MF6016

Dissertation in Financial and Computational Mathematics

Nathaniel Dureza Volfango
115321436

Academic Supervisor: Dr. Tom Carroll
Industrial Supervisor: Joseph Collins
Coordinator: Dr. Cónall Kelly

Electricity Price Forecasting

School of Mathematical Sciences
University College Cork
Ireland
September 2020

Abstract

Accurate electricity price forecasting has become a substantial requirement since the liberalisation of the electricity markets, which has created a more competitive system of electricity generation and distribution. For example, electricity prices in the day-ahead market (DAM) are considered by energy traders when making decisions. However, electricity prices display characteristics that are difficult from a modelling perspective, including heteroscedasticity (non-constant variance), sharp price spikes and several levels of seasonality.

In this paper, we present some of the methodologies in the current wealth of literature on electricity price forecasting, with the aim of forecasting electricity spot prices in the Irish DAM. More specifically, we use methods in time series and machine learning to forecast prices for all 24 hours of each DAM auction operated by the Single Electricity Market Operator (SEMO).

First, we give a brief overview of the electricity market and its operation as far as it relates to our goal of price forecasting in the DAM. Secondly, some important information is given about the data used for this project. Thirdly, we give a short explanation of each model that was used and then provide details on how each one was structured and prepared for the modelling task. Then, most importantly, we present results on the performance of each model relative to a benchmark and a brief note on the importance of predictors. Finally, we give conclusions based on the results and note some areas for further development.

Contents

1 Introduction

Although electricity is traded as a commodity, it has certain characteristics that influence the balance of the demand and supply and hence require particular consideration:

1. In economic terms, **electricity has an inelastic demand**. This means that large changes in the price will not significantly affect the quantity bought or consumed in the short term.
2. **Electricity cannot be stored easily** like other commodities (e.g. wheat, gold, water, natural gas, etc.), which in turn requires that the volume of the demand and supply are continuously and perfectly balanced.
3. **Electricity has no close substitute**. There will always be a continuous demand for electricity, requiring a constant and stable power supply.

1.1 Historical Context

These characteristics of electricity, along with a few others such as regulatory and environmental requirements, pose a high barrier to entry, as with many companies in the utilities sector, thus creating natural monopolies in electricity generation and distribution. Hence, the infrastructure of the European energy industry became segregated, with each country or region developing its own market with a distinct set of rules and governance procedures. In Ireland, the electricity and gas sectors were dominated by the Electricity Supply Board and Bord Gáis, respectively. These were vertically integrated and state-owned, which meant that everything from electricity generation to the distribution to end consumers was done by one entity.

Over time, the different markets began introducing competition to reduce costs and increase efficiency by privatising the electricity supply chain. Markets also became more physically interlinked, allowing for more efficient generation and consumption. For example, power generators that produced too much electricity could now make use of electricity interconnectors to sell this surplus electricity to another market region. While this did not remove all inefficiencies in the systems, this integration between electricity markets, referred to as *market coupling*, was considered to be a step towards further energy liberalisation.

As another step of market coupling, the wholesale electricity market, called the Single Electricity Market (SEM), was set up in 2007, which combined the markets for the Republic of Ireland and Northern Ireland into one. Then several years later, the SEM was replaced by the Integrated Single Electricity Market (I-SEM) in October 2018 which brought Ireland in line with the European Target Model, which is the blueprint for integration across electricity markets in Europe.

1.2 Basic Electricity Market Structure

There are several markets within the I-SEM, but there are three for which the delivery of electricity in each hour of the day is decided. The auctions are set up in a way that gives market participants the freedom to place bids ahead of time while also being able to balance their overall positions closer to the day of delivery which, for an example, we will refer to as day D. Day $D-1$ is the day on which the day-ahead market auction takes place.

1. **Day-Ahead Market (DAM)**: Bids are submitted up to 19 days before day $D-1$ for hourly delivery periods in the interval [23:00 $D-1$, 23:00 D]. This market closes at around 11:00 $D-1$. A market coupling algorithm called *EUPHEMIA* is used to set the market prices and decide on the most optimal distribution of power.

2. **Intraday Market (IDM)**: Participants can adjust their positions in this market. The intraday market is split up into three auctions, with half-hourly delivery periods: IDA-1, IDA-2 and IDA-3. The IDA-1 auction starts at 17:30 $D-1$, covering all 24 delivery hours; IDA-2 starts at 8:00 D, covering the last 12 hours; IDA-3 starts at 14:00 D, covering the last 6 hours.

3. **Balancing Market (BM)**: Once the two 'ex-ante' markets are closed for a given delivery hour, suppliers can no longer trade for that hour and they must take the price that is set. The balancing market ensures that the demand and supply of energy is perfectly matched. The Transmission System Operator (TSO) decides how the demand is met by calling on generators to deliver power as needed. The TSO in Ireland is EirGrid, and SONI for Northern Ireland.

The focus of this project is modelling and forecasting DAM prices, and when we refer to 'market clearing prices', 'electricity prices', or where there is any ambiguity, assume that we are referring to DAM prices, unless specified otherwise. We will not be looking at the IDM because it has very low trading volume and sparser data. Also, modelling the BM requires very different datasets to what is available to us and is commonly used in the DAM, although we will use the BM data as an explanatory variable. We will say more about the datasets used in Section 2.

For convenience in grouping and manipulating the data, we have shifted all the prices and other time series data forward by one hour, so that a full set of day-ahead prices begins at hour 0 and the last delivery hour is hour 23.

For another brief overview of the Irish electricity market, see [1]

2 Data

In this section, we will explain some of the features of DAM electricity prices that require proper consideration from a modelling perspective. We will also give some details on the dataset that was used. The available data starts from the 12$^{\text{th}}$ November 2018 and ends on the 9$^{\text{th}}$ July 2020, so all our data corresponds to the markets within the I-SEM framework. Figure 1 shows a plot of the DAM electricity prices for the available period.

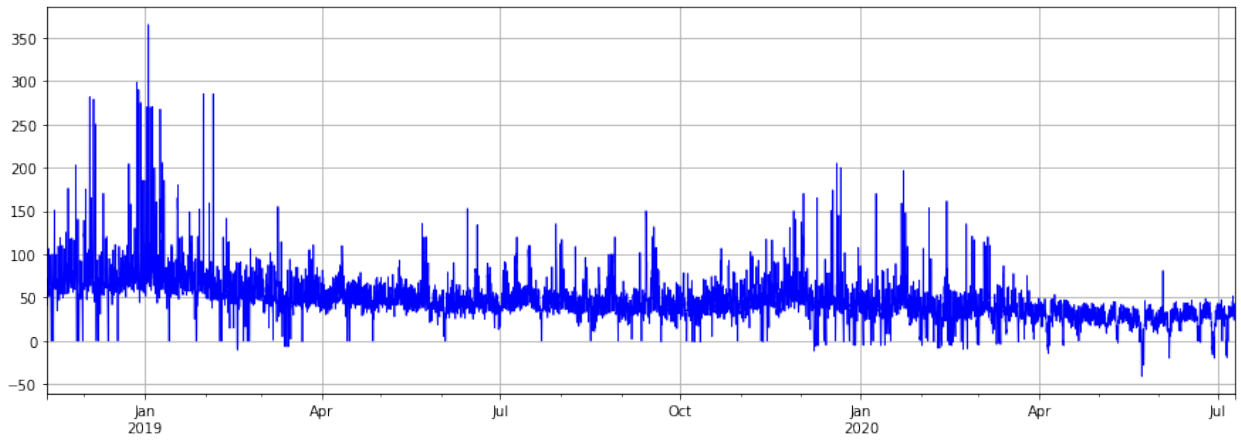


Figure 1: Market clearing prices (in €/MWh) for the day-ahead market from 2018-11-12 to 2020-07-09

2.1 Key Features of DAM Electricity Prices

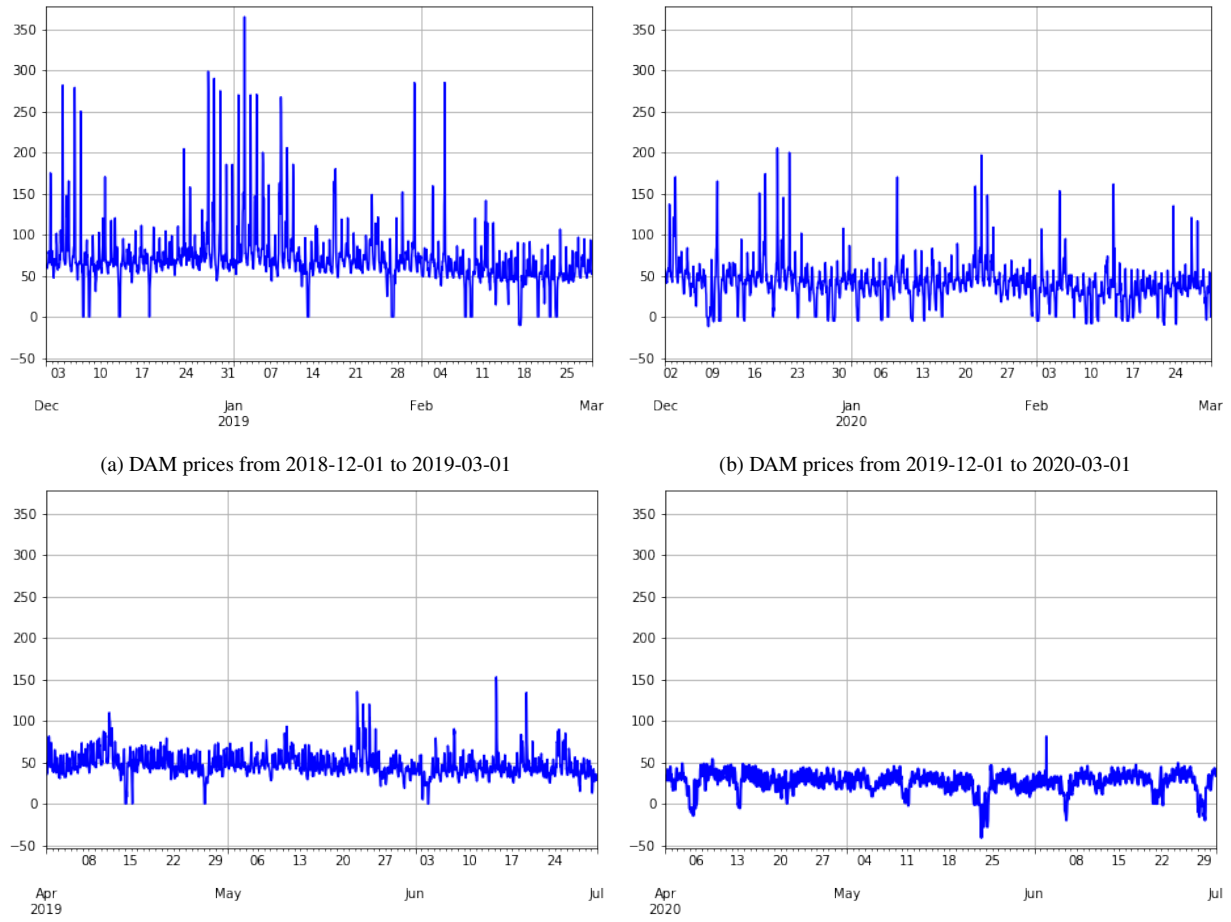
DAM prices have certain defining characteristics that we would expect our model to capture. If these characteristics cannot be properly accounted for in a model, then it would most likely be insufficient for accurate price forecasting. Figure 2 shows DAM prices at different periods, with the same y-axis limits for comparison of scale. We will use the plots to highlight the key characteristics of DAM prices presented here.

2.1.1 Heteroscedasticity and Price Spikes

In general, an increase in supply (or a reduction in demand) results in lower prices while a reduction in supply (or an increase in demand) results in higher prices. Due to the requirement to continuously maintain the balance between supply and demand, shocks to the system would usually result in price spikes (or periods of high volatility).

Examples of such shocks on the supply side are outages or higher than expected wind forecasts. On the demand side, one could have sudden hot/cold temperatures or, more recently since February 2020, a viral pandemic resulting in a partial or complete lockdown in different countries (which in turn causes a reduction in industrial electricity demand, partially offset by higher residential use).

Figures 2a and 2b corresponding to the winter months, where heating is in heavy use, show price spikes with higher intensity than those in Figures 2c and 2d corresponding with the summer months.



(a) DAM prices from 2018-12-01 to 2019-03-01

(b) DAM prices from 2019-12-01 to 2020-03-01

(c) DAM prices from 2019-03-01 to 2019-07-01

(d) DAM prices from 2020-03-01 to 2020-07-01

Figure 2: Electricity spot prices for different periods

Models that assume constant variance can still be used with great accuracy using data preprocessing methods such as factorial ANOVA[2].

Models used in electricity price forecasting consist of Markov regime-switching and jump diffusion. Geman and Roncoroni [3] use a mean-reverting jump diffusion model. On the other hand, there are many papers that use hybrid models, e.g. where spikes are modelled separately (see Cartea and Figueroa [4]; Janczura et al. [5]; Hayfavi and Talasli [6]; Janczura and Weron [7]; Eichler and Türk [8]).

2.1.2 General Trend

We note a slight downward trend in the prices during the period (December 2018 to July 2020).

Comparing the two December-March periods (Figures 2a and 2b), the price spikes are smaller in magnitude in Figure 2b than in Figure 2a. Moreover, the non-extreme prices tend to move between 50 €/MWh and 100 €/MWh in Figure 2a and between 25 €/MWh and 75 €/MWh in Figure 2b.

Similarly with the two April-July periods (Figures 2c and 2d), the price spikes are smaller in

magnitude in Figure 2d than in Figure 2c. In fact, there is only one noticeable positive price spike and significantly more negative price spikes in Figure 2d. The non-extreme prices for these periods tend to move around 50 €/MWh in Figure 2c and between 30 €/MWh and 40 €/MWh in Figure 2d.

Although we believe that the COVID-19 pandemic is one factor, another key reason for this would be the continual increase in the number and capacity of wind generators in Ireland. In February 2020, wind power accounted for 56% of electricity demand during the period, compared to 47% in February 2019 [13]. Wind generators have zero marginal cost (e.g. fuel and startup costs) compared to traditional power generators. These lower costs would result in lower market clearing prices.

2.1.3 Negative (and Zero) Prices

Zero and negative prices are fairly common in electricity markets, which might seem unusual in a market—why would a rational market participant give away a valuable commodity, or much less pay someone else to take it?

DeliveryPeriod	EURPrices
2020-05-22 23:00:00	-1.03
2020-05-23 00:00:00	-1.02
2020-05-23 01:00:00	-10.00
2020-05-23 02:00:00	-22.89
2020-05-23 03:00:00	-29.81
2020-05-23 04:00:00	-39.97
2020-05-23 05:00:00	-41.09
2020-05-23 06:00:00	-33.52
2020-05-23 07:00:00	-20.00
2020-05-23 08:00:00	-5.00
2020-05-23 09:00:00	-1.01
2020-05-23 10:00:00	-1.03
2020-05-23 11:00:00	-5.59
2020-05-23 12:00:00	-11.61
2020-05-23 13:00:00	-15.18
2020-05-23 14:00:00	-27.93
2020-05-23 15:00:00	-27.93
2020-05-23 16:00:00	-10.00
2020-05-23 17:00:00	-1.00

Figure 3: Negative DAM prices

One key reason is that a generator has much lower marginal costs relative to fixed costs, so it can often be cheaper for generators to continue running and selling their electricity at a loss than to have to shut down only to turn back on again later.

We can see from Figures 2a, 2c and 2b that the price touches the zero line several times, and even goes negative from time to time. We also note that Figure 2d has lower prices on average and noticeably more negative prices than the other periods, particularly from end of May 2020. In fact, the largest value in the negative direction for the whole period occurred on the 23rd May 2020 at 5:00.

Figure 3 shows the list of negative prices for that day which is the longest series of negative prices for the whole period and the biggest in terms of magnitude. This reflects the impact of the COVID-19 pandemic (although we note that the impact is not as drastic as in most other markets (e.g. bonds, equities, etc.) since, as we said, electricity is a unique and almost irreplaceable commodity).

The existence and frequency of zero and negative electricity prices rules out models with a non-negative output space, e.g. Black-Scholes.

2.1.4 Multiple Seasonal Cycles

Seasonality is a feature in time series analysis which refers to effects that occur at regular intervals. As we will see later, electricity prices can have at least a daily and weekly seasonality. That is, day D prices have a strong dependency on $D-1$ and $D-7$ prices.

Multiple seasonality effects in electricity prices should not be a surprise, again, due to the inelastic demand of electricity prices. In general, energy consumption follows a pattern (e.g. most people are asleep in the early hours of the day and are at work throughout the afternoon, etc.), hence the daily seasonality, and a similar reasoning for weekly seasonality applies.

Looking back at Figure 2, there is a noticeable similarity in overall structure between the winter periods (Figures 2a and 2b) and also between the summer periods (Figures 2c and 2d). The increased demand (e.g. for heating) is reflected in higher prices on average and more frequent price spikes.

In time series analysis, the SARIMA/SARIMAX model allows for directly modelling the seasonal effects (see Weron[9] for a discussion on this). Another widely used approach in the price forecasting literature is to forecast separately across the hours, which involves creating 24 models, one for each hour of the day (see Karakatsani and Bunn [10, 11]; Misiorek et al. [12]).

Several models like the neural networks and ensemble methods (e.g. random forests and bagging), do not explicitly allow for seasonal effects but are flexible enough that they can be incorporated when the data is processed in the right manner. For example, using price data from 1 day ago, 7 days ago and 1 year ago as explanatory variables may be sufficient for a model to learn the corresponding daily, weekly and annual seasonal effect.

2.1.5 Other Datasets

For this project, we also used DAM bid/ask curves, balancing market prices, wind forecast and demand forecast. Figure 4 shows a plot of BM prices (in terms of €/MWh), wind forecast and demand forecast (both in terms of MW) for the same period.

Compared to DAM prices, BM prices are more volatile, with negative prices and price spikes occurring more frequently. On the 24^{th} January 2019, the BM prices reached as high as €3,773.7/MWh (see Figure 4), which meant that generators who underdelivered (i.e. fell short of the amount of power they promised to sell in previous auctions) had to buy it back at this astronomical price. Because the price spike seems like a once-off event, this could be filtered out for modelling purposes, but we decided not to do this.

