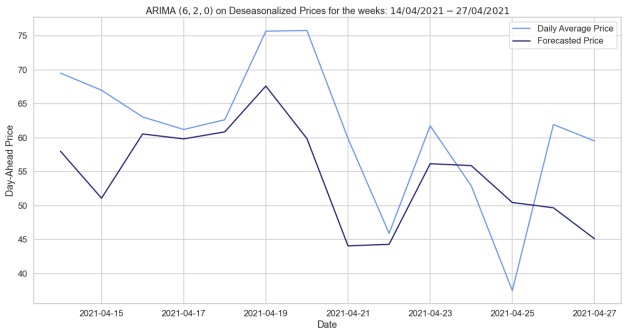


**Figure 11.** Comparison of target and forecast values for two weeks in 2021 using spring-summer as training set with ARIMA on the deseasonalized daily average prices.

Hourly Daily Price:

Let us now consider the hourly prices at which we removed the LTSC component with wavelet decomposition on the wavelet $S_{12}$. After observing plots concerning ACF and PACF and after ascertaining the stationarity of our time series using the ADF test, we obtain the hyperparameters for the different time windows presented in Table 15. Before evaluating the errors, we checked the reliability of the models using the Ljung-Box test in Table 16.

| RMSE of one month of testing in 2021 | | |
|---|---|---|
| **Training Period** | **ARIMA$(p,d,q)$** | **RMSE** |
| 2020 | ARIMA$(4,0,1)$ | 71.1059 |
| Spring-Summer | ARIMA$(2,0,1)$ | 71.0840 |
| Autumn-Winter | ARIMA$(7,0,3)$ | 71.3371 |

**Table 15.** RMSE of the ARIMA model on different training sets for testing one month on the hourly deseasonalized prices.

| Ljung-Box Test on one month of test | | |
|---|---|---|
| **Training Period** | **$p$-value** | **Null Hypothesis** |
| 2020 | 0.97 | not rejected |
| Spring-Summer | 0.99 | not rejected |
| Autumn-Winter | 1 | not rejected |

**Table 16.** The $p$-values of the Ljung-Box test for the ARIMA models on one month test.

Once we have ensured the reliability of the models, we can state that ARIMA on the seasonally adjusted hourly prices performs better than the SARIMA and Naive Benchmark models. We now see in Table 17 the results obtained by ARIMA on different training sets testing one week. The RMSE in the case of the model trained on 2020 is clearly decreased, whereas this is not the case with the model trained on the autumn-winter period.

| RMSE of one week of testing in 2021 | | |
|---|---|---|
| **Training Period** | **ARIMA$(p,d,q)$** | **RMSE** |
| 2020 | ARIMA$(4,0,1)$ | 19.1394 |
| Spring-Summer | ARIMA$(2,0,1)$ | 19.2664 |
| Autumn-Winter | ARIMA$(7,0,3)$ | 19.2074 |

**Table 17.** RMSE of the ARIMA model on different training sets for testing one week on the hourly deseasonalized prices.

As observed for the daily average prices, Figure 12 reveals how the forecasts made by ARIMA trained on 2020 do not predict price trends despite the fact that the error is not high. Looking at Table 18 we see a worsening in the RMSE values by testing a day in 2021, indeed Table 10 is definitely better.

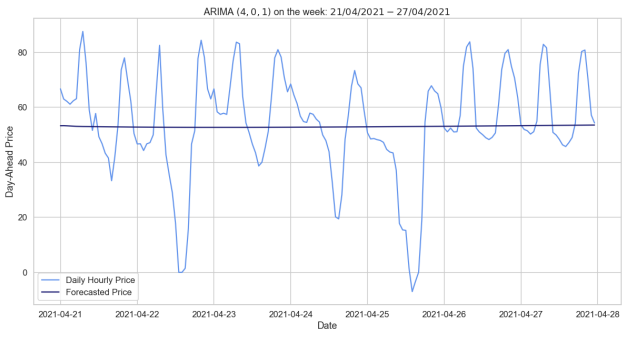


**Figure 12.** Comparison of target and forecast values for one week using 2020 as training set with ARIMA$(4,0,1)$ for daily hourly prices.

| RMSE of one day of testing in 2021 | | |
|---|---|---|
| **Training Period** | **ARIMA$(p,d,q)$** | **RMSE** |
| 2020 | ARIMA$(2,0,1)$ | 16.8249 |
| Spring-Summer | ARIMA$(2,0,1)$ | 16.6854 |
| Autumn-Winter | ARIMA$(6,0,3)$ | 14.2826 |

**Table 18.** RMSE of the ARIMA model of different training sets for testing one day on the hourly deseasonalized prices.