When trading electricity for day D, the wind and demand forecasts for that day are available a few hours before the market auction for day D closes, so it is a given that market participants would use this data to inform their trading. We have incorporated these forecasts (and their lagged values) into our models.

The bid/ask curves dataset provides a list of price/quantity pairs corresponding to various buy and sell orders that were placed during trading (see Section 3.4 for more information on how to interpret the data).

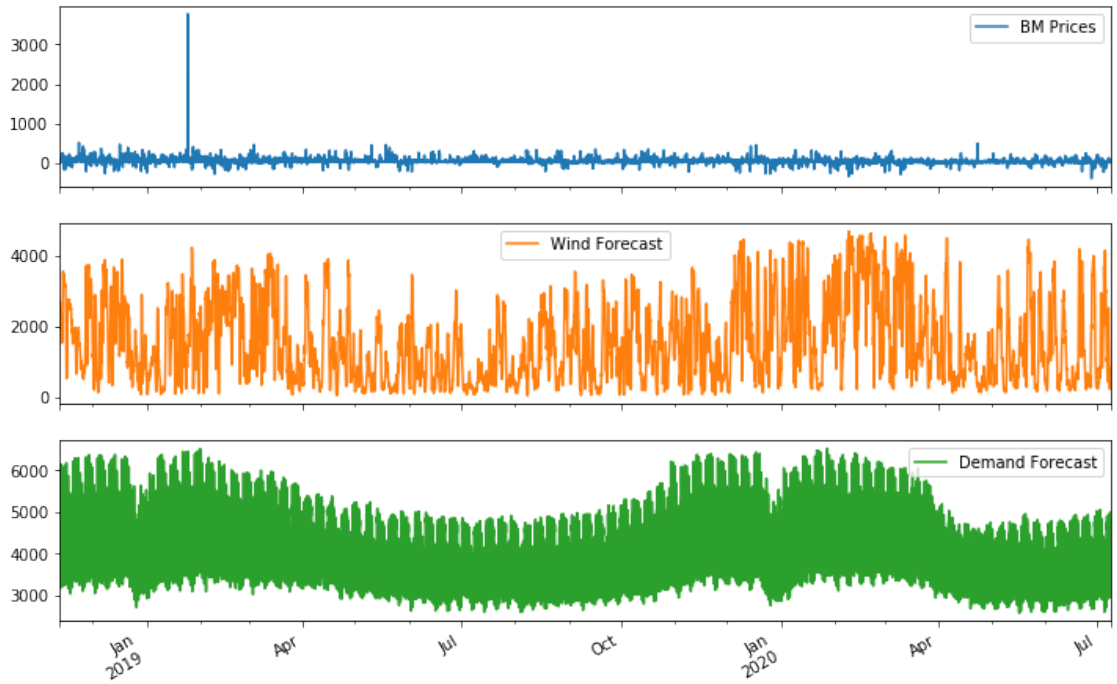


Figure 4: Data for balancing market and forecasted wind and demand load from 2018-11-12 to 2020-07-09

2.2 Data Sources

For readers interested in accessing the publicly available datasets, we provide some details here. More detailed information is available in the website of SEMOpx [14] (for DAM prices and Bid/Ask curves) and SEMO [15] (for BM prices and Wind/Demand forecast). The files for the datasets can be accessed through the "Market Data" section on SEMO/SEMOpx using the report names given below.

2.2.1 DAM Prices

Document Name: SEMOpx Data Publication Guide (version 6.0 – 05 February 2020)
Report Name: MarketResult_SEM-DA_PWR-MRC-D+1
File Format: CSV

2.2.2 BM Prices

Document Name: SEMO Data Publication Guide (version 3.1 – 28 February 2020)
Report Name: PUB_30MinAvgImbalPrc
File Format: XML

2.2.3 Wind Forecast

Document Name: SEMO Data Publication Guide (version 3.1 – 28 February 2020)
Report Name: PUB_4DayAggRollWindUnitFcst
File Format: XML

2.2.4 Demand Forecast

Document Name: SEMO Data Publication Guide (version 3.1 – 28 February 2020)
Report Name: PUB_DailyLoadFcst
File Format: XML

2.2.5 DAM Bid/Ask Curves

Document Name: SEMOpx Data Publication Guide (version 6.0 – 05 February 2020)
Report Name: BidAskCurves_ROI-DA
File Format: XML

3 Models and Methodology

In this section, we present the models that were used in the project. We will give a broad overview of some of the key concepts required to understand the particular model/s. We cannot hope to give a comprehensive survey of each model in only a few pages, but we hope that the condensed information is enough to provide a basic intuition of the underlying mechanics. For readers who are interested in reading further into the subject, we have included some resources for reference at the start of each subsection (except for the Naive model, which is basic enough to be understood here).

Weron [16] gives a comprehensive overview of the academic literature in electricity price forecasting, detailing the complexity of the proposed models, their strengths and weaknesses, and the problems that may be encountered from them.

3.0 Notation and Definitions

Before proceeding to the model descriptions, we define some of the general notation and definitions in the context of day-ahead price forecasting.

All our datasets are considered **time series data**, which is simply a sequence of observations ordered by time. We denote the day-ahead electricity price series by the random variable Y_t with $n-1$ observed samples $\{y_t\}_{t=1}^{n-1}$, $y_t \in \mathbb{R}$. We use the past observations $\{y_t\}_{t=1}^{n-1}$ to forecast y_n. We refer to Y_t as an *endogenous predictor* because the y_t terms depend on previous values in the system (e.g. on other y_t terms).

We also consider other time series data (e.g. BM prices and wind/demand forecasts) that we do not necessarily need to model but may help improve our forecast of y_n. This kind of data is referred to as *exogenous predictors*, denoted by the random variables X_t^m with observed samples $\{x_t^m\}_{t=1}^{n}$, $m = 1, \ldots, M$ (M predictors).

The key difference between these two types of predictors is that endogenous predictors are assumed to depend on other variables in the system, whereas exogenous predictors are assumed to be independent. Some model variants use only an endogenous predictor (i.e. only take the electricity price as input).

In general, we denote a *one-step ahead* forecast by

$$\hat{y}_t = f\left(y_1, \ldots, y_{t-1},\ x_1^1, \ldots, x_t^1,\ x_1^2, \ldots, x_t^2,\ \ldots, x_1^M, \ldots, x_t^M\right),$$

where $f(\cdot)$ is specified by our forecasting model. Note that we have allowed for a dependency on the future value X_t^m. This is possible for some variables like wind and demand forecasts because they are usually available for the day-ahead prices before the auction closes and so can be used as predictors.

Recall that each DAM auction posts 24 new prices at a time. With this in mind, we want to generate 24 price forecasts, $y_n, \ldots, y_{n+23}$, at a time. It is therefore sometimes more convenient to reindex the variables by day and hour. Hence, for $h = 1, \ldots, 24$, given observed DAM prices $\{y_{d,h}\}_{d=1}^{\mathcal{D}-1}$ and observed predictor values $\{x_{d,h}^m\}_{d=1}^{\mathcal{D}}$, $m = 1, \ldots, M$, we can express our model forecasts $\hat{y}_{d,h}$ for day

d as

$$\hat{y}_{d,h} = f\left(y_{1,1}, \ldots, y_{1,24}, \ldots, y_{d-1,24},\ x^1_{1,1}, \ldots, x^1_{d,24},\ x^2_{1,1}, \ldots, x^2_{d,24}, \ldots, x^M_{1,1}, \ldots, x^M_{d,24}\right).$$

In modelling tasks, it is standard practice to divide the data into training and test sets. The *training set* is used to configure the models (often referred to as 'training the model'), while the *test set* is set to one side until the very end to allow us to measure the performance of the models on unseen data. Some models have *hyperparameters*, which are parameters that cannot be trained and must be specified by the user in advance. In this case, it is common to further divide the training set into a training and validation set (see Section 5.1 of [25] for more on cross-validation), repeatedly training the model on the training set with different hyperparameters and then evaluating performance on the validation set to determine the best hyperparameter.

Because we are dealing with time series data, it is important to ensure that the training set precedes the test set. It would not make sense to generate forecasts for a given day using future values that would be unknown in practice.

3.1 Naive Model (Benchmark)

Benchmarks are an important 'sanity check' tool to determine the minimum level that more complex models should be expected to perform. For example, in the case where a time series is relatively constant with small deviations (e.g. a low volatility white noise process), the sample mean would be a good benchmark.

The naive model (also known as persistence model) is one of the simplest algorithms for predicting time series data and is often used as a benchmark in electricity price forecasting(see Keles et al. [32]; Nogales et al. [17]; Contreras et al. [18]; Conejo et al. [19]).

The naive method uses past values as the forecast values. We therefore define the n-day naive forecast for day d by

$$\hat{y}_{d,h} = y_{d-n,h}.$$

So, a daily (1-day) naive model uses the price for the same hour from the day before and a weekly (7-day) naive model uses the price for the same hour from a week before, and so on.

For this project, we have used 1-day, 7-day and 1-year naive forecasts as benchmarks with which to compare other models.

3.2 Time Series

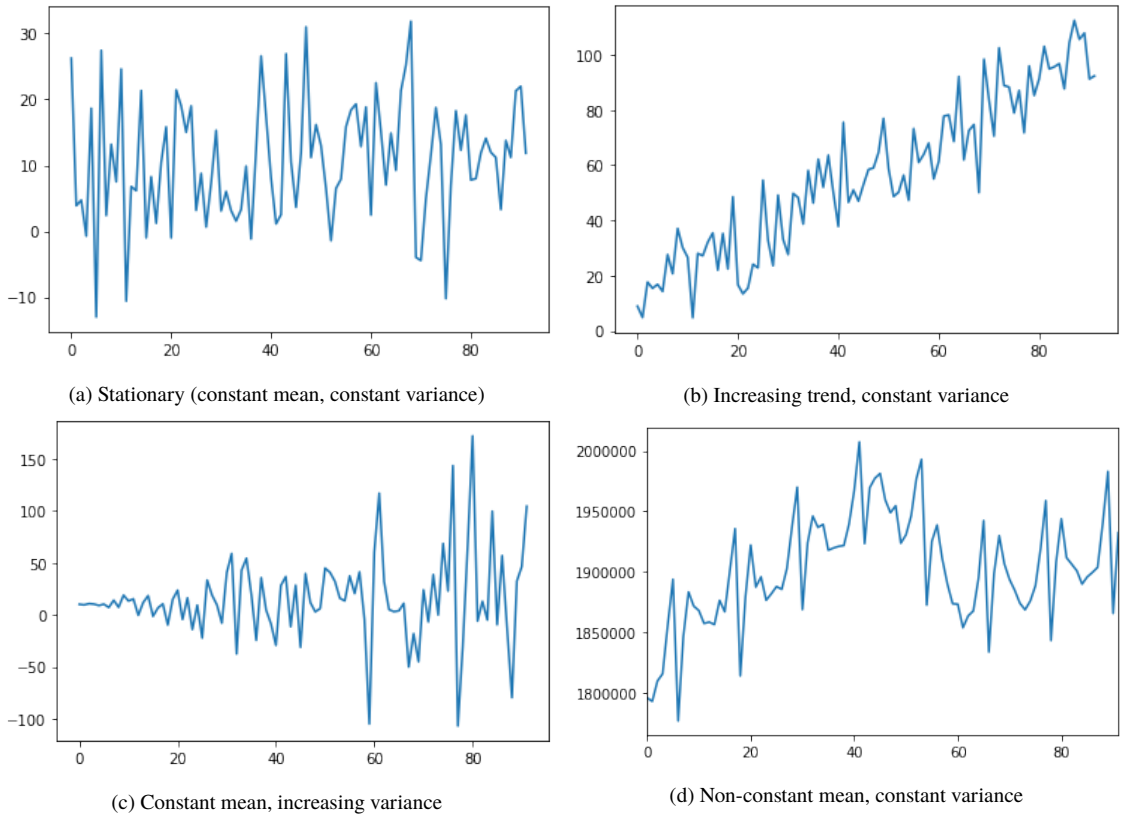
Time series analysis is, as the name suggests, dedicated to modelling time series data. In Section 4, we will consider two models and two variants of each given model (endogenous-only and endogenous + exogenous). Hamilton [20] gives a comprehensive treatise of time series analysis in his 820-page book. For something less overwhelming, see Chatfield [21].

3.2.1 Stationarity

A time series process $\{y_t\}$ is **stationary** if its statistical properties remain unchanged over time (when using the term 'stationary', we are referring to *weak stationarity* which is used more commonly than *strong stationarity*). More particularly, stationarity of $\{y_t\}$ requires that:

- $\mathbb{E}[y_t] = \mu_t = \mu \ \forall \ t$ (constant mean), and that
- $\text{cov}(y_t, y_{t-k}) = \gamma_k$ depends only on $k \in \mathbb{N} \implies \mathbb{V}[y_t] = \text{cov}(y_t, y_t) = \gamma_0$ (constant variance).

It is important to know whether our time series data is already stationary or not before we can think about model selection. A usual first step is to visually check if the data appears stationary (if mean and variance appear constant through time). Figure 5 shows plots of stationary and non-stationary data against time. Only Figure 5a is stationary.



(a) Stationary (constant mean, constant variance)

(b) Increasing trend, constant variance

(c) Constant mean, increasing variance

(d) Non-constant mean, constant variance

Figure 5: Stationary and non-stationary data

Of course, a visual inspection will almost always be insufficient to confirm stationarity, but it gives an idea of the behaviour of the time series. For example, the data in Figure 5a was sampled from a Normal distribution with mean $\mu = 10$ and variance $\sigma^2 = 10^2$, so it is stationary, but one could argue that its variance appears to be greater in the beginning and towards the end. From Section 3.2.2 to 3.2.5, we will assume that $\{y_t\}$ is stationary.

3.2.2 Autocorrelation and Partial Autocorrelation

The **autocorrelation function (ACF)** and **partial autocorrelation function (PACF)** are central to time series analysis. We must be able to estimate the ACF and PACF in order to find a model to fit the sequence $\{y_t\}$. We will now explain what ACF and PACF mean and how they are estimated.

We define $\gamma_k = \text{cov}(y_t, y_{t-k})$, referred to as the *autocovariance at lag k*. This measures the relationship of each electricity price to the price from k hours before (note that $\text{cov}(y_t, y_{t+k}) \equiv \text{cov}(y_t, y_{t-k})$). The correlation between y_t and y_{t-k} is then given by

$$\text{corr}(y_t, y_{t-k}) = \frac{\text{cov}(y_t, y_{t-k})}{\sqrt{\mathbb{V}[y_t]}\sqrt{\mathbb{V}[y_{t-k}]}} = \frac{\gamma_k}{\sqrt{\gamma_0}\sqrt{\gamma_0}} = \frac{\gamma_k}{\gamma_0}.$$

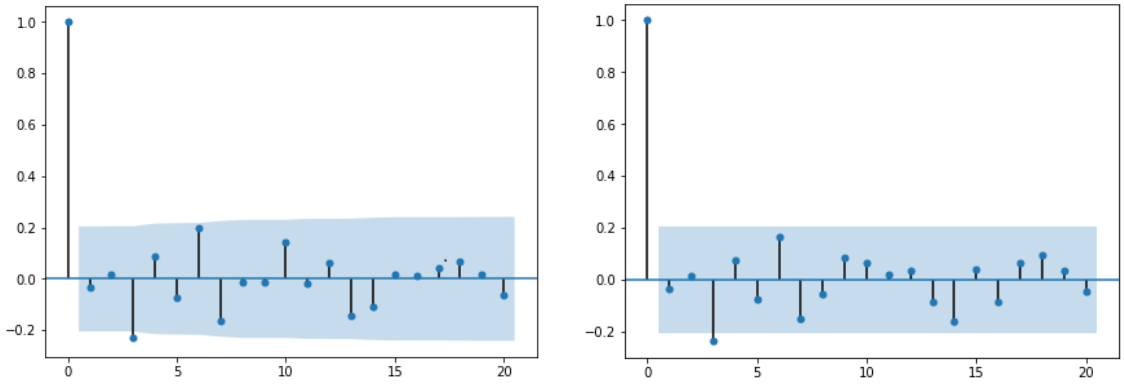
We define $\rho_k = \text{corr}(y_t, y_{t-k})$, referred to as the *autocorrelation at lag k*. The sequence $\{\rho_k : k \geq 0\}$ is the ACF. Note that $\rho_0 = \text{corr}(y_{t-k}, y_{t-k}) = 1$ for all k.

The PACF is defined similarly. Let $\varphi_k = \text{corr}(y_t, y_{t-k} | y_{t-1}, \ldots, y_{t-(k-1)})$, referred to as the *partial autocorrelation at lag k*, i.e.

$$\begin{aligned}
\varphi_0 &= \text{corr}(y_t, y_t) = \rho_0 = 1 \\
\varphi_1 &= \text{corr}(y_t, y_{t-1}) = \rho_1 \\
\varphi_2 &= \text{corr}(y_t, y_{t-2} | y_{t-1}) \\
\varphi_3 &= \text{corr}(y_t, y_{t-3} | y_{t-1}, y_{t-2}) \\
&\vdots \\
\varphi_k &= \text{corr}(y_t, y_{t-k} | y_{t-1}, \ldots, y_{t-(k-1)}),
\end{aligned}$$

so then the sequence $\{\varphi_k : k \geq 0\}$ is the PACF.

Given $\{y_t\}_{t=1}^{n-1}$, we can estimate the sample ACF (SACF) and sample PACF (SPACF) and plot them against the lag values. Figure 6 shows a plot of the SACF and SPACF (also known as the *correlogram* and *partial correlogram*, respectively) of the stationary data from Figure 5a.



(a) Autocorrelation Function

(b) Partial Autocorrelation Function

Figure 6: Correlogram and partial correlogram

The blue shaded region represents the 95% confidence interval (remember that we are calculating statistical estimates of the ACF and PACF). Any values that are inside the region are assumed to be zero with 95% confidence.

Recall how we said that the data was sampled from a normal distribution. The resulting time series process is called a *white noise process* whose behaviour is completely random, and hence the ACF and PACF for lags $k \geq 1$ should be zero. The lag 3 ACF and PACF appears to be outside the confidence interval, but we note that this is a coincidence from the given sample, and given that we are using a 95% CI, we allow for 1 in 20 values to diverge from what we expect. Of course, this is much more difficult in practice since we do not know what the underlying process is beforehand, so it would be difficult to say whether values can be safely disregarded.

We will now introduce two basic linear time series models: the autoregressive (AR) and moving average (MA) models. These models assume that the data is stationary (which is why ensuring stationarity is important). We will later explain how to deal with non-stationarity, but we will assume for these models that the data being used is stationary.

3.2.3 Autoregressive (AR) Model

An autoregressive model of order p is commonly denoted $\mathrm{AR}(p)$ and is defined by the following general equation:

$$y_t = \mu + \theta_1(y_{t-1} - \mu) + \theta_2(y_{t-2} - \mu) + \ldots + \theta_p(y_{t-p} - \mu) + \varepsilon_t, \quad \text{where } \varepsilon_t \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma^2)$$

The set of parameters $\{\theta_i\}_{i=1}^{p}$ are estimated from the data, and μ is the mean of the process (which is constant because the data is assumed to be stationary), and ε_t is zero-mean white noise (which cannot be predicted). To show some basic properties of the AR model, we will assume without loss of generality that $\mu = 0$, and we will look at the simplest one, AR(1), given by:

$$y_t = \theta y_{t-1} + \varepsilon_t, \quad \text{where } \varepsilon_t \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma^2)$$

Notice that 3.2.3 is recursive, so we can rewrite this in terms of y_0, the initial value of the process (which is also a random variable):

$$\begin{aligned} y_t &= \theta y_{t-1} + \varepsilon_t \\ &= \theta\left(\theta y_{t-2} + \varepsilon_{t-1}\right) + \varepsilon_t \\ &= \theta\Big(\theta\big(\theta y_{t-3} + \varepsilon_{t-2}\big) + \varepsilon_{t-1}\Big) + \varepsilon_t \\ &\vdots \\ &= \theta^t y_0 + \sum_{k=0}^{t} \theta^k \varepsilon_{t-k} \end{aligned}$$

It can then be shown that $\lim_{k\to\infty} \rho_k = 0$ with $\rho_k > 0$ (ACF decays to zero). Intuitively, this means that previous values will have a lasting but shrinking effect on subsequent values in the process. It can also be shown that, for an $\mathrm{AR}(p)$ process, $\varphi_k = 0$ for $k > p$ (PACF cuts off to zero after lag p).

Figure 7 shows a simulated AR(2) process and its associated SACF and SPACF plots. We note that the SACF decays and that the SPACF cuts off after lag 2. These two observations would immediately indicate that the data can be modelled by an AR(2) process.

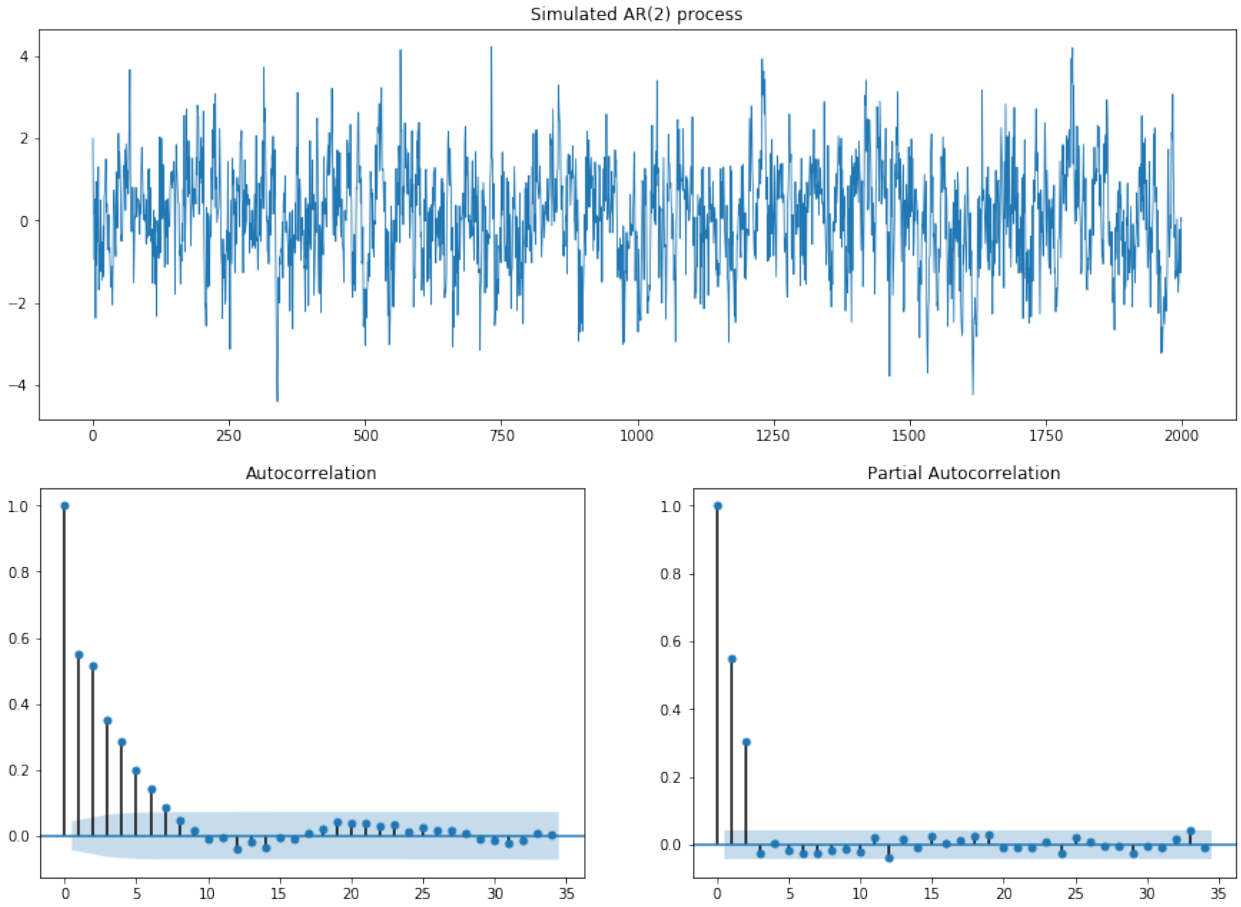


Figure 7: Plot of AR(2) process and corresponding correlogram and partial correlogram

3.2.4 Moving Average (MA) Model

A moving average model of order q is commonly denoted MA(q) and is defined by the following general equation:

$$y_t = \mu + \varepsilon_t + \beta_1 \varepsilon_{t-1} + \beta_2 \varepsilon_{t-2} + \ldots + \beta_q \varepsilon_{t-q}$$

The set of parameters $\{\beta_j\}_{j=1}^{q}$ are estimated from the data, μ is the mean of the process and ε_t is a zero-mean white noise.

For an MA(q) model, it can be shown that $\rho_j = 0$ for $j > q$ (ACF cuts off to zero after lag q). Hence, an MA model would make sense if we know that only some of the most recent values have an effect on subsequent values in the process, unlike the AR model. On the other hand, $\lim_{j\to\infty} \varphi_j = 0$ with $\varphi_j > 0$ (PACF decays to zero).

Figure 8 shows a simulated MA(3) process and its associated SACF and SPACF plots. The SPACF decays, while the SACF cuts off after lag 3, indicating that the data can be described by an MA(3) model.

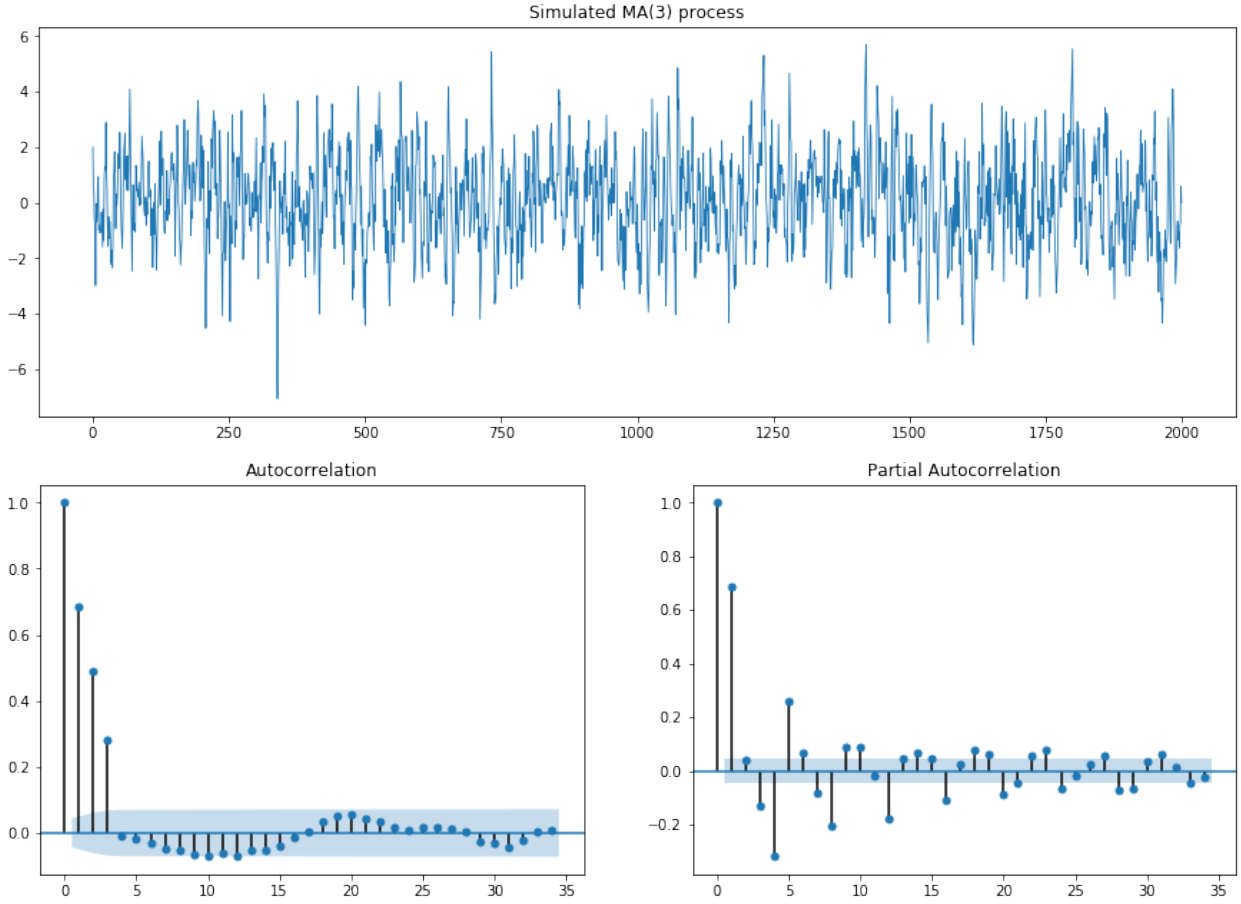


Figure 8: Plot of MA(3) process and corresponding correlogram and partial correlogram

3.2.5 ARMA Model

An ARMA model is simply a combination of an AR and MA component, and its specification is denoted by ARMA(p, q), where p and q correspond to the AR and MA parameters, respectively (ARMA$(p, 0) \equiv$ AR(p); (ARMA$(0, q) \equiv$ MA(q)). Its general equation is given by

$$y_t = \mu + \underbrace{\theta_1(y_{t-1} - \mu) + \ldots + \theta_p(y_{t-p} - \mu)}_{\text{AR}(p)\text{ component}} + \varepsilon_t + \overbrace{\beta_1\varepsilon_{t-1} + \ldots, +\beta_q\varepsilon_{t-q}}^{\text{MA}(q)\text{ component}}.$$

Figure 9 shows the plot of a simulated ARMA(2,3) process. As can be seen, the corresponding correlogram and partial correlogram for ARMA models are less straightforward to interpret, since both will show a decaying behaviour, and so we cannot immediately determine the p and q orders from the plots alone, but we at least know that an ARMA model is needed to describe the data.

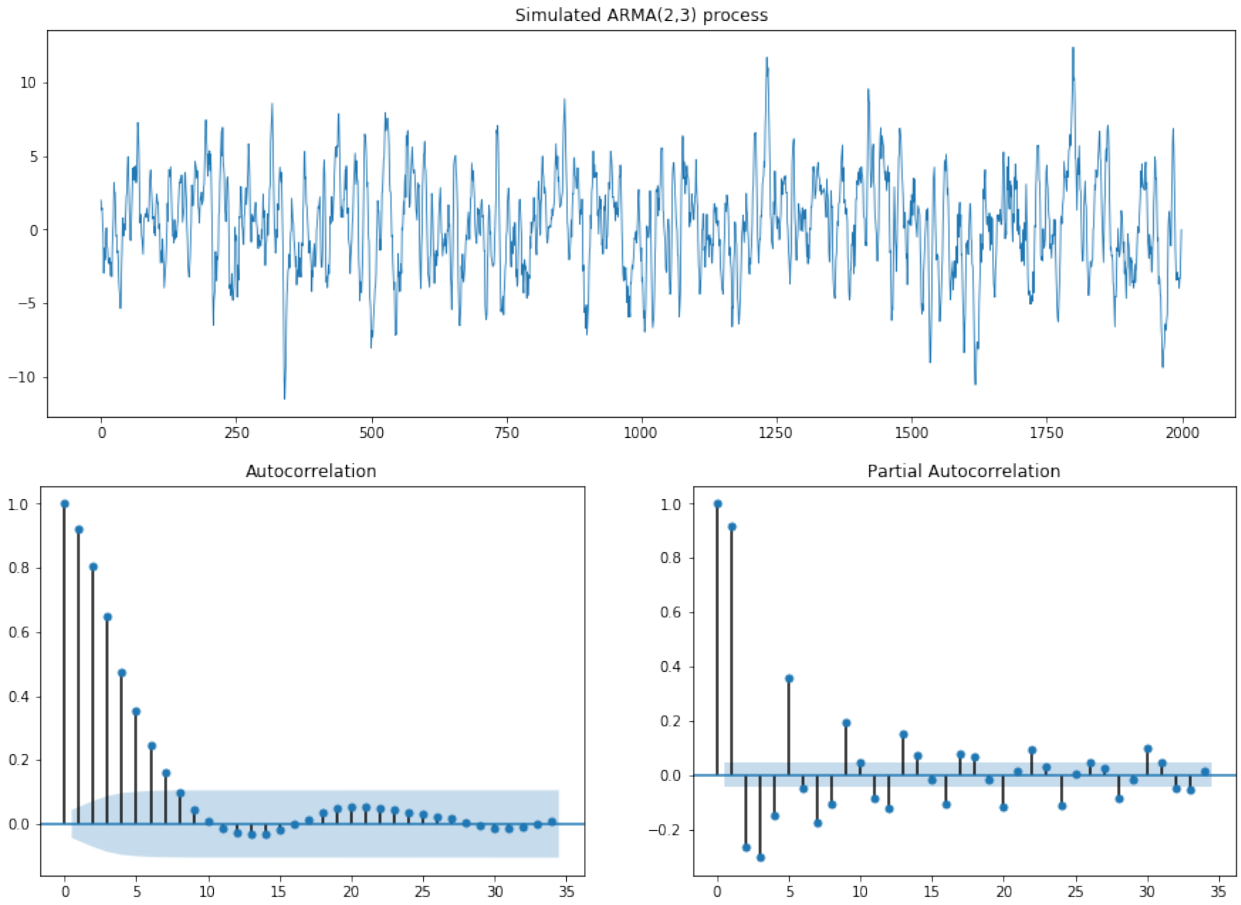


Figure 9: Plot of ARMA(2,3) process and corresponding correlogram and partial correlogram

Different configurations are trained on the data, e.g. ARMA(1,1), ARMA(1,2), ARMA(2,1), ARMA(2,2), . . . , until the best model is found. This points to the idea of *goodness of fit*. Goodness of fit is usually assessed using the AIC and BIC criteria (see Fabozzi et al. [22]), which take into account the number of model parameters. For example, if ARMA(1,1) and ARMA(5,5) perform equally, then ARMA(1,1) should have lower (and better) AIC and BIC values. Note that the same kind of AIC/BIC model evaluation is still commonly done in practice for AR and MA models. The initial ACF/PACF analysis only serves to give an idea of the models that should be considered for the selection process.

3.2.6 ARIMA Model

So far, we have assumed that our data is stationary. In reality, this is rarely the case, and we often have to convert the data to a stationary form. The method we used in our analysis of the electricity price series, and which is the most common, is known as *differencing*.

The difference operator applied to $\{y_t\}_{t=1}^{n-1}$ gives us the first order differences:

$$\nabla y_t = y_t - y_{t-1}, \quad t = 2, \ldots, n-1. \tag{3.1}$$

Then applying it again gives us the second order differences:

$$\begin{aligned}\nabla^2 y_t &= \nabla\left(\nabla y_t\right)\\ &= \nabla y_t - \nabla y_{t-1}\\ &= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2})\\ &= y_t - 2y_{t-1} + y_{t-2}, \quad t = 3, \ldots, n-1,\end{aligned}$$

and so on. If a time series $\{y_t\}_{t=1}^{n-1}$ is not stationary but its first order differences $\{\nabla y_t\}_{t=2}^{n-1}$ are, then we say that $\{y_t\}$ has order of integration 1, i.e. $\{y_t\}$ is I(1). Additionally, if ∇y_t can be modelled by an ARMA(p, q), then we say that y_t can be modelled by an ARIMA$(p, 1, q)$ process. More generally, fitting an ARIMA(p, d, q) is equivalent to fitting an ARMA(p, q) model to the d-order differences.

3.2.7 SARIMA Model

We can now briefly introduce the concept of *seasonality* for time series. When a time series is influenced by the same effects at regular intervals, we say that there is a seasonal effect. For example, sales data tends to have peaks in summer and winter periods, reflecting the increase in sales due to the holidays and people going out more.

The SARIMA (seasonal autoregressive integrated moving average) model is composed of a standard ARIMA component and a seasonal ARIMA component. The latter works much in the same way as the standard ARIMA, but the time steps are determined by the order of seasonality. One can think of a standard ARIMA model as having an order of seasonality 1 because its effects occur at each subsequent time step, whereas the effects of a seasonal ARIMA with order of seasonality S occur every S time steps.