### *4.4. AR with Dummies*

In this Section, we observe the results obtained on deseasonalized daily hourly prices with wavelet $S_{12}$ applying the model explained in Section 2. In Table 19 we observe the errors of the AR model with dummies, these errors are larger than those obtained with SARIMA and ARIMA, probably because we are not considering the exogenous variables. Figure 13 shows the daily hourly prices and those predicted by this model which are very far from the actual prices, which have two peaks, one around 9AM an the second at 7PM which are not predicted by the model.

| AR with Dummies | |
|---|---|
| **Training Period** | **RMSE** |
| 2020 | 26.1521 |
| Spring-Summer | 43.4731 |
| Autumn-Winter | 39.1706 |

**Table 19.** RMSE of the AR with dummies model of different training sets for testing one day on the hourly deseasonalized prices.


**Figure 13.** Comparison of target and forecast values with AR Dummies for one day in 2021 using 2020 as training set on daily hourly prices.

### *4.5. XGBoost*

In order to apply the XGBoost model we have to first select characterizing features, e.g. time, quarter, month, year, day of the week/month/year, week of the year, of our time series as to cast the latter into a supervised learning problem. Next we identify the so called *training set*, composed by $Xtrain$ and $ytrain$, the former being defined on the previously calculated features, while the second contains the German market electricity prices. Next we decide the objective function hyper-parameter, defined by a training loss and a regularization term. The training loss indicates how predictive our model is with respect to the training data, the MSE was chosen. While the regularization term controls the complexity of the model in order to avoid overfitting, the $L_1$ regularization was chosen with $\alpha = 0.1$.

After the model has been fitted on $Xtrain$ and $ytrain$, the XGBoost library allows to display the feature importance. In our case features are ordered (from *heaviers* to *lighters* ones as follows: hour, day

of the year, day of the week, day of the month, month and quarter, and we can then forecast prices of the test set based on $Xtest$.

Daily Average Price:

Running the model on daily average prices over two different time windows, we obtain the results shown in Table 20. The XGBoost library provides the feature importance, as shown in Figure 14 which manifests the features that most influence our endogenous variable, i.e. average electricity prices, in the model with training set over autumn winter and predicting two weeks.

| XGBoost | | |
|---|---|---|
| **Training Period** | **Test Period** | **RMSE** |
| 2020 | 2021 | 90.1138 |
| 2020-2021 | 2022 | 148.2663 |

**Table 20.** RMSE of XGBoost model on different training and test periods for the daily average prices.

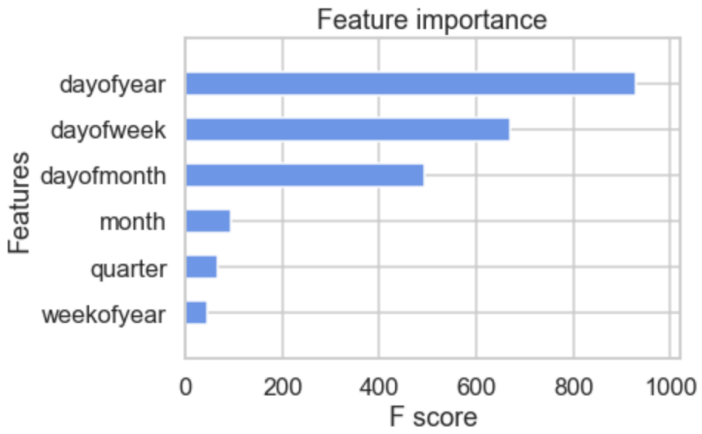


**Figure 14.** Feature importance of the XGBoost on daily average prices for two weeks using the autumn-winter period as training set.