The model specification is often given as SARIMA$(p, d, q) \times (P, D, Q)_S$, where p, d, q are the usual ARIMA orders, P, D, Q are the orders of the seasonal ARIMA component, and S is the order of seasonality. $S = 12$ implies a seasonal effect occurring after every 12 time steps and can be used for monthly data with a yearly seasonality. $S = 24$ implies a seasonal effect occurring after every 24 time steps and can be used for hourly data with daily seasonality.

In the context of an ACF/PACF analysis, seasonality is observed when peaks occur at regular intervals, and these can be 'differenced' away by using a seasonal difference. For example, if there was a daily seasonality for hourly data, we would observe noticeable peaks in the SACF and SPACF at lags $24n$, $n \in \mathbb{N}$, and these peaks should be absent when we observe the SACF/SPACF of the seasonally differenced series $\{\nabla_{24} y_t\}_{t=25}^{n-1}$ given by

$$\nabla_{24} y_t = y_t - y_{t-24}. \tag{3.2}$$

In Section 3.2.9, we will show how to do a simple ACF/PACF analysis to determine a suitable model for the electricity price data.

3.2.8 Time Series with Exogenous Variables

Until now, we have only dealt with endogenous data (where future values of the price are predicted using only previous values of the same price series). Hyndman [23] gives a simple method which

allows the inclusion of other time series data as exogenous predictors. We refer to these exogenous analogues of the ARIMA and SARIMA as ARIMAX and SARIMAX, respectively.

The ARIMAX method, in short, is as follows: Given the usual observed time series $\{y_t\}_{t=1}^{n-1}$ and another times series $\{x_t\}_{t=1}^{n}$ (the exogenous predictor), we first apply simple linear regression:

$$y_t = \alpha x_t + \eta_t, \tag{3.3}$$

and then model the error term η_t with a time series process (e.g. ARMA):

$$\eta_t = \mu + \theta_1(\eta_{t-1} - \mu) + \ldots + \theta_p(\eta_{t-p} - \mu) + \varepsilon_t + \beta_1\varepsilon_{t-1} + \ldots + \beta_q\varepsilon_{t-q}. \tag{3.4}$$

The idea is to use x_t to explain as much of y_t as possible before using a time series model. A similar procedure applies for SARIMAX. It should be clear from (3.3) and (3.4) that this framework could easily be extended to include any number of exogenous predictors.

3.2.9 ACF/PACF Analysis of Electricity Prices

We have explained the concepts of stationarity and non-stationarity, looked at the ACF and PACF and their general features with respect to the different time series models, and have considered the mathematical definition of differencing, but we have yet to show how all of these pieces fit into an ACF/PACF analysis starting from the original time series data and arriving at a (tentative) conclusion on a suitable time series model.

The following ACF/PACF analysis is carried out on the whole period of available data, which includes the data that we will assume to be unknown for each forecast when applying our model validation function (see Section 4.1). In reality, one would do the analysis using only data prior to the forecast date, but we have found the difference to be negligible, indicating that the different levels of seasonality that we will see are consistent throughout the available period.

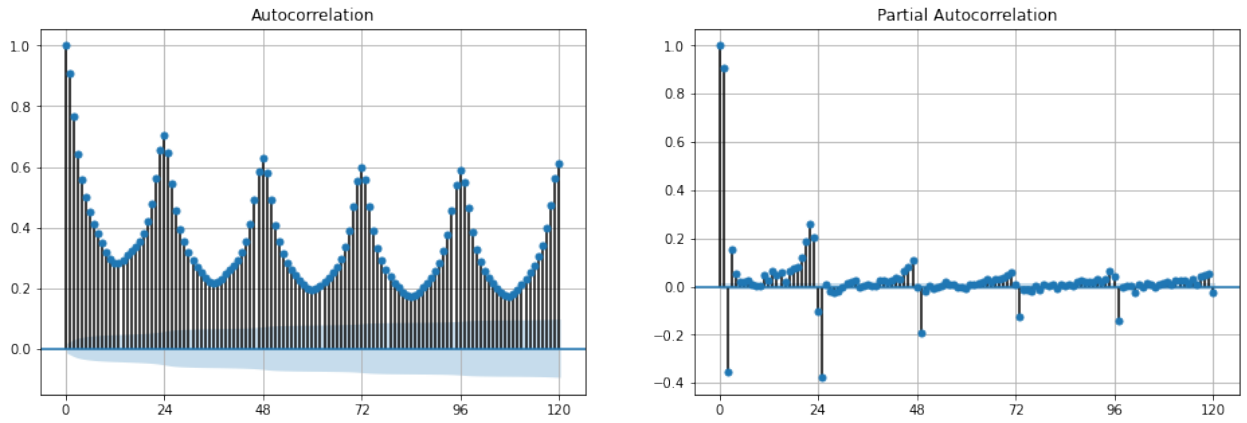


Figure 10: SACF and SPACF up to lag 120 – original DAM prices

Starting with the original time series data given by Figure 1, we obtain SACF and SPACF plots, shown in Figure 10. The general idea here is to apply differencing to reduce as many of the correlations as

possible. The wave pattern of the SACF peaks every 24 steps at lags 0, 24, 48, etc., and we notice a significant spike in the SPACF every 24 steps at lags 1, 25, 49, etc., which indicates a 24-hour seasonality.

So, our next step is to apply a 24-hour seasonal difference, as defined by (3.2). Figure 11 shows the SACF and SPACF plots of the seasonally differenced series. In practice, one seasonal differencing is seen to be sufficient. And in fact, further differencing by 24 periods will actually increase the correlations, so we will not consider it any further.

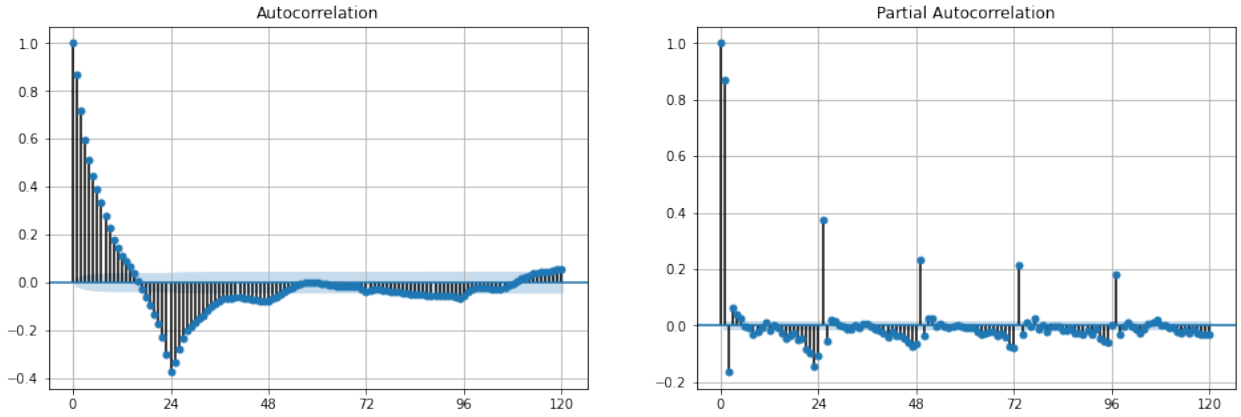


Figure 11: SACF and SPACF up to lag 120 – seasonal difference ($S = 24$)

Having carried out a 24-hour difference, we now need to look at the lags between the $24n$ lags, i.e. between 0 and 24, between 24 and 48, etc. The SACF appears to decay overall, but in between the 24-hour lags, it actually swings between highs and lows, which would indicate that further differencing is required. So, we apply a first-order difference, as defined by (3.1). The corresponding SACF and SPACF is given by Figure 12. Although it appears that there is still some swinging, further differencing no longer reduces correlations, so we will not implement any further differencing and will now begin interpreting the SACF and SPACF for potential models.

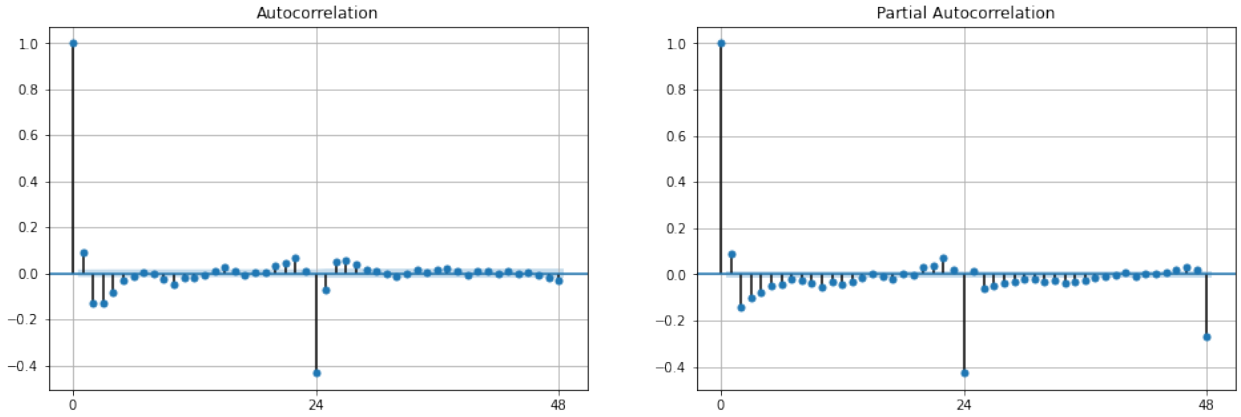


Figure 12: SACF and SPACF up to lag 48 – first-order difference ($d = 1$) and seasonal difference ($S = 24$)

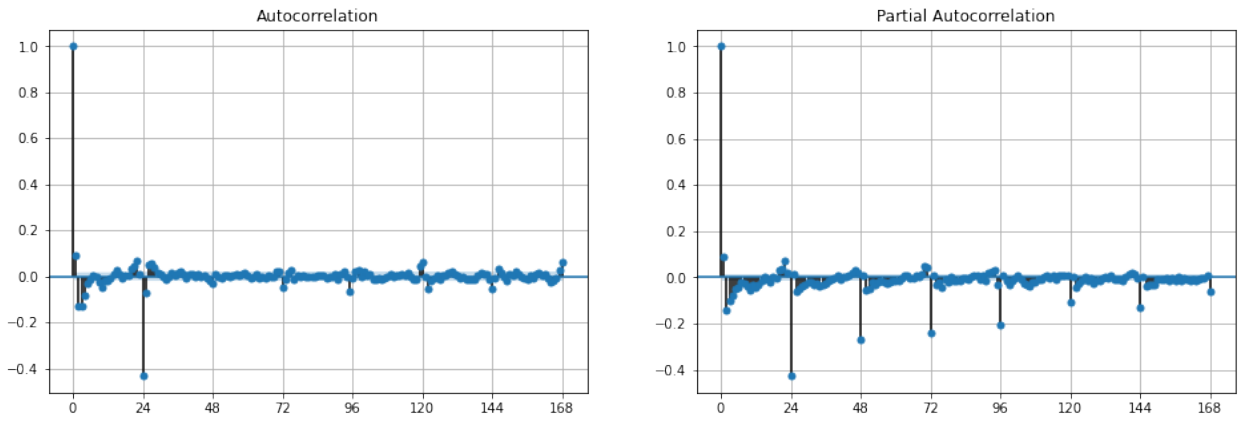


Figure 13: SACF and SPACF up to lag 168 – first-order difference ($d = 1$) and seasonal difference ($S = 24$)

First, the standard ARIMA component. Since we applied first-order differencing, we will consider an ARIMA$(p, 1, q)$. Looking at Figure 12, the pattern of the lags between 0-24 and between 24-48 for the ACF and PACF are difficult to distinguish as either AR (ACF decays to zero and PACF cuts off after lag p) or MA (PACF decays to zero and ACF cuts off after lag q), so we would consider an ARIMA$(p, 1, q)$, with $p, q > 0$. Since the SACF for lags 1-4 seem significantly larger than the subsequent lags, and the SPACF for lags 1-6 seem the most significant, we might consider choosing different order combinations from 4-6, e.g. ARIMA$(4, 1, 4)$, ARIMA$(6, 1, 4)$, ARIMA$(5, 1, 4)$, ARIMA$(6, 1, 5)$, etc. and decide between these using an AIC or BIC criterion.

Now for the seasonal ARIMA component. Figure 13 shows the twice-differenced series over a longer timeframe. As we have said, the interpretation is the same as for the standard ARIMA, but we only look at the lags that are multiples of the order of seasonality, i.e. lag $24n$, $n \in \mathbb{N}$. Firstly, because we applied seasonal differencing, we would consider a SARIMA$(p, d, q) \times (P, 1, Q)_{24}$ This is more straightforward since the SACF seems to cut off after lag 24 and the SPACF decays to zero, indicating a SARIMA$(p, d, q) \times (0, 1, 1)_{24}$. One could argue that there are further significant correlations at lags 72, 96, 120, etc., and it might be worth considering higher values for the seasonal MA parameter Q.

Putting the two components together, we would conclude that the original electricity price series data could be adequately described by, say, a SARIMA$(4, 1, 6) \times (0, 1, 1)_{24}$ or similar. Our final model choice was a SARIMA$(4, 1, 4) \times (0, 1, 1)_{24}$ (see Section 4).

3.3 Random Forests

The random forest model is a favourite first choice for many forecasting and prediction tasks because it is robust and relatively easy to implement. It is often used in top-ranked submissions in Kaggle Data Science competitions[24]. Random forests are a tree-based method (i.e. its main component is the decision tree algorithm), which are explained in good detail in [25].

3.3.1 Decision Trees

Decision trees can be used for both *regression* (real-valued output) and *classification* (categorical output), and in the context of modelling a time series, we use them for regression. We want to model the behaviour of a random variable Y with observed values $\{y_t\}_{t=1}^{n-1}$ using M predictors, $X_1, X_2, \ldots, X_M$, with observed values $\{x_{1,t}\}_{t=1}^{n}, \{x_{2,t}\}_{t=1}^{n}, \ldots, \{x_{M,t}\}_{t=1}^{n}$. Note that each X_m is also a random variable. The aim is to generate a prediction $\hat{y}_n$ of y_n using the predictor values $x_{1,n}, x_{2,n}, \ldots, x_{M,n}$. A decision tree is built as follows:

1. Partition the predictor space $X_1, X_2, \ldots, X_M$ into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$. Each R_j will have the form $\{X_1 \leq t_1, X_2 \leq t_2, \ldots, X_M \leq t_M\}$, where the optimal values $t_m \in \mathbb{R}$, $m = 1, \ldots, M$, are determined from the training data.

2. For observations that fall into the region R_j, the same prediction value is given, usually determined by the mean, $\bar{y}$, of all observations in the training set that fall within R_j.

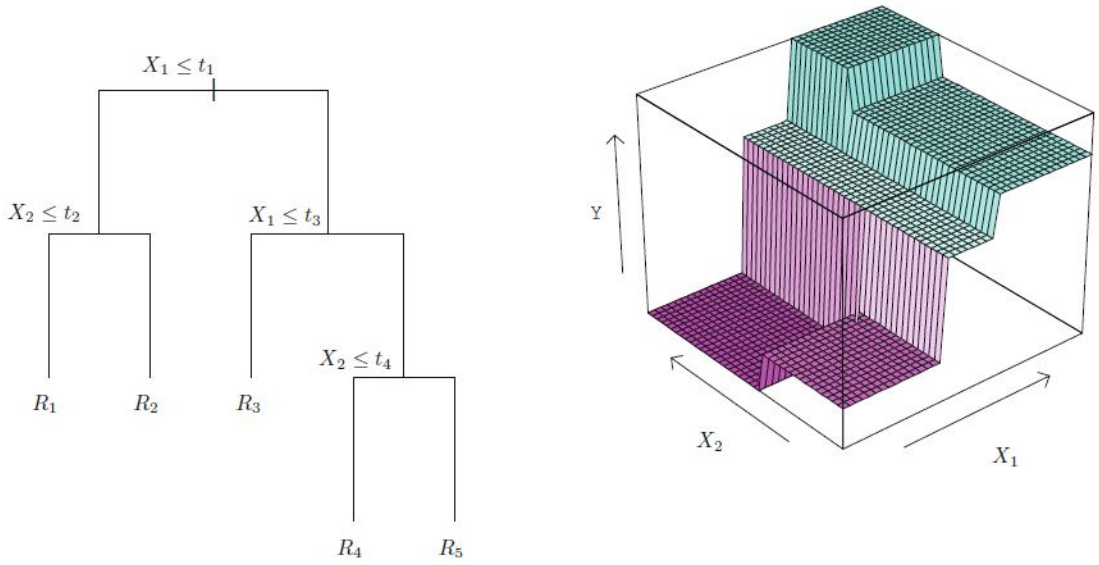


Figure 14: Decision tree for a prediction problem with response variable Y and predictors X_1 and X_2

Figure 14 (taken from [25]) gives a picture of how decision trees would work with two predictors. Note on the left diagram that the splitting of each branch considers only one variable at a time, either X_1 or X_2, and the same variable can be partitioned multiple times. The values at the bottom of the tree (called the *terminal nodes*) represent the region resulting in that partition. In particular, the i^{th} observation will be in region R_1 if its predictor values are $(x_{1,i}, x_{2,i})$ such that $x_{1,i} \leq t_1$ and $x_{2,i} \leq t_2$; the j^{th} observation will be in region R_3 if its predictor values are $(x_{1,j}, x_{2,j})$ such that $x_{1,j} \in (t_1, t_3]$ and $x_{2,j} < \infty$; and so on.

A decision tree is fit to the training set by minimising the residual sum of squares (RSS) given by:

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} \left(y_i - \hat{y}_{R_j} \right)^2, \tag{3.5}$$

where y_i is the observed value in the training sample, and $\hat{y}_{R_j}$ is the predicted value of the decision tree for that particular observation. There are further steps to optimising a decision tree, but we will not mention them here.

3.3.2 Bagging

Decision trees usually suffer from high variance or overfitting, i.e. they can fit the training data well while underperforming on new and unseen data. One solution to this is to use *bagging* (short for *bootstrap aggregation*). Bagging uses multiple learning algorithms to generate a forecast by using the average over all the predictions of the individual trees. This significantly helps to reduce overfitting and increase the stability of the results.

A question might arise as to how the trees get trained differently if they are all being fed the same data. The key idea behind this is bootstrap sampling. Each tree is trained on a bootstrap sample of the original data.

Bootstrapping is the process of resampling from the original data with replacement. Figure 15 (taken from [25]) illustrates how bootstrapping works for a toy example with a sample of $n = 3$ observations and two variables, X and Y. The first bootstrap sample Z^{*1} contains two instances of the third observation, one instance of the first, and none of the second. In this way, the distribution of the original training data is preserved while allowing us to train decision trees differently using the same data.

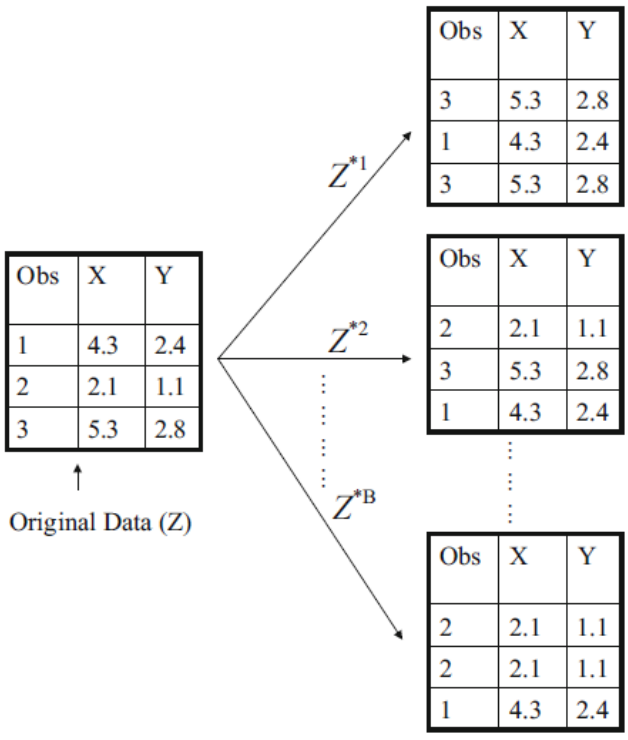


Figure 15: A graphical illustration of bootstrapping for a sample containing $n = 3$ observations and two variables X and Y.

3.3.3 Random Forests

Although bagging of decision trees presents a big improvement over using individual trees, there are still shortcomings in the model. A main problem with bagging is that the component trees tend to be highly correlated since the same M predictors are being considered at every split in the tree building process. This means that strong predictors (predictors that reduce the RSS significantly when partitioned) would be favoured over weaker predictors.

Random forests provide a solution to this problem. They work in very much the same way as bagging but considering only a subset $m < M$ of the predictors at each split of a tree. This serves to decorrelate the trees within the ensemble.

Note that bagging can be used with different algorithms other than decision trees but that random forests are specific to decision trees since it is designed to be an improvement on bagging with trees.

In the context of our time series modelling, each hourly electricity price is one observation/sample and, just as with time series models, we can consider endogenous predictors (i.e. lagged values of the electricity prices) as well as exogenous predictors (e.g. wind and demand forecasts and their lags).

3.3.4 Variable Importance

When using ensemble methods such as bagging or random forests, we gain an improvement in accuracy at the cost of model interpretability (as was the case with decision trees), since each forecast is an aggregation of dozens or hundreds of decision tree forecasts. An alternative method of determining feature importances in such methods involves using the RSS, given by (3.5). We record the mean reduction in the RSS due to splits over a given predictor, averaged over all B trees. See Section 4.3.4 for an analysis of variable importances.

3.4 X-Model

The X-Model is a novel approach by Ziel and Steinert [26] designed specifically for electricity price forecasting. This method sets itself apart from other models in its approach to the data. Many algorithms in electricity price series forecasting directly model the electricity price. On the other hand, the X-Model tries to capture the behaviour of the underlying demand and supply curves of the DAM auctions and then infer the corresponding prices from the forecasted curves, thus utilising more of the data that is publicly available and making use of the particular structure of the electricity market.

3.4.1 Law of Supply and Demand

In economics, the law of supply and demand refers to the relationship between the *price* of a product or resource (in this case, electricity) and the *quantity* of that resource that is bought and sold in a market. The quantity bought is described by the *demand curve* (sometimes called purchase or bid curve) while the quantity sold is described by the *supply curve* (sometimes called sale or sell curve). We will restrict our attention to these curves in the context of the DAM auction.

These curves are theoretical and are often a simplified version of the real market dynamics. For example, there are different order types (see Section 4.3.7 of [27]), but we make the assumption that all buy (sell) orders are 'simple' and are filled as soon as a matching sell (buy) order is received. Another simplification is in the way we build these curves from real auction data. The curves are assumed to be continuous, but we only have a limited set of data points for each auction, so we simply assume a linear relationship between each subsequent point. Figure 16 shows the price curves for the 12$^{\text{th}}$ November 2018. Each point corresponds to an individual order.

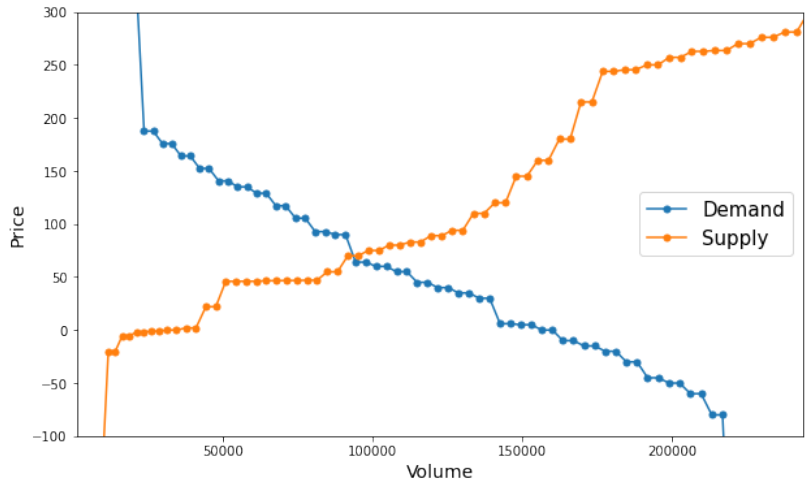


Figure 16: Constructed supply and demand curve for one-hour electricity delivery from 11 pm on the 12$^{\text{th}}$ November 2018

In the available data, each order (buy or sell) comes in price-quantity pairs. Participants submit one or more orders, each for a certain quantity in megawatt hours (MWh) and a corresponding price (in €/MWh). The buy (sell) orders are sorted in decreasing (increasing) order of price and then a cumulative sum of the volume is calculated. This cumulative sum is the demand (supply) curve for the corresponding auction data for that delivery hour. Mathematically, the supply and demand curves in Figure 16 for day d and hour h can be defined by the following functions, respectively:

$$S_{d,h}(P) = \sum_{\substack{p \in \mathbb{P}_{S,d,h} \\ p \leq P}} V_{S,d,h}(p) \text{ for } P \in \mathbb{P}_{S,d,h} \quad \text{and} \quad D_{d,h}(P) = \sum_{\substack{p \in \mathbb{P}_{D,d,h} \\ p \geq P}} V_{D,d,h}(p) \text{ for } P \in \mathbb{P}_{D,d,h}, \tag{3.6}$$

where $\mathbb{P}_{S,d,h}$ and $\mathbb{P}_{D,d,h}$ are the set of prices that had bids with non-zero volume for day d and hour h (S for supply and D for demand). $V_{S,d,h}$ and $V_{D,d,h}$ are the corresponding volumes. Note that even though the curves are a function of the price P, the common convention is to present the price on the y-axis and the volume on the x-axis.

Left to itself, the market should settle at a given price, called the *equilibrium price*. In the absence of external influences, this will usually be the intersection of the two curves, but in the presence of such effects (e.g. subsidies, tariffs, taxes, collusion etc.), the equilibrium price may not always be the intersection. However, because of the unique requirement for electricity supply and demand to always

be perfectly balanced, and because this 'balancing' is being done explicitly by the EUPHEMIA algorithm, it is not unreasonable to make the assumption that the intersection price will be the final price.

The key idea behind the X-Model is to model the components $V_{S,d,h}(p)$, defined on $\mathbb{P} = \{-500.0, -499.9, -499.8, \ldots, 2999.8, 2999.9, 3000.0\}$ (more on this later), of the underlying supply and demand curves, forecasting these components and then reconstructing the curves to obtain a forecasted electricity price from the intersection of the reconstructed curves.

Having explained the concept of supply and demand, we now present a summary of the modelling steps taken in the paper (with several simplifications), culminating in 24 day-ahead price forecasts.

3.4.2 Methodology

1. Price classes for bids

DAM auction bid prices in the data set fall in the range $[-500, 3000]$, and in steps of size 0.1, so in total, there are 35,001 possible prices, and we define this set of possible prices as $\mathbb{P} = \{-500.0, -499.9, -499.8, \ldots, 2999.8, 2999.9, 3000.0\}$. Theoretically, one would model the bid volume of each of these prices in time and then build the corresponding price curves. However, this is not computationally feasible, so a simple dimension reduction technique is used, grouping the data into *price classes*.

We will create a different set of price classes for each curve because the concentration of bids is different between the two (i.e. consumers prefer low prices while suppliers prefer high prices). To start, we calculate the mean bid volumes

$$\overline{V_S}(P) = \frac{1}{T}\sum_{t=1}^{T} V_{S,t}(P) \quad \text{and} \quad \overline{V_D}(P) = \frac{1}{T}\sum_{t=1}^{T} V_{D,t}(P)$$

for supply and demand for a given price $P \in \mathbb{P}$, where T is the total number of hourly observations. This gives a general measure of the importance of a price P within the given price curve. Note that only the non-zero bid volumes incur a computational expense in the sum calculations. Since there are usually only a few dozen bids in total at each time t, then most of the 35,001 bid prices will have zero volume.

Now define $\mathbb{P}_{S,t}$ and $\mathbb{P}_{D,t}$ as the set of sell and buy prices, respectively, that had bids with non-zero volume at time t. We then construct the *mean price curves* using a similar formula to (3.6):

$$\overline{S}(P) = \sum_{\substack{p\in\mathbb{P}_S \\ p\le P}} \overline{V_S}(p) \text{ for } P \in \mathbb{P}_S \quad \text{and} \quad \overline{D}(P) = \sum_{\substack{p\in\mathbb{P}_D \\ p\ge P}} \overline{V_D}(p) \text{ for } P \in \mathbb{P}_D, \tag{3.7}$$

where $\mathbb{P}_S = \cup_{t=1}^{T}\mathbb{P}_{S,t}$ and $\mathbb{P}_D = \cup_{t=1}^{T}\mathbb{P}_{D,t}$ are the set of sell and buy prices, respectively, that had non-zero volume for the whole period (i.e. at least one bid order was placed for each given price at some point during the period).

Next, we choose a volume amount V_* by which to partition the mean price curves. We then define a grid $\mathbb{V}_* = \{iV_* | i \in \mathbb{N}\}$ to partition the price curves where each class is determined by the volume

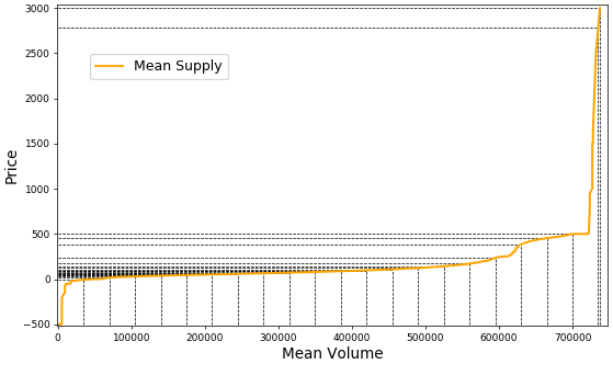
upper bound of each interval, represented by the vertical lines in Figure 17. Using $\mathbb{V}_*$ and the inverse supply and demand functions $\overline{S}^{-1}$ and $\overline{D}^{-1}$, we can define the price class bounds by

$$\mathbb{C}_S = \overline{S}^{-1}(\mathbb{V}_*) = \{\overline{S}^{-1}(iV_*)|i \in \mathbb{N}\} \quad \text{and} \quad \mathbb{C}_D = \overline{D}^{-1}(\mathbb{V}_*) = \{\overline{D}^{-1}(iV_*)|i \in \mathbb{N}\}.$$

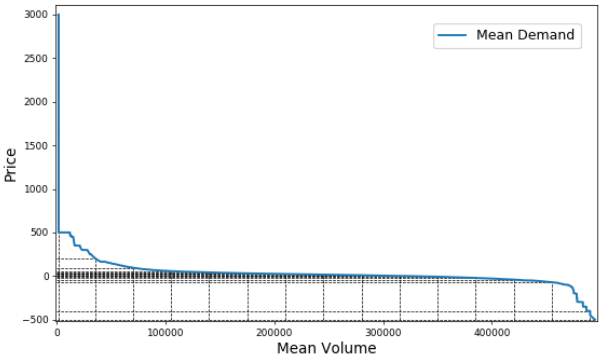
The resulting price classes for $V_* = 35{,}000$ for the Irish bid data are given in Table 1, and the bounds are represented by the horizontal lines in Figure 17. In total, there are 23 supply price classes and 16 demand price classes. Note that the choice of V_* is arbitrary, and in general a lower value will result in more price classes while a higher value will give fewer price classes.

$\mathbb{C}_S$	-500, -6.0, 15.6, 29.4, 38.0, 45.0, 51.31, 58.17, 65.0, 71.6, 79.3, 87.7, 97.1, 108.3, 122.5, 142.1, 171.0, 234.1, 380.24, 453.0, 498.9, 2781.4, 3000
$\mathbb{C}_D$	3000, 204.0, 95.2, 56.1, 41.1, 31.0, 23.4, 16.0, 9.1, 0.8, -8.5, -21.6, -41.5, -70.0, -400.9, -500

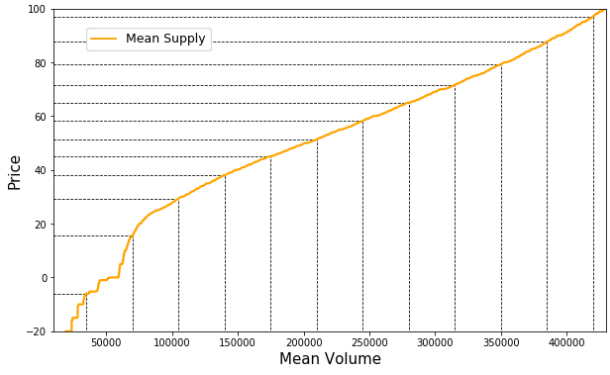
Table 1: Price class bounds, $\mathbb{C}_S$ and $\mathbb{C}_D$, for supply and demand



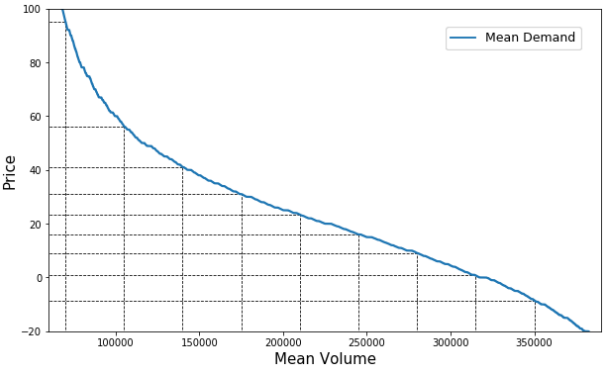
(a) Mean supply on price range $[-500, 3000]$



(b) Mean demand on price range $[-500, 3000]$



(c) Mean supply on price range $[-20, 100]$



(d) Mean demand on price range $[-20, 100]$

Figure 17: Mean supply and demand curves, $\overline{S}$ and $\overline{D}$, with volume grid $\mathbb{V}_*$ and price class bounds $\mathbb{C}_S$ and $\mathbb{C}_D$ for $V_* = 35{,}000$.

The supply price classes are defined by their upper bound and the demand price classes by their

lower bound. Moreover, the volumes at time t associated to the price classes $\mathbb{C}_S$ and $\mathbb{C}_D$ are given by

$$X_{S,t}^{(c)} = \sum_{P \in \mathbb{P}_S(c)} V_{S,t}(P) \quad \text{for } c \in \mathbb{C}_S, \tag{3.8}$$

$$X_{D,t}^{(c)} = \sum_{P \in \mathbb{P}_D(c)} V_{D,t}(P) \quad \text{for } c \in \mathbb{C}_D, \tag{3.9}$$

where $\mathbb{P}_S(c)$ and $\mathbb{P}_D(c)$ are the set of prices in the price class defined by c in $\mathbb{C}_S$ and $\mathbb{C}_D$. So instead of having to model the volumes for 35,001 price points, we now only need to model the volumes for $M_S = 23$ supply price classes and $M_D = 16$ demand price classes.

2. Time series model for bid classes

The classes are modelled by a simple linear regression approach, where each price class is allowed to depend on values of the same class and of other price classes from previous days. For each delivery hour, we have $M_S + M_D = 23 + 16 = 39$ classes to model, and hence 39 models. With 24 hours in a day, this means that every 24-step ahead forecast involves fitting $39 \times 24 = 936$ models.