Table 21 shows the errors obtained from the forecasts over different training periods in the two weeks forecast, which are not particularly encouraging as the RMSEs are large. The XGBoost model trained on the autumn-winter period only succeeds in identifying a few negative peaks and fails to predict the behavior of the two weeks as a whole, as Figure 15 reveals.

| XGBoost | | |
|---|---|---|
| **Training Period** | **Test Period** | **RMSE** |
| 2020 | two weeks in 2021 | 46.6614 |
| Spring-Summer | two weeks in 2021 | 47.8949 |
| Autumn-Winter | two weeks in 2021 | 13.2064 |

**Table 21.** RMSE of the XGBoost model on different training periods for testing two weeks in 2021 on the daily average prices.

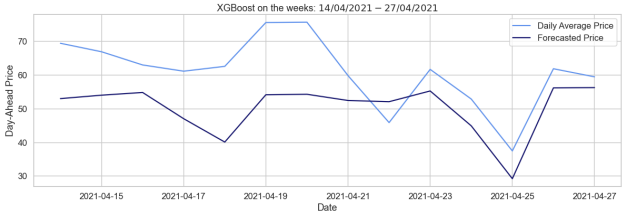


**Figure 15.** Comparison of target and forecast with XGBoost on daily average prices for two weeks using Autumn-Winter as training set.

Hourly Daily Price:

Here we present the XGBoost performed on the daily hourly prices, firstly in the case of one month forecasts, showing that this model is more accurate than the Naive Benchmark but less with respect to ARIMA and SARIMA. The results obtained in the weekly forecast displayed in Table 23 are large errors, considering the RMSEs, excluding the case where autumn and winter 2020-2021 is used as the training set.

| XGBoost | | |
|---|---|---|
| **Training Period** | **Test Period** | **RMSE** |
| 2020 | one month in 2021 | 122.8169 |
| Spring-Summer | one month in 2021 | 114.8268 |
| Autumn-Winter | one month in 2021 | 123.3496 |

**Table 22.** RMSE of the XGBoost model of one month in 2021 for the daily hourly prices.

| XGBoost | | |
|---|---|---|
| **Training Period** | **Test Period** | **RMSE** |
| 2020 | one week in 2021 | 42.4270 |
| Spring-Summer | one week in 2021 | 41.3770 |
| Autumn-Winter | one week in 2021 | 19.7381 |

**Table 23.** RMSE of the XGBoost model of different train periods for testing one week in 2021 on the daily hourly prices.

In the concrete, as shown in Figure 16 depicting the forecast for the week of April 2021, we observe that this model predicts the regular behaviour quite well but fails to predict the positive and negative price peaks present in the German market. In the scenario of hourly pricing, Table 24 illustrates that the XGBoost model works well when utilizing autumn-winter as the training period. Summing up, the XGBoost model performs better than the Naive Benchmark. Nevertheless, it shows considerable errors when applied to our data, probably because of their irregularity, a second possible factor being that, we are considering variables with similar characteristics as exogenous ones.

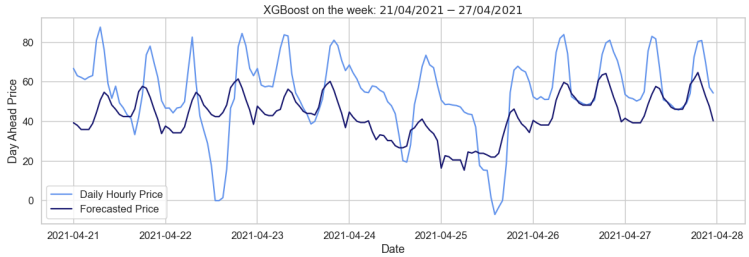


**Figure 16.** Comparison of target and forecast values for one week using Autumn-Winter as training set with XGBoost model on daily hourly prices.

| XGBoost | | |
|---|---|---|
| **Training Period** | **Test Period** | **RMSE** |
| 2020 | one day in 2021 | 40.3105 |
| Spring-Summer | one day in 2021 | 40.9339 |
| Autumn-Winter | one day in 2021 | 7.7653 |

**Table 24.** RMSE of the XGBoost model of different train periods for testing one day in 2021.

### *4.6. Selection of the Network Structure*

Neural Networks require input data series to be characterized by low variance level, otherwise associated training process requires exponentially (in volatility level) growing computational time with low probability of pattern-learning. A possible solution can be found in scaling our time series, hence by a standardization and transformation approach. We consider the Median and Median Absolute Deviation (MAD) to avoid latter cited potential delay. In addition, the NN can be used to model univariate time series forecasting problems, but we can not apply them directly to the time series, as we did with ARIMA; rather, we transformed the time series into a multivariate input sample, where each sample has a specific number of time steps, the output being the value of a single step.