We also use dummy variables (i.e. numeric variables that represent categorical data) to allow for effects that occur with respect to the day of the week since electricity supply and demand are known to have a different structure each day, especially between weekdays and weekends. We define six weekday dummy variables by

$$W_k(d) = \begin{cases} 1, & \mathcal{W}(d) \leq k \\ 0, & \mathcal{W}(d) > k, \end{cases}$$

where $\mathcal{W}(d)$ is a function that assigns a number (from 0 to 6) for the day of the week corresponding to day d. We use WLOG $k = 0$ for a Monday, $k = 1$ for a Tuesday, up to $k = 5$ for a Saturday. So then we have a total of 6 dummy variables for 7 days in the week, and

$$\begin{aligned} \mathbb{W} &= \{W_0(d), W_1(d), W_2(d), W_3(d), W_4(d), W_5(d)\} \\ &= \begin{cases} \{1,1,1,1,1,1\}, & \text{for } d \equiv \text{Monday} \\ \{0,1,1,1,1,1\}, & \text{for } d \equiv \text{Tuesday} \\ \{0,0,1,1,1,1\}, & \text{for } d \equiv \text{Wednesday} \\ \quad\vdots & \quad\vdots \\ \{0,0,0,0,0,0\}, & \text{for } d \equiv \text{Sunday}. \end{cases} \end{aligned}$$

To fully present the considered time series model, we introduce the following object:

$$\mathbf{X}_{d,h} = \left(X_{1,d,h}, \ldots, X_{M,d,h}\right)',$$

where $M = M_S + M_D + M_X$ corresponds to the total number of price classes and exogenous predictors. Note that we are only modelling the $M_S + M_D$ components. We normalise each element of $\mathbf{X}_{d,h}$ because the linear regression model uses regularisation which requires scaling (more on this later). So, we are actually modelling the process $\mathbf{Y}_{d,h}$ given by

$$\mathbf{Y}_{d,h} = \frac{\mathbf{X}_{d,h} - \mu_h}{\sigma_h}, \tag{3.10}$$

where $\mu_h = \mathbb{E}\left[\mathbf{X}_{d,h}\right]$ and $\sigma_h^2 = \mathbb{V}\left[\mathbf{X}_{d,h}\right]$. The proposed time series model for $Y_{m,d,h}$ for each hour h and $m \in \{1, \ldots, M_S + M_D\}$ is given by

$$Y_{m,d,h} = \sum_{l=1}^{M} \sum_{j=1}^{24} \sum_{k \in \mathcal{I}_{m,h}(l,j)} \phi_{m,h,l,j,k} Y_{l,d-k,j} + \sum_{k=2}^{7} \varphi_{m,h,k} W_k(d) + \varepsilon_{m,d,h}, \tag{3.11}$$

where $\phi_{m,h,l,j,k}$ and $\varphi_{m,h,k}$ are the parameters that have to be estimated from the data and $\varepsilon_{m,d,h}$ is the error term. $\mathcal{I}_{m,h}(l,j)$ is a set of lags that determine the dependency of $Y_{m,d,h}$ on previous values of all the other classes (including itself). We have followed the same choice of $\mathcal{I}_{m,h}(l,j)$ as in the original paper [26]:

$$\mathcal{I}_{m,h}(l,j) = \begin{cases} \{1,2,\ldots,36\}, & m = l \text{ and } h = j \\ \{1,2,\ldots,8\}, & (m = l \text{ and } h \neq j) \text{ or } (m \neq l \text{ and } h = j) \\ \{1\}, & m \neq l \text{ and } h \neq j. \end{cases} \tag{3.12}$$

Note that these only indicate possible dependencies, and the coefficients could be zero (i.e. no dependency on that specific predictor). Figure 18 (taken from [26]) shows the possible dependency structure when $\mathcal{I}_{m,h}(l,j)$ is defined by (3.12). The target forecast is hour 2 of class m of day d, and its value dependency on previous hour 2 values of the same class goes as far back as 36 days, whereas its dependency on other hours within the same price class only goes as far back as 8 days. Likewise, its dependency on hour 2 of different classes goes back 8 days. For values that come from different hours and different classes, we only go back 1 day. Note that the planned classes in Figure 18 are the wind and demand forecasts so the target price class is allowed to depend on the day d values because they are known in advance.

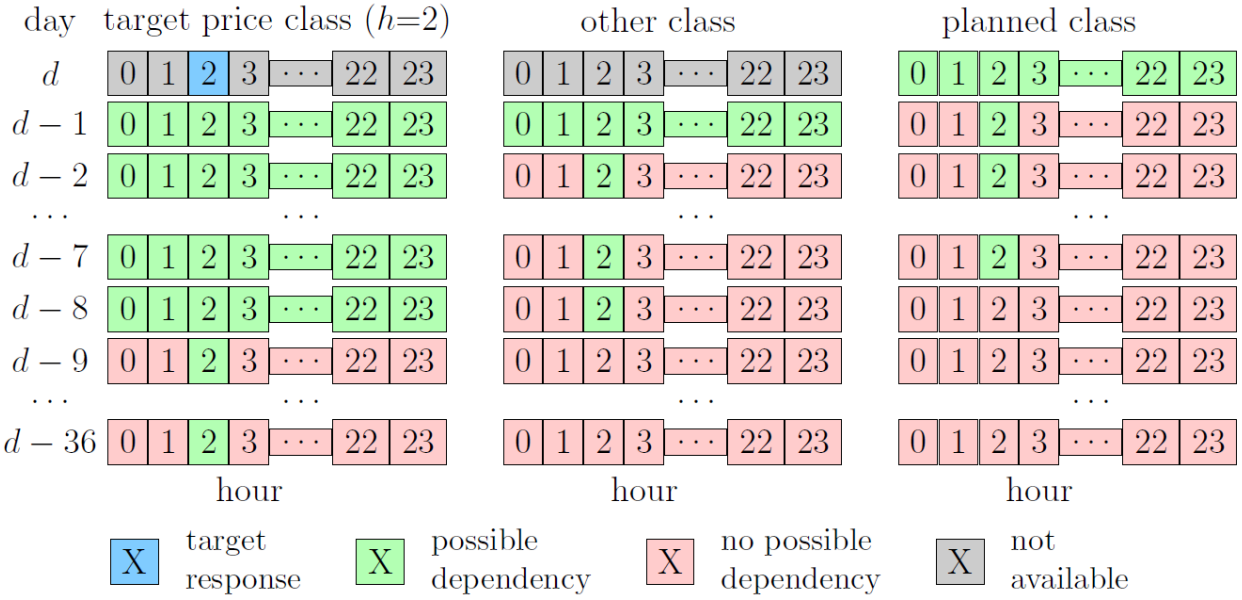


Figure 18: Illustration of the dependency structure for a target bid volume process of a price class at hour 2.

As previously mentioned, the linear model uses a regularization procedure to estimate the parameters. In this context, *regularization* is the process of reducing the likelihood of overfitting by incorporating additional information when minimising the objective function. More specifically, we use the lasso method (see Tibshirani [28]), which works as follows. Define $\beta_{m,h}$ as the set of parameters and $\mathbb{X}_{m,d,h}$ as the set of predictors for $Y_{m,d,h}$ from (3.11), given by

$$\begin{aligned}\beta_{m,h} &= \left[\beta_{m,h,1}, \beta_{m,h,2}, \ldots, \beta_{m,h,p_{m,h}}\right]^\top, \\ \mathbb{X}_{m,d,h} &= \left[\mathbb{X}_{m,d,h,1}, \mathbb{X}_{m,d,h,2}, \ldots, \mathbb{X}_{m,d,h,p_{m,h}}\right].\end{aligned}$$

Note that the parameters are indexed until $p_{m,h}$ instead of just p to indicate that the specific set of predictors for each price class is different, e.g. price class m will have more dependencies on price class m (on itself) than on any other price class. Then we can write the (multivariate) ordinary least squares representation of (3.11) as

$$Y_{m,d,h} = \mathbb{X}_{m,d,h}\beta_{m,h} + \varepsilon_{m,d,h}.$$

Given $\mathcal{D} - 1$ observable days of data, we estimate the parameter vector $\beta_{m,h}$ using the lasso estimator $\hat{\beta}_{m,h}$:

$$\hat{\beta}_{m,h} = \underset{\beta \in \mathbb{R}^{p_{m,h}}}{\arg\min} \left[\sum_{d=1}^{\mathcal{D}-1} \left(Y_{m,d,h} - \mathbb{X}_{m,d,h}\beta\right)^2 + \lambda_{m,h} \sum_{j=1}^{p_{m,h}} |\beta_{m,h,j}| \right]. \tag{3.13}$$

The first sum is the usual least-squares term for estimating the parameters for a simple linear regression. The second sum is part of the lasso procedure, where $\lambda_{m,h}$ is the penalty parameter. This serves as a variable selection method because if a given variable does not significantly improve the forecast, then it gets 'penalised' in the form of having a lower (or zero) parameter estimate for $\beta_{m,h,j}$ because the model is 'better off' without that parameter. In fact, most of the parameters end up being reduced to zero so that even though we allow for more than 1,000 variables, the resulting linear models (recall that we have 936 of them) have only a few parameters (less than a dozen).

Once we fit each linear model, we can finally generate our 24-hour day-ahead forecast. For each hour, we generate a total of $M_S + M_D$ price class forecasts for day $\mathcal{D}$, each given by

$$\hat{Y}_{m,\mathcal{D},h} = \sum_{l=1}^{M} \sum_{j=1}^{24} \sum_{k \in \mathcal{I}_{m,h}(l,j)} \hat{\phi}_{m,h,l,j,k} Y_{l,d-k,j} + \sum_{k=2}^{7} \hat{\varphi}_{m,h,k} W_k(d),$$

where $\hat{\phi}_{m,h,l,j,k}$ and $\hat{\varphi}_{m,h,k}$ are estimated from the data using (3.13). Finally, recall that we normalised the original time series data $Y_{d,h}$ using (3.10). Doing this speeds up the estimation procedure and allows us to use the parameter estimates to measure variable importance, i.e. which variables have the most predictive capability. We apply an inverse transformation to $\hat{Y}_{m,\mathcal{D},h}$ to get the day-ahead volume forecasts $\hat{X}_{m,\mathcal{D},h}$ for the price classes.

3. Reconstructing bid orders and price curves

After computing the bid volume forecast $\hat{X}_{m,\mathcal{D},h}$ for each class $m \in \{1, \ldots, M_S + M_D\}$ and hour h, we now have to split these into individual buy/sell bid orders. To do this, we model the probability of each price point having a non-zero volume (i.e. that an order was placed for that price). We

then go back to the previous price class notation: $X^m_{\mathcal{D},h} \to X^{(c)}_{S,\mathcal{D},h}$ for supply price classes and $X^m_{\mathcal{D},h} \to X^{(c)}_{D,\mathcal{D},h}$ for demand price classes. Similar to (3.8) and (3.9), we are assuming for the forecasted bid volumes that

$$\hat{X}^{(c)}_{S,\mathcal{D},h} = \sum_{P \in \mathbb{P}_S(c)} V_{S,\mathcal{D},h}(P) \quad \text{for } c \in \mathbb{C}_S,$$

$$\hat{X}^{(c)}_{D,\mathcal{D},h} = \sum_{P \in \mathbb{P}_D(c)} V_{D,\mathcal{D},h}(P) \quad \text{for } c \in \mathbb{C}_D,$$

and our goal is to estimate $V_{S,\mathcal{D},h}(P)$ and $V_{D,\mathcal{D},h}(P)$ for all $P \in \mathbb{P}$. Let $\pi_{S,\mathcal{D},h}(P)$ and $\pi_{D,\mathcal{D},h}(P)$ be the probabilities that $V_{S,\mathcal{D},h}(P)$ and $V_{D,\mathcal{D},h}(P)$ respectively are greater than zero. We assume that these probabilities are constant over time and estimate them by the relative frequencies $\hat{\pi}_{S,\mathcal{D},h}(P)$ and $\hat{\pi}_{D,\mathcal{D},h}(P)$ in the given training sample. We also assume proportionality between the prices with respect to the mean volume $\overline{V}_S(P)$ and $\overline{V}_D(P)$. Then we estimate the reconstructed volumes by

$$\breve{V}_{S,\mathcal{D},h}(P) = \frac{R_S(P)\overline{V}_S(P)}{\sum_{Q \in \mathbb{P}_S(c)} R_S(Q)\overline{V}_S(Q)} X^{(c)}_{S,\mathcal{D},h}$$

$$\breve{V}_{D,\mathcal{D},h}(P) = \frac{R_D(P)\overline{V}_D(P)}{\sum_{Q \in \mathbb{P}_D(c)} R_D(Q)\overline{V}_D(Q)} X^{(c)}_{D,\mathcal{D},h},$$

where c is the price class of $\mathbb{C}_S$ or $\mathbb{C}_D$ associated with price $P \in \mathbb{P}$ and $R_S(P) \sim \text{Ber}(\pi_{S,\mathcal{D},h}(P))$ and $R_D(P) \sim \text{Ber}(\pi_{D,\mathcal{D},h}(P))$ are Bernoulli random variables with estimated probabilities $\hat{\pi}_{S,\mathcal{D},h}(P)$ and $\hat{\pi}_{D,\mathcal{D},h}(P)$. For making point forecasts (as opposed to probabilistic forecasts, which are done in the paper but are excluded here), we set $R_S(P)$ and $R_D(P)$ to 1 if the corresponding probability is above a certain threshold, and 0 otherwise. In the paper, they used 1/12 as a threshold, but we note that this is dependent on the data, and we used 1/96 to get a similar distribution of probabilities. The higher the value is, the more likely the price class volume will be distributed across the prices within each price class.

Once we have our estimates $\breve{V}_{S,\mathcal{D},h}(P)$ and $\breve{V}_{D,\mathcal{D},h}(P)$, we can now build the corresponding price curves as before, giving us

$$\breve{S}_{\mathcal{D},h}(P) = \sum_{\substack{pf \in \mathbb{P}_{S,\mathcal{D},h} \\ p \le P}} \breve{V}_{S,\mathcal{D},h}(p) \quad \text{and} \quad \breve{D}_{\mathcal{D},h}(P) = \sum_{\substack{p \in \mathbb{P}_{D,\mathcal{D},h} \\ p \ge P}} \breve{V}_{D,\mathcal{D},h}(p).$$

The intersection of the reconstructed supply and demand curves $\breve{S}_{\mathcal{D},h}$ and $\breve{D}_{\mathcal{D},h}$ then gives us the corresponding electricity price forecast for hour h of day $\mathcal{D}$.

3.5 Neural Networks

Artificial neural networks, or neural networks, are a branch of machine learning which have proved very successful in solving a large variety of problems, including image recognition, speech recognition, natural language processing, targeted advertising, fraud detection, and so on. This class

of methods (very loosely) derives its name from the biological structure of the human brain, i.e. several 'neurons' (from a few dozen to a few million) that are linked together by a simple set of mathematical operations, resulting in a structure that can model complex patterns.

Francois Chollet [31] gives a detailed explanation of how neural networks operate, directed more towards data scientists and software engineers (for example, most of the explanations are given using Python code snippets rather than with mathematical equations). The code examples make use of the Keras API (`keras`), a Python deep learning library [29]. For a more mathematical perspective, Higham [30] gives a brief summary of neural networks and provides an example of using convolutional neural networks for image classification, which is the task of predicting the content of an image from a list of possible choices.

In the price forecasting literature, Keles et al. [32] give a good explanation of how to apply neural networks to electricity price forecasting.

Once again, many key points are omitted, and we only present the subject areas that are immediately relevant to our use in the context of electricity price series forecasting. The models we will consider are the feedforward neural network and recurrent neural network.

3.5.1 Feedforward Neural Networks

For those familiar with graph theory terminology, a feedforward neural network (FFNN) is a fully connected, directed, multipartite network. Figure 19 (taken from [30]) gives a representation of a FFNN with four layers. Each FFNN has an input layer (layer 1), where values are passed through, and an output layer (layer 4), where the final output values are given by the model. Any other layers between the input and output layer are referred to as *hidden layers*.

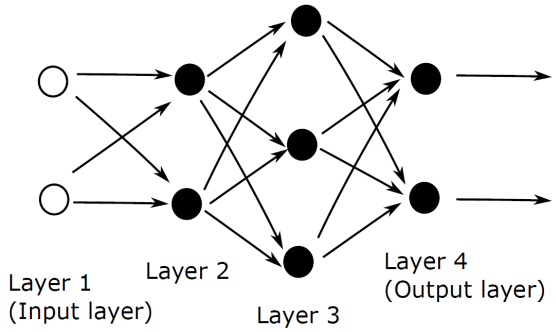


Figure 19: A feedforward network with two hidden layers

Each layer consists of a set of neurons each holding a real number. The number of neurons in the input layer corresponds to the dimension of the input samples, while the output layer will have the same dimension as our desired prediction output. The number of hidden layers and neurons within each layer is user specified, and in practice there is often no right answer. The arrows between neurons correspond to multiplying the number in the starting neuron by a *weight* and adding it to the value in the next neuron. A *bias* is also added to each neuron which intuitively represents the 'default' value of a neuron when all weights are zero (and hence all other numbers in the sum are zero). A transformation is then applied to this sum, using an *activation function*. This output is

passed to the next layer as an input to repeat the whole process, starting from the input layer and ending at the output layer.

In the context of model fitting or training, the weights and biases are the parameters that must be estimated or 'learned' by the model through a process called *backpropagation* (see Higham [30]).

We now introduce some mathematical notation to the representation in Figure 19. Let $\sigma(\cdot)$ be the activation function (which we will explain in more detail later). Since the input data has the form $x \in \mathbb{R}^2$ and layer 2 has 2 neurons, we represent the weights and biases for layer 2 by the matrix $W^{[2]} \in \mathbb{R}^{2\times 2}$ and the vector $b^{[2]}$, respectively. The columns of $W^{[2]}$ correspond to the neurons in the previous layer, while the rows correspond to the neurons in the next layer. Define the values in the input layer by $a^{[1]} \coloneqq x$. The output from layer 2 is then given by

$$a^{[2]} = \sigma\left(W^{[2]}a^{[1]} + b^{[2]}\right) \in \mathbb{R}^2. \tag{3.14}$$

Layer 3 has three neurons, each receiving input in $\mathbb{R}^2$ (from layer 2), so the weights and biases for layer 3 may be represented by the matrix $W^{[3]} \in \mathbb{R}^{3\times 2}$ and the vector $b^{[3]} \in \mathbb{R}^3$, respectively. The output from layer 3 is then given by

$$a^{[3]} = \sigma\left(W^{[3]}a^{[2]} + b^{[3]}\right) \in \mathbb{R}^3. \tag{3.15}$$

Finally, the output layer (layer 4) has two neurons, each receiving input in $\mathbb{R}^3$ (from layer 3), so we represent the weights and biases for the output layer by the matrix $W^{[4]} \in \mathbb{R}^{2\times 3}$ and the vector $b^{[4]} \in \mathbb{R}^2$, respectively. The output from the output layer, and hence the final output from the neural network is given by

$$a^{[4]} = \sigma\left(W^{[4]}a^{[3]} + b^{[4]}\right) \in \mathbb{R}^2. \tag{3.16}$$

Combining (3.14), (3.15) and (3.16), the neural network represented by Figure 19 is defined by the following function:

$$F(x) = \sigma\left(W^{[4]}\sigma\left(W^{[3]}\sigma\left(W^{[2]}x + b^{[2]}\right) + b^{[3]}\right) + b^{[4]}\right) \in \mathbb{R}^2, \tag{3.17}$$

with input $x \in \mathbb{R}^2$. In general, given an input $x \in \mathbb{R}^{n_1}$, we can summarise a network of L layers with $n_1, n_2, \ldots, n_L$ neurons in layer 1, layer 2, ... , layer L, respectively, as follows:

$$\begin{aligned} a^{[1]} &= x \in \mathbb{R}^{n_1}, \\ a^{[l]} &= \sigma\left(W^{[l]}a^{[l-1]} + b^{[l]}\right). \in \mathbb{R}^{n_l} \quad \text{for } l = 2, 3, \ldots, L, \end{aligned}$$

where $a^{[L]} \in \mathbb{R}^{n_L}$ is the final output of the neural network. For electricity price forecasting, we set $n_L = 1$. This means that we will have to generate 24 separate forecasts for each delivery hour (see [32] for other ways of configuring the output layer).