The most common technique for choosing the number of hidden layers and the number of hidden neurons is through experiments, as no fixed methods are provided for finding them, unlike for ARIMA, cf. Section 4.3.2. The Neural Network must be trained, i.e. examples of the problem to be solved must be presented to the network, then connection weights must be adjusted based on the difference between the output obtained and the desired data (ground truth). On the daily average prices, we implemented a simple RNN, but this did not provide noteworthy results as by definition it has no memory cell compared to LSTM.

#### 4.6.1. LSTM

Our data need to be normalized and after that we split the normalized time series into train and test set, then we made the time series a multivariate sample as described at the beginning of this section. At this point, we obtained as $Xtrain$ and $Xtest$ two vectors of dimension $(n, 2)$, where $n$ denotes the time windows chosen as input and test, respectively, containing the electricity prices.

In our LSTM the Adam algorithm, a stochastic gradient descent method based on an adaptive estimation of first and second order moments [19], has been chosen as optimizer with a learning rate of 0.001. In all the Neural Network based methods, we considered Mean Squared Error as the loss function, minimized throughout the training process. Furthermore all the NNs are defined with LSTM hidden layers followed by a dropout layer. This dropout layer randomly sets input to zero, with a frequency given by the rate, at each step during the training. Instead, the inputs that are not set to zero are scaled up by $1/(1 - \text{rate})$. The batch size defines the number of samples to work on before the internal parameters are updated, usually 32, 64, 128 and 256.

Daily Average Price:

We will show the results obtained on the LSTM model on the daily average prices using different training and test sets. The model chosen to predict the daily average prices for the year 2021 relies on a NN made up of 4 layers: input, hidden, dropout and output. In particular, the hidden layer has 300 LSTM cells followed by a dropout layer with rate=0.4. The dropout layer can be seen as a regularization technique to reduce overfitting, while a fully connected layer with one neuron is chosen as the output layer. Let us briefly note that in Python we used the *Keras* library, hence exploiting classes for the layers 'LSTM', 'Dropout' and 'Dense'.

| LSTM with one hidden layer | | |
|---|---|---|
| **Training Period** | **Epochs** | **RMSE** |
| 2020 | 250 | 34.0001 |

**Table 25.** RMSE of the LSTM of training period 2020 for testing one year on the daily average prices.

In Table 25 we observe the results obtained from the training of an LSTM with a training set of one year and a test set of the following year. Our NN is trained for 250 epochs starting with arbitrary weights, which are updated at each step minimizing the loss function chosen, in our case the MSE with Adam optimizer in which the learning rate is 0.001 and the batch size is 60.

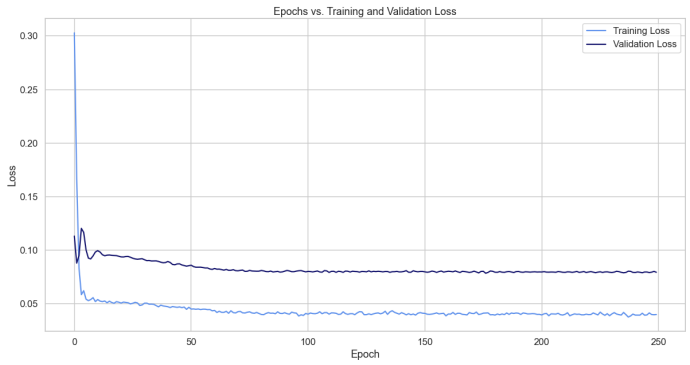


**Figure 17.** Loss during the training and validation process of an LSTM for one year testing.

Furthermore, during the training of the NN, we use the month of December 2020 as validation set; we display the training and validation loss for each epoch in Figure 17. As desired, the loss function drops and approaches zero as the epochs progress; however the validation loss does not reach the training loss, meaning that our model is probably underfitting. The result displayed in the previous Table 25 and Figure 18 shows that this NN performs very well on our time series, in fact the RMSE is lower compared to all the models analyzed before.

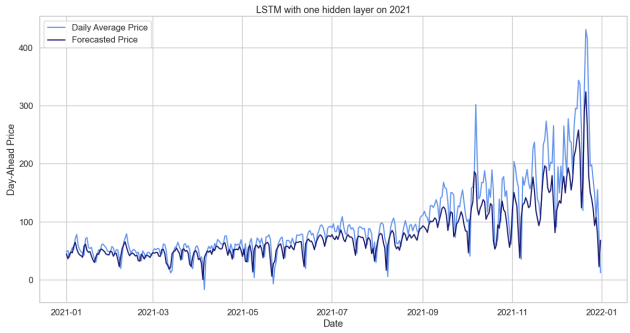


**Figure 18.** Comparison of target and forecast values for the year 2021 using 2020 as training set with LSTM model on daily average prices.

The LSTM architecture for predicting mid-2022 is defined with input and output layers like the one of the previous model, while it has 2 hidden layers of 300 LSTM cells each, both followed by a dropout layer with rate 0.2. The training is done with the same optimizer and loss function.

| LSTM with two hidden layers | | |
|---|---|---|
| **Training Period** | **Epochs** | **RMSE** |
| 2020-2021 | 250 | 57.8398 |

**Table 26.** RMSE of LSTM of training period 2020-2021 for testing mid-2022 on the daily average prices.

Table 26 shows a smaller RMSE, i.e. a better forecast; however, even the LSTM does not perform very well in the case of this forecast. The price behaviour in 2022 is totally irregular and different from previous years under consideration. Finally, in order to predict two weeks of daily average prices in 2021, we implemented LSTMs with a single hidden layer defined as an LSTM of 400 cells followed by a dropout layer with rate 0.2.

| LSTM with one hidden layer | | | |
|---|---|---|---|
| **Training Period** | **Hidden Layers** | **Epochs** | **RMSE** |
| 2020 | 400 units | 300 | 11.0914 |
| Spring-Summer | 400 units | 300 | 9.5359 |
| Autumn-Winter | 400 units | 300 | 13.8709 |

**Table 27.** RMSE of the LSTM model of different training periods for testing two weeks in 2021 on daily average prices.

The results obtained from this LSTM, in Table 27, are not optimal, if we compare them to the models previously analyzed on the same training and test sets they are slightly lower. Figure 19 suggests that our LSTM predicts well the general behavior excluding the fact that it has a lower price range, this is due to the fact that our model can not predict the trend in our time series.

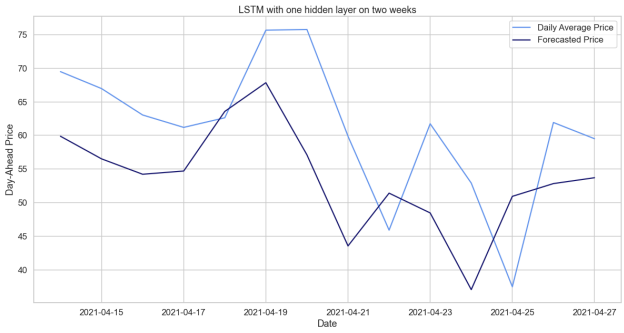


**Figure 19.** Comparison of target and forecast values for two weeks using 2020 as training set with LSTM model on daily average prices.