If $\sigma(\cdot)$ was the identity function, then (3.17) can be reduced to a linear equation:

$$\begin{aligned} F(x) &= W^{[4]}\left(W^{[3]}\left(W^{[2]}x + b^{[2]}\right) + b^{[3]}\right) + b^{[4]} \\ &= W^{[4]}W^{[3]}W^{[2]}x + W^{[4]}W^{[3]}b^{[2]} + W^{[4]}b^{[3]} + b^{[4]} \\ &= W^{*}x + b^{*}, \end{aligned} \tag{3.18}$$

where $W^{*} = W^{[4]}W^{[3]}W^{[2]} \in \mathbb{R}^{2\times 2}$ and $b^{*} = W^{[4]}W^{[3]}b^{[2]} + W^{[4]}b^{[3]} + b^{[4]} \in \mathbb{R}^{2}$. This means that if we do not apply an activation function at all (or indeed if we use a linear activation function), the model reduces to a linear regression like (3.3), and we note that using any number of hidden layers is equivalent to having only an input and output layer. The value in using nonlinear activation functions is that it allows us to capture nonlinear patterns mappings. We will now present the activation functions that are most common in practice and that were used in the project (except for the linear function).

3.5.2 Activation Functions

Sigmoid: The sigmoid function is shown in Figure 20a and is given by

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{e^{x}+1} \in (0,1). \tag{3.19}$$

Hyperbolic Tangent (tanh): The tanh function is shown in Figure 20b and is given by

$$\sigma(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \in (-1,1).$$

Rectified Linear Unit (ReLU): The ReLU function is shown in Figure 20c and is given by

$$\sigma(x) = \max\{0, x\}, \in \mathbb{R}^{+}.$$

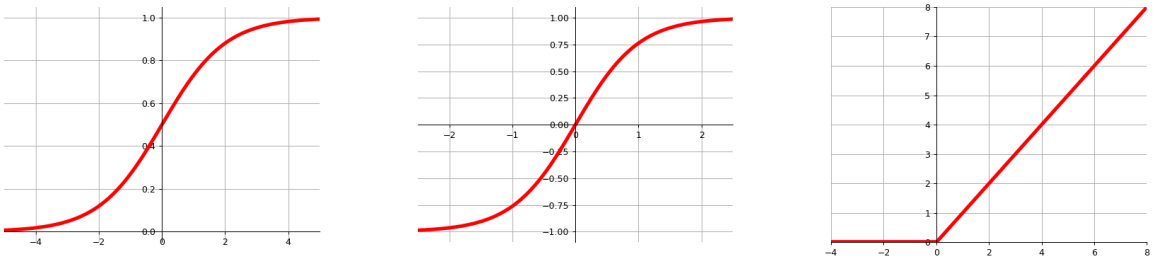
Linear activation function: Similar to (3.18), it can be shown that using a linear activation function of the form

$$\sigma(x) = ax + b \in \mathbb{R}, \qquad a, b \in \mathbb{R},$$

reduces the output to a linear combination of the inputs plus a bias term.

In practice, input samples are often rescaled to be within the range of the activation function output. This serves to speed up the model fitting process. For a sigmoid activation, the sample data $\{x_t\}_{t=1}^{n-1}$ is scaled to $\{\tilde{x}_t\}_{t=1}^{n-1}$ so that $\max\{\tilde{x}_t\}_{t=1}^{n-1} = 1$ and $\min\{\tilde{x}_t\}_{t=1}^{n-1} = 0$. Similarly for tanh activation, $\max\{\tilde{x}_t\}_{t=1}^{n-1} = 1$ and $\min\{\tilde{x}_t\}_{t=1}^{n-1} = -1$. Unbounded activation functions like ReLU may require a different treatment, e.g. normalisation (subtract by sample mean and divide by sample standard deviation).

Determining the best particular network structure (i.e. number of layers, number of neurons within each layer, activation functions, etc.) is often an art more than a science, and there are many different configurations that could work equally well. See Section 4.4.1 for an analysis of FFNN performance with respect to model complexity as measured by the number of layers and neurons. Keles et al. [32] carry out a more comprehensive comparison of several different FFNN architectures for electricity price forecasting.



(a) Sigmoid function (b) Hyperbolic Tangent (tanh) function (c) Rectified Linear Unit (ReLU) function

Figure 20: Shapes of different activation functions

3.5.3 Recurrent Neural Networks

Chollet explains the mechanics of the recurrent neural network and long-short term memory network (LSTM) in Section 6.2 of his book [31] and gives some examples of how to implement these using the `keras` library. Christopher Olah [33] also gives a good breakdown of how RNNs and LSTMs work in his blog. Our explanation is based on Olah's explanation in [33], and all figures were taken from there.

Ugurlu et al. [34] carried out a comparison of RNNs with other traditional methods in electricity price forecasting and with other types of neural networks.

When working with sequences or time-ordered data, FFNNs do not have *persistence*. They cannot remember previously observed values of a given sequence. While it is possible to pass previous values of the sequence as inputs in FFNNs, they cannot distinguish between day $D-1$ values and day $D-7$ values, for example, and we are relying on the training process to capture this difference.

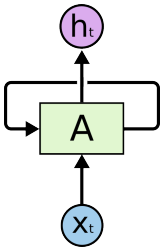


Figure 21: Layout of a basic RNN

The structure of an RNN is similar to that of an FFNN, but the RNN contains a loop that allows it to 'remember' previous values. Figure 21 shows a basic RNN unit A, with input x_t and output h_t. The loop from A to itself represents the cell *state* being passed from $t-1$ to t, then from t to $t+1$ and so on.

While the FFNN processes all the values of the sequence at the same time, the RNN will go through the sequence in order, updating the cell state as it passes through each value x_t in the sequence, and then generating an output when it reaches the final value. Figure 22 shows the same unit in Figure 21 being 'unrolled'.

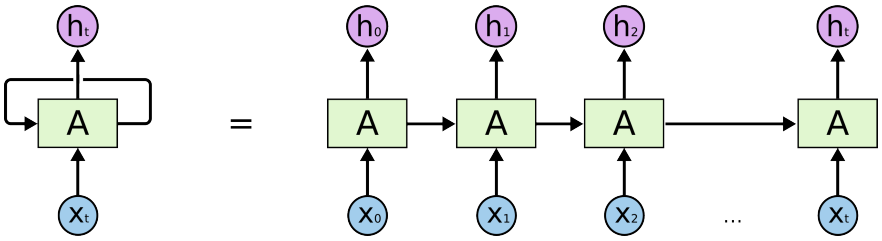


Figure 22: Unrolled RNN unit

Here, A is similar to a neuron in the FFNN in that it also applies an activation function, but with RNNs, the activation function is applied to a linear combination of the previous cell state h_{t-1} and the current input x_t, i.e.

$$h_t = \sigma\left(W_h h_{t-1} + W_x x_t + b\right), \tag{3.20}$$

where W_h and W_x are the model weights and b is the bias term. As with FFNNs, we can easily extend (3.20) to multiple dimensions (e.g. when we have multiple predictors). Note that an output h_i can be generated at each step, but for the purposes of forecasting, we configure the final RNN layer so that it only returns the last value of the sequence, while any intermediate RNN layers must return the entire sequence $\{h_0, \ldots, h_t\}$ as an input to the next RNN layer in the network.

Simple RNNs suffer from *short term memory*. They are able to model short sequences well, but as the gap grows, the RNNs become unable to retain very old information in the current cell state.

3.5.4 Long-Short Term Memory

The LSTM is a proposed improvement of the simple RNN that deals with the short term memory problem. Figure 23 shows the general structure within an LSTM unit, which we will go through step by step.

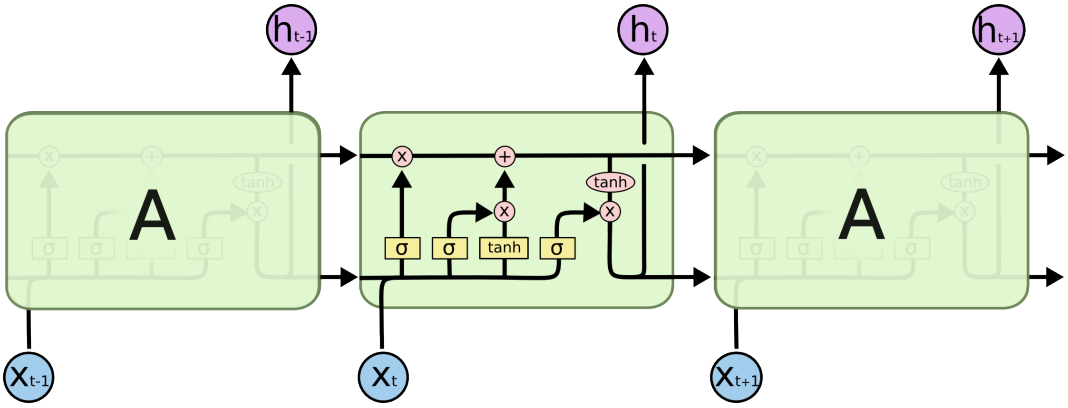


Figure 23: A closer look inside an LSTM unit

The yellow boxes represent a neural network layer within the unit that applies a given activation function to some linear combination of values, similar to an FFNN hidden layer. Note here that we define $\sigma(\cdot)$ as the sigmoid function given by (3.19) The pink circles represent pointwise operations,

e.g. pointwise multiplication, addition and (tanh) transformation.

The key behind LSTMs is the careful management of the cell state C_t: how much information should be forgotten from the previous cell state and how much new information should be added from the current cell state. Note that in simple RNNs, the output h_t is used as the cell state, whereas with RNNs, we differentiate between the cell state C_t and the output h_t.

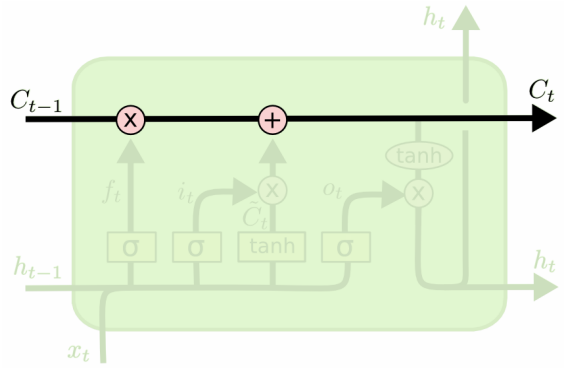


Figure 24: RNN cell states

The cell states are represented by the horizontal line along the top of the cell in Figure 24. The change in cell states is regulated by structures called *gates*. In total for LSTMs, there are three gates: the *forget gate*, the *input gate* and the *output gate*.

The forget gate, represented in Figure 25, controls how much of the previous cell state C_{t-1} will be 'forgotten'. Mathematically, it consists of calculating the proportion f_t, given by

$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right),$$

where W_f and U_f are the weights and b_f is the bias term. Recall that the sigmoid function output is between 0 and 1, so $f_t \in (0, 1)$. Thus, f_t can be understood as the proportion of C_{t-1} that will be retained. A value of 0 would mean 'forget all previous information', and a value of 1 would mean 'retain all previous information'.

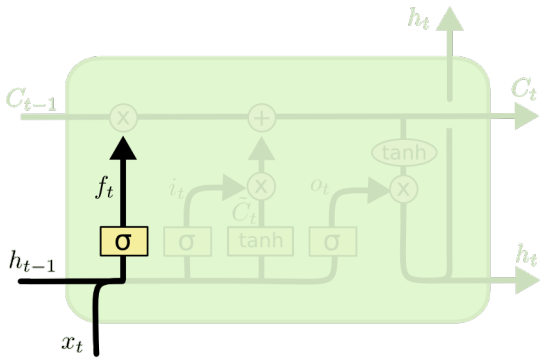


Figure 25: RNN forget gate

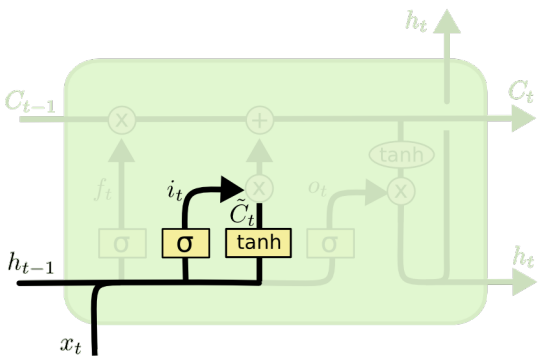


Figure 26: RNN input gate

Next, we look at the input gate, represented in Figure 26. The input gate calculates the candidate $\tilde{C}_t$ for the new current cell state and the proportion i_t of $\tilde{C}_t$ that will be used to create the new current cell state C_t:

$$\tilde{C}_t = \tanh\left(W_c x_t + U_c h_{t-1} + b_c\right)$$
$$i_t = \sigma\left(W_i x_t + U_i h_{t-1} + b_i\right)$$

We can now use f_t, C_{t-1}, i_t and $\tilde{C}_t$ to generate our new cell state C_t (see Figure 27):

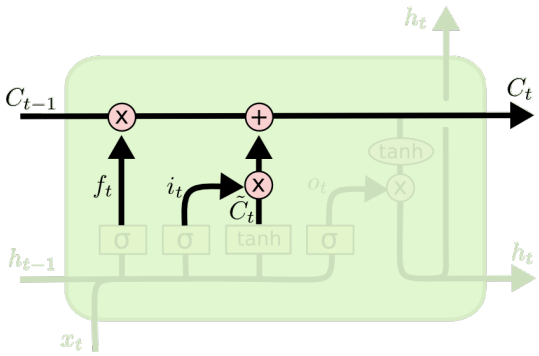


Figure 27: Updating C_{t-1} into C_t

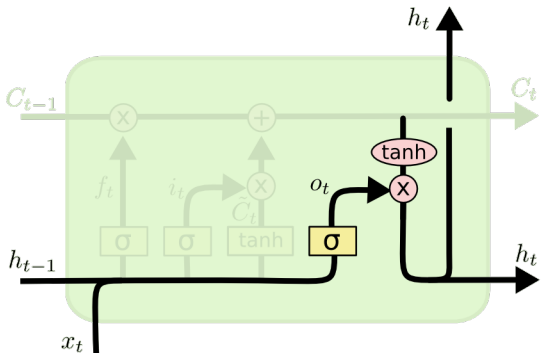


Figure 28: RNN output gate

$$C_t = \underbrace{f_t C_{t-1}}_{\text{old information}} + \underbrace{i_t \tilde{C}_t}_{\text{new information}}$$

Once we have calculated our new current cell state C_t, we can finally calculate the cell output h_t as follows:

$$o_t = \sigma\left(W_o x_t + U_o h_{t-1} + b_o\right)$$
$$h_t = o_t \tanh\left(C_t\right),$$

where W_o and U_o are the weights and b_o is the bias term (see Figure 28). Similar to i_t and f_t, o_t is a proportion that determines how much of the (transformed) current cell state $\tanh\left(C_t\right)$ will be used to calculate the output h_t. C_t and h_t will then be used along with x_{t+1} as inputs for the next step, until we reach our final observation x_{t^*} in the sequence, for which the output h_{t^*} will correspond to our forecast value.

Note that we use the tanh function in this example, but other activation functions can also be used, e.g. sigmoid and ReLU.

4 Implementation, Results and Observations

In Section 3, we gave an introduction to each modelling approach that was used. Here, we will give some details on how the models were implemented and how their performance was evaluated. We will then go through the different categories under which the models were evaluated and compared. It is important to note that these analyses are preliminary, but we hope that they will give the reader an idea of how to undertake an analysis and validation of a model in electricity price forecasting. All code scripts of the implementations can be found in the following GitHub link: `https://github.com/nvolfango/electricity_price_forecasting`.

4.1 Model Validation Framework

Recall that the main exercise here is to generate 24 day-ahead price forecasts for each delivery hour of the following day. Most, if not all, participants in the day-ahead market rely on the accuracy of such forecasts and other tools to aid their decisions. Hence, our primary measures of performance will be based on accuracy. In particular, we assess forecasts using the *root-mean-square error* (RMSE) and *mean absolute error* (MAE). Given a set of day-ahead forecasts $\hat{y}_{d,h}$ over $\mathcal{D}$ days for each observed price $y_{d,h}$, then

$$\text{RMSE} = \sqrt{\frac{1}{24\mathcal{D}} \sum_{d=1}^{\mathcal{D}} \sum_{h=1}^{24} \Big(\underbrace{y_{d,h} - \hat{y}_{d,h}}_{\text{error/residual}} \Big)^2} \tag{4.1}$$

$$\text{MAE} = \frac{1}{24\mathcal{D}} \sum_{d=1}^{\mathcal{D}} \sum_{h=1}^{24} \left| y_{d,h} - \hat{y}_{d,h} \right|. \tag{4.2}$$

It can be shown from (4.1) and (4.2) that RMSE $\geq$ MAE. Note that we can also calculate RMSEs over different time frames, as we will show later.

The MAE gives a linear (equal) weighting to each forecast error (also known as *residual*), whereas the RMSE gives a 'quadratic' weighting due to the squaring of the errors. Hence the RMSE is useful when we want to monitor larger deviations of the forecasts from the original values.

Some best practice guidelines to consider when evaluating a forecasting model are as follows:

- Forecast accuracy can vary from day to day, so we must assess forecasts over multiple days to give more reliable RMSE and MAE values for comparison and evaluation.
- We should simulate real life conditions when training the models, i.e. only use data that would have been available at that date and time to train the model.