Hourly Daily Price:

Here we report the results obtained using different training and testing periods on hourly data. Considering hourly data enables us to provide the NN significantly more data than those in the preceding subsection, which typically results in improved accuracy. Each of the three hidden layers in the first NN, which has 300 LSTM cells in each, is followed by a dropout layer, whose purpose it is to prevent overfitting, at a rate of 0.2. Moreover, the training is carried out with a batch size of 60.

| LSTM with different number of hidden layers | | | |
|---|---|---|---|
| **Training Period** | **Epochs** | **LSTM Layers** | **RMSE** |
| 2020 | 300 | 3 | 67.5297 |
| Spring-Summer | 300 | 2 | 75.1064 |
| Autumn-Winter | 300 | 3 | 74.7541 |

**Table 28.** RMSE of the LSTM model of different training periods for testing one month in 2021.

Both the 2020 training and the autumn-winter training adopt the initial neural network with the previously described architecture. The amount of data available, notably in the case of 2020, and the increased complexity of the neural network itself are undoubtedly responsible for the fact that in both situations the training implementation time takes few minutes. The neural network, implemented considering spring-summer as training period, has two LSTM layers with 300 cells each, followed by a dropout layer with a rate of 0.2 and batch size of 60.

Table 28 presents the RMSE of these NNs, which have been verified as accurate after looking at the loss functions of both the training and validation sets. However, a closer look at Figure 20 suggests that LSTM is better able to represent the behaviour of the time series than ARIMA, even though the RMSE of these NNs is marginally greater than the one achieved with the ARIMA model in Table 15.

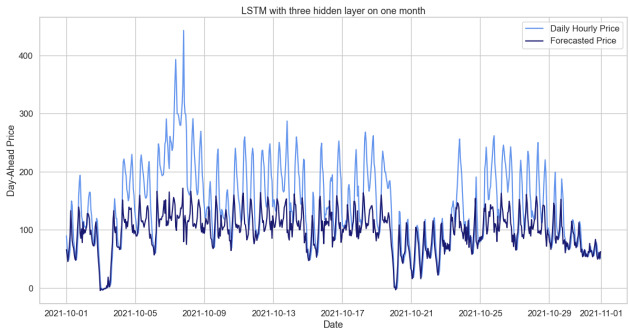


**Figure 20.** Comparison of target and forecast values for one month using 2020 as training set with LSTM model on daily hourly prices.

The predictions generated by the LSTM trained on 2020 accurately forecasted negative peaks, which were barely predicted by the other models. One week forecasts based on 2020 hourly prices are computed via a NN with three hidden LSTM layers, with each having a unit size of 300, followed as broadly explained by a 0.2 rate dropout. In the spring-summer training, we define two LSTMs with 250 units each, whereas in the autumn-winter training period we used two LSTMs with 300 units each. Take into consideration that the first NN's batch size is 60, while the other NN's batches are 30.

| **LSTM with different number of hidden layers** | | |
|---|---|---|
| **Training Period** | **Epochs** | **RMSE** |
| 2020 | 300 | 11.3126 |
| Spring-Summer | 150 | 14.3904 |
| Autumn-Winter | 250 | 12.6480 |

**Table 29.** RMSE of the LSTM model of different training periods for testing one week in 2021.

The errors from the recently introduced Neural Networks are shown in Table 29, and during this test period, the LSTM remains to be the most accurate model. In order to estimate the hourly prices of a day on 3 training time windows, two distinct NN are designed. With one hidden layer made up of 300 LSTM cells spread throughout 250 epochs, the one trained on 2020 and autumn-winter is the same. The single hidden layer with 500 units was chosen as the NN using spring-summer as training set.

| **LSTM with different number of hidden layers** | | |
|---|---|---|
| **Training Period** | **Epochs** | **RMSE** |
| 2020 | 250 | 7.2073 |
| Spring-Summer | 250 | 8.2753 |
| Autumn-Winter | 250 | 8.2147 |

**Table 30.** RMSE of the LSTM model for different training periods for testing one day in 2021.

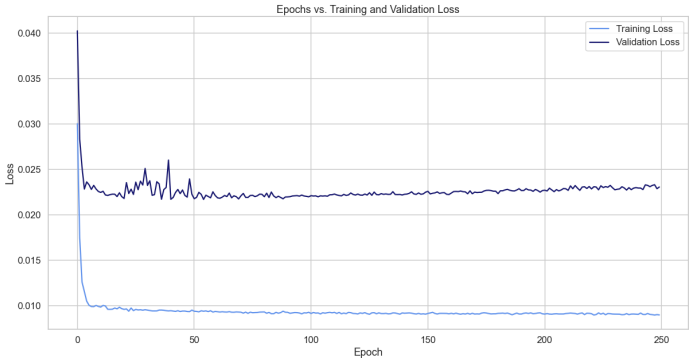


**Figure 21.** Loss during the training and validation process of an LSTM for one day testing.

Figure 21 shows the loss function for the training and validation set. We see that LSTM is the best model among the tested ones to predict our data. Comparing the hourly prices predicted by the LSTM with the actual prices (Figure 22), we can see how accurately the behaviour and peaks are predicted.

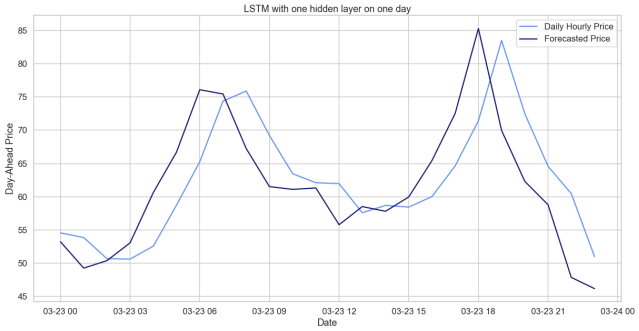


**Figure 22.** Comparison of target and forecast values for one day using 2020 as training set with LSTM model on daily hourly prices.

## 5. Conclusions

In the present work we compared statistical, similar-day and machine learning approaches to the Electricity Price Forecasting problem with respect to German market electricity prices, within the period ranging from 2020 to the mid of 2022.

The latter having being characterized by a high grade of volatility, with several up and downs and irregular seasonality components, also caused by exogenous socio-political events as the well known COVID19-pandemic, the climate related energy crisis and the Ukraine-Russia war. Our analysis has shown that an LSTM-based approach outperforms all other models when evaluating a medium term forecast, by using daily average prices, as well as when dealing with short term predictions based on hourly prices.

We have also shown that removing the Long Trend Seasonal Component (LTSC) and applying the ARIMA model on deseasonalized prices, lead to errors mitigation. Unfortunately, this model worked poorly when we examined plots emphasizing differences between the actual and forecasted values. The XGBoost model performed better than the Naive Benchmark model, even if predicted prices are significantly less than the ones obtained with SARIMA and ARIMA.

As a result of our analysis, we claim that exogenous variables such as, day-ahead system load forecasts, day ahead wind power generation, may be taken into account to improve obtained forecasts by exploiting more structured models, as, e.g. the Neural Basis Expansion Analysis with exogenous variables (NBEATSx) one, recently introduced as an extension of the Neural Basis Expansion Analysis

(NBEATS) approach. A possible further improvement could be achieved studying outliers behavior; i.e. the predictions of "normal" prices and spiky prices is carried out and then compared. Further possible alternatives, within the field of Electricity Price Forecasting research, rely on hybrid models, i.e. models obtained by combining statistical and NN models.

**Author Contributions:** All authors contributed equally to this work.

**Funding:** This research received no external funding.

**Acknowledgments:** Aurora Poggi would like to underline that the material developed within the present paper has been the result of her Erasmus Internship she spent exploiting the Erasmus + agreement between the University of Wuppertal and the Department of Computer Science of the University of Verona, collaborating with Prof. Matthias Ehrhardt, under the supervision of Prof. Luca Di Persio.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Acronyms

**ACF** Autocorrelation Function.
**ADF** Augmented Dickey-Fuller.
**AIC** Akaike Information Criteria.
**AR** Autoregressive.
**ARIMA** Autoregressive Integrated Moving Average.
**ARMA** Autoregressive Moving Average.
**ARX** Autoregressive with exogenous inputs.

**BIC** Bayesian Information Criteria.

**CNN** Convolutional Neural Network.

**DNN** Deep Neural Network.

**EPF** Electricity Price Forecasting.

**GARCH** Generalized Autoregressive Conditional Heteroskedasticy.

**IC** Information Criteria.

**K-NN** K-Nearest Neighbourhood.
**KPSS** Kwiatkowski Phillips Schmidt Shin.

**LASSO** Least Absolute Shrinkage and Selection Operator.
**LSTM** Long Short Term Memory.
**LTSC** Long Trend Seasonal Component.

**MA** Moving Average.
**MAD** Median Absolute Deviation.
**MADE** Mean Absolute Deviation.
**MAPE** Mean Absolute Percentage Error.
**MLP** Multilayer Perceptron.
**MRAE** Median Relative Absolute Error.
**MSE** Mean Squared Error.

**NBEATS** Neural Basis Expansion Analysis.
**NBEATSx** Neural Basis Expansion Analysis with exogenous variables.
**NN** Neural Network.
**NOT** Narrowest Over Threshold.

**PACF** Partial Autocorrelation Function.
**PP** Phillips Perron.

**RBF** Radial Basis Function.
**RMSE** Root Mean Squared Error.
**RNN** Recurrent Neural Network.

**SARIMA** Seasonal Autoregressive Integrated Moving Average.

**XGBoost** Extreme Gradient Boosting.

## References


1. Nitka, W.; Serafin, T.; Sotiros, D. Forecasting Electricity Prices: Autoregressive Hybrid Nearest Neighbors (ARHNN) Method. Computational Science – ICCS 2021; Paszynski, M.; Kranzlmüller, D.; Krzhizhanovskaya, V.V.; Dongarra, J.J.; Sloot, P.M., Eds.; Springer: Cham, 2021; pp. 312–325.
2. Nasiadka, J.; Nitka, W.; Weron, R. Calibration window selection based on change-point detection for forecasting electricity prices, 2022.
3. Bollerslev, T. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics* **1986**, *31*, 307–327.
4. Novak, J.; Cañas, A. The origins of the concept mapping tool and the continuing evolution of the tool. *Information Visualization* **2006**, *5*, 175–184.
5. Singh, P.; Kaushik, R.; Ceylan, M.; Prokofjevs, A. Analysis and Forecasting of Coal Prices **2021**.
6. XGBoost, 2021. https://xgboost.readthedocs.io/en/stable/.
7. Chen, T.; Guestrin, C. XGBoost: A scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on knowledge discovery and data mining. ACM, 2016, KDD '16, pp. 785–794.
8. Saâdaoui, F.; Rabbouch, H. A wavelet-based hybrid neural network for short-term electricity prices forecasting. *Artificial Intelligence Review* **2019**, *52*, 649–669.
9. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Computation* **1997**, *9*, 1735–1780.
10. Weron, R. Electricity Price forecasting: A review of the state of the art with a look into the future. *International Journal of Forecasting* **2014**, *30*, 1030–1081.
11. Weron, R. *Modeling and forecasting electricity loads and prices: A statistical approach*; Vol. 403, John Wiley & Sons, 2007.
12. Jedrzejewski, A.; Marcjasz, G.; Weron, R. Importance of the long-term seasonal component in day-ahead electricity price forecasting revisited: Parameter-rich models estimated via the LASSO. *Energies* **2021**, *14*, 3249.
13. Lee, G.; Gommers, L.; Waselewski, F.; Wohlfahrt, K.; O'Leary, A. PyWavelets: A Python package for wavelet analysis. *Journal of Open Source Software* **2019**, *4*, 1237.
14. Box, G.E.P.; Jenkins, G.M.; Reinsel, G.; Ljung, G. *Time series analysis : forecasting and control*; Wiley: San Francisco, 2015.
15. Tang, Z.; De Almeida, C.; Fishwick, P.A. *Time series forecasting using neural networks vs. Box-Jenkins methodology*; 1991.
16. Makridakis, S.; Hibon, M., ARMA Models and the Box-Jenkins Methodology; Journal of Forecasting, 1997; Vol. 16, pp. 147–163.
17. Choon, O.H.; Chuin, J.L.T. A Comparison of Neural Network Methods and Box-Jenkins Model in Time Series Analysis. Proceedings of the Fourth IASTED International Conference on Advances in Computer Science and Technology; ACTA Press: USA, 2008; p. 344–350.
18. De-Graft Acqua, H. Comparison of Akaike information criterion (AIC) and Bayesian information criterion (BIC) in selection of an asymmetric price relationship. *Journal of Development and Agricultural Economics* **2010**, *2*, 1–6.
19. Diederik, P.K.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* **2014**.