To stay consistent with these guidelines, we use a *walk-forward validation* method which works as follows:

1. Pick a day d_0 for which to start generating forecasts.
2. Train the model on data available from day 1 to day d_0-1.

3. Generate forecasts $\hat{y}_{d_0,h}$, $h = 1, \ldots, 24$, for day d_0.
4. Add day d_0 to the training data.
5. Pick day $d_1 = d_0 + 1$ for the next set of forecasts.
6. Repeat steps 2-5 until the final day d_* has been forecasted.

We chose $d_0 = 1^{\text{st}}$ January 2020 and $d_* = 9^{\text{th}}$ July 2020 which is equivalent to 191 days or 4,584 hours of price forecasts. Because our available data begins on the 12^{th} November 2018, our walk-forward forecast on the 1^{st} January 2020 will have 415 days of data to train on. We used the `mean_squared_error()` and `mean_absolute_error()` functions in the scikit-learn (`sklearn`) library [38] to facilitate the calculation of the RMSEs and MAEs, respectively.

4.2 Implementation

We have already mentioned some of the specific model configurations that we used in Section 3, but we have included them here for completeness. We also provide information on the Python libraries used and key points about the data preparation and training algorithm of each model, where necessary.

We will not give specific detail on the training time under each model because it is very dependent on the machine (physical or virtual) being used as well as the level of optimisation and efficiency in the code. However, we will informally mention the training times where they are relevant to the implementation design.

To maintain comparability between models, we allowed for model dependencies only as far back as 7 days, except for the SARIMAX which is configured according to our ACF/PACF analysis in Section 3.2.9.

4.2.1 Data Preprocessing

Much of the data manipulation was done using the Python Data Analysis (`pandas`) library [35] for organising the data in `Dataframe` (table) objects and the `datetime` library [36] for manipulating dates and times. A significant portion of the work in this project (as with any data analysis exercise) was focused on:

- Ensuring data integrity: Removing or replacing invalid values with other appropriate values (e.g. the mean or median), ensuring that each date has exactly 24 hours of data unlike the daylight savings days which have either 23 or 25 hours, removing duplicates, ensuring Python data types are consistent, etc.
- Reorganising the data into the appropriate format as input to the models: The random forest and feedforward neural network models require a column for each new variable (endogenous or exogenous) within the same table, while the time series methods require separate tables for the endogenous and exogenous data. Interestingly, recurrent neural networks require the data to be organised in three-dimensional tables (also called *3D tensors* in machine learning terminology).

4.2.2 Time Series

We used the `statsmodels` library [37] to train the AR/ARX and SARIMA/SARIMAX models. We use the `ar_model.AutoReg()` function to train both the AR and ARX model and the `sarimax.SARIMAX()` function to train the SARIMA and SARIMAX models (where adding exogenous variables is as simple as passing the data as inputs).

The training time of an AR model is significantly shorter (in fact, almost instantaneous) compared to that of the SARIMA. Because of this, we are able to fit several AR models with different (possible) dependencies and then select the best one to generate the forecast. On the other hand, one SARIMA model takes much longer to fit due to the computational expense in estimating the seasonal components.

For the AR model, we train seven models for each iteration of the walk-forward validation – AR(24), AR(48), ..., AR(168) – and then use the model with the lowest AIC to generate the day-ahead forecast. For the SARIMA, we train only one model: SARIMA$(4,1,4) \times (0,1,1)_{24}$. We followed the same procedure for the ARX and SARIMAX models.

4.2.3 Random Forest

We used the `sklearn` library to train the random forest and its underlying decision trees (using the `ensemble.RandomForestRegressor()` function).

As an initial step, we implemented a *randomised search* (using the `RandomizedSearchCV()` function) which randomly chooses different combinations of some of the hyperparameters of the model and then fits each random forest model to the data. We then use the model with the lowest training data accuracy for each iteration of the walk-forward validation. This has the benefit of minimising the required training time for each walk-forward forecast.

Unlike the time series models, each forecast hour is only allowed to depend on the same hour of previous days, e.g. hour 1 of day d will depend only on hour 1 of days $d-1, \ldots, d-7$.

The final model configuration we used had

- 100 decision trees (`n_estimators = 100`),
- each with a maximum tree depth of 26 (`max_depth = 26`),
- and 15 features/predictors considered at each split (`max_features = 15`).

Recall that each lagged value of a time series counts as a feature. Finally, because random forests allow for extracting a summary of variable importance (see Section 3.3.4), we make use of this feature. In particular, at each walk-forward iteration, we store the variable importances. The final variable importance is obtained by calculating the mean across all the forecast dates.

4.2.4 X-Model

This model required the greatest amount of preprocessing work. Recall that the premise of the X-Model is that DAM prices are largely determined by the behaviour of the underlying supply and demand curves, so several preliminary steps are required before getting to the model training

step. We used the `sklearn` library's `linear_model.LassoCV()` function to train each of the 936 linear models with lasso cross-validation which estimates the model parameters using the coordinate descent optimisation algorithm (as in the original paper [26]).

The unique dependency structure (see again Figure 18) required great care to ensure a correct and efficient implementation. The X-Model also had the longest training time of all the models. Even though each linear model took one second to train on average on our local machine, there were 936 of these models. Assuming a one-second training time per linear model amounts to 15.6 minutes for each day-ahead forecast. Then 191 days of walk-validation forecasts would require approximately two days to run.

4.2.5 Neural Networks

As previously mentioned, we used the `keras` library to train the FFNNs (using the `Dense()` layer object) and the LSTMs (using the `LSTM()` layer object). We decided to run the walk-forward validation on each neural network architecture using each of the three activation functions mentioned in Section 3.5.2 (sigmoid, tanh and ReLU).

Recall that we set both FFNNs and LSTMs with a default of 3 hidden layers, with each layer having the same number of neurons. We used 12 neurons per layer for FFNNs and 6 neurons per layer for LSTMs. Similar to random forests, each hour is only allowed to depend on the same hour of previous days, as far back as 7 days.

To minimise the execution time of the walk-forward validation for the entire period, we designed it so that the model is well-calibrated to the training data for the first forecast and only partially calibrated every time we add a new day of data, since the model will already be trained to all but one day of the new training data.

As previously mentioned, the 'best' neural network architecture can often only be found by trial and error, so our choice of layers may not be optimal, or even near optimal. Section 4.4.1 shows a deeper analysis to help determine a more optimal model (in terms of the number of neurons) than what was used in the walk-forward validation.

4.3 Walk-Forward Validation Results

Here, we finally present the results of the walk-forward validation of each model. We will first look at the overall performance of each model for the period between 1$^{\text{st}}$ January 2020 and 9$^{\text{th}}$ July 2020, but we will also show the performance in different ways. Note that for brevity, we will only show the results that have something worth noting.

4.3.1 Performance for the Entire Period

Figure 29 shows the walk-forward results (RMSE and MAE) for the entire period, sorted in increasing order. All three naive models were the worst performers, which is a good indicator that the other models were able to utilise the predictors to improve performance.

	RMSE
RNN_sigmoid	7.91
FFNN_sigmoid	8.21
random_forest	8.36
RNN_relu	8.74
ARX	8.79
RNN_tanh	9.04
XModel	9.66
FFNN_relu	9.96
SARIMAX	10.29
FFNN_tanh	10.69
SARIMA	12.47
AR	13.08
naive_1d	15.69
naive_7d	17.07
naive_1y	33.02

	MAE
RNN_sigmoid	5.67
FFNN_sigmoid	5.89
random_forest	5.92
RNN_relu	6.00
RNN_tanh	6.34
ARX	6.42
FFNN_relu	6.71
XModel	6.78
FFNN_tanh	6.93
SARIMAX	7.81
SARIMA	8.77
AR	9.05
naive_1d	10.54
naive_7d	11.59
naive_1y	25.32

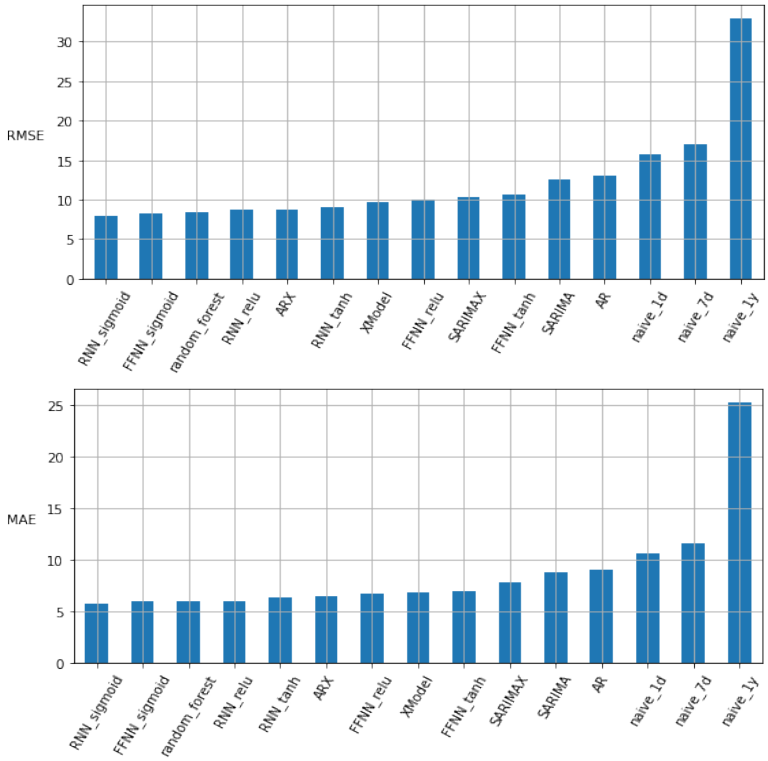


Figure 29: Overall RMSE and MAE of all models for the entire period.

We can see that the general order of the models is not much different between the RMSE and MAE. Overall, the sigmoid neural networks have the best performance. In comparison, the tanh and ReLU neural networks underperformed. The random forest model is the third best in terms of both RMSE and MAE. We note that the (sigmoid) FFNN and random forest models performed better than the time series models, but even though the (sigmoid) RNN is explicitly designed for time-indexed data, it only marginally outperformed the FFNN which does not take the time order into account.

For the time series methods, the exogenous variants (ARX/SARIMAX) performed better than the endogenous-only variants, which is not surprising, since we expect this extra information would help improve forecast accuracy. The SARIMAX did not perform better than the ARX, which could be due to the limitations caused by the computational expense in the `SARIMAX()` function or because the SARIMAX component only has a seasonal AR of order 1, which intuitively means that the dependence goes back only one day, in contrast to 7 days for the ARX model. However, we would tentatively say that the former is more likely because we do not see the same result between the AR and SARIMA. In this case, the SARIMA performed slightly better.

In general under the `statsmodels` package, The SARIMA/SARIMAX requires significantly more calculations than the AR/ARX, and each step (also called *iteration*) of the optimisation algorithm towards the minimum can take a few seconds to run. Because of this, and due to time constraints, we chose 100 maximum iterations for each training step of the walk-forward validation. See Section 4.4.2 for an analysis of SARIMAX performance with respect to the maximum number of iterations.

From here on, we will only include the best variants under each model/method. We will omit the following: `naive_1y`, `naive_7d`, `AR`, `SARIMA`, `FFNN_tanh`, `FFNN_relu`, `RNN_tanh`, `RNN_relu`.

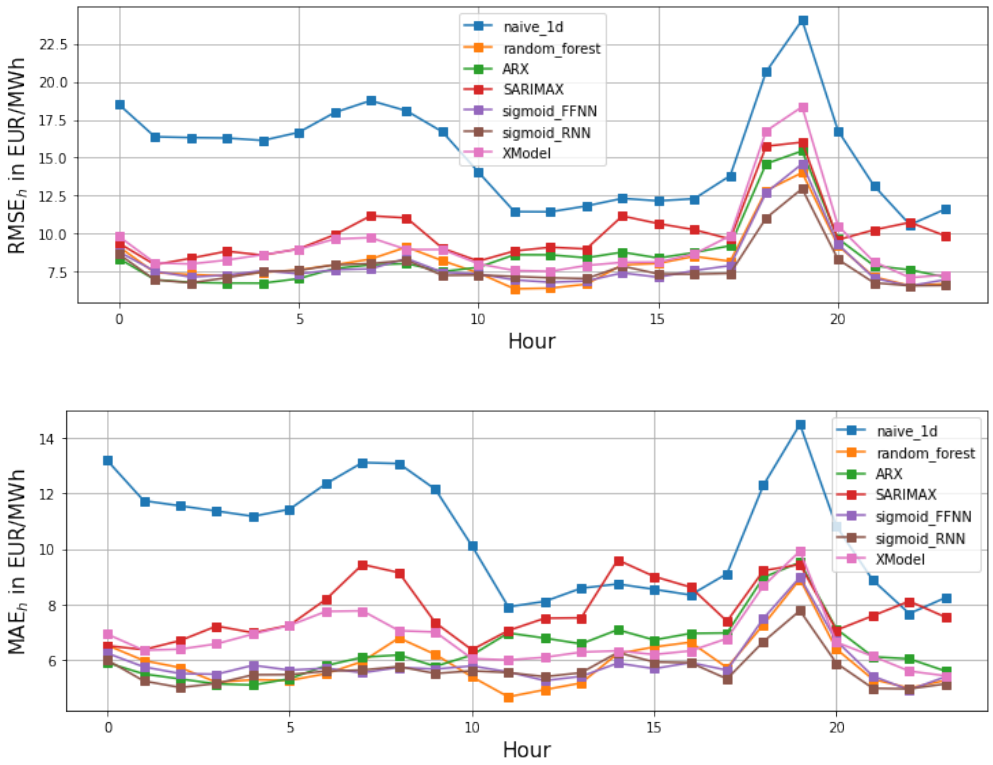


Figure 30: RMSE and MAE for the entire period – grouped by hour

Figure 30 shows the plots of the RMSE and MAE for the entire period, grouped according to each hour of the day. Once again, we see that the `naive_1d` is the worst performing and sets a kind of upper bound of the minimum expected performance for the other models for each hour. The model RMSEs appear to follow a similar structural pattern, with a spike at hours 18 and 19, indicating poorer performance for these hours.

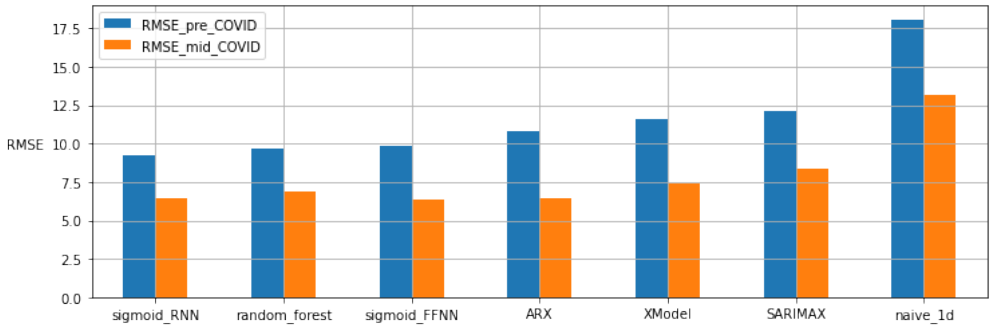
The SARIMAX seems to deviate from the other models in its MAE more than in its RMSE. This again could point to the computational limitations in the estimation procedure used for SARIMAX (see Section 4.4.2).

4.3.2 COVID Performance

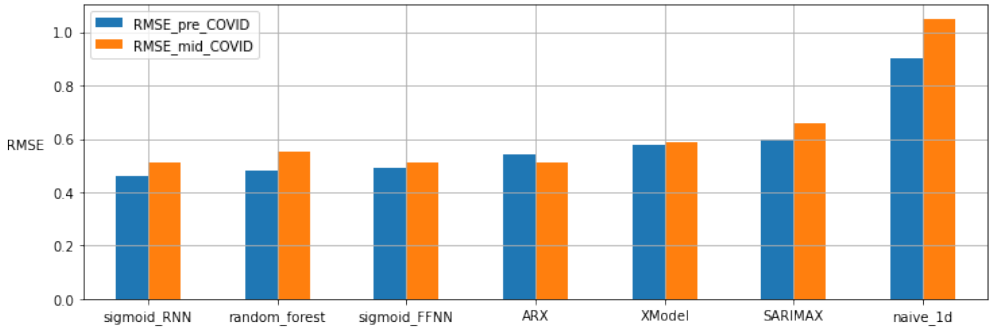
As we mentioned in Section 2, the COVID pandemic has had downward pressure on prices (along with the increase of wind capacity), but we would be interested to see how it has affected the performance of the models. To do this, we have divided the forecast results into 'pre-COVID' (January-March) and 'mid-COVID' (April-July) forecasts.

For brevity, we will only examine the RMSE values. Figure 31 shows a comparison of the RMSE between the pre-COVID and mid-COVID periods. Figure 31b shows the plot of the same data from

Figure 31a but where each RMSE is scaled down relative to the standard deviation of the DAM prices in each period.



(a) Unscaled



(b) Scaled

Figure 31: RMSE for pre-COVID -v- mid-COVID forecasts

Figure 31a indicates that all models perform better in the mid-COVID period in absolute terms. However, recall from Section 2 that the price levels and volatility are smaller in the later periods (mid-COVID) than in the earlier periods (pre-COVID). Figure 31b shows a fairer representation of the models. In general, we see the opposite – the pre-COVID performance was relatively better than the mid-COVID performance, except for the ARX model.

Figure 32 shows the RMSE plots grouped by hour for the pre-COVID and mid-COVID periods. The general structural pattern is different between the two periods. In particular, we no longer see the same spike of poorer performance at hour 18 and 19 in the mid-COVID RMSEs. Instead, it has a flat shape, with some minor peaks at hours 6-9 and 14-16. We believe this is mainly due to the distribution of the price spikes, which we will now deal with under the next heading.

4.3.3 Price Spike Performance

In this context, a price spike is a price level that is considered ‘extreme’. The price forecasting literature has several different criteria to determine ‘extreme’ values. The X-Model paper [26] picks specific days that, visually, have visually noticeable price spikes and then monitors performance on these price spikes using a probabilistic forecasting method called *residual-based bootstrapping* to measure how well a model captures the *likelihood* of a price spike. Alternatively, Keles et al. [32] pick specific price upper and lower bounds to determine extreme price levels.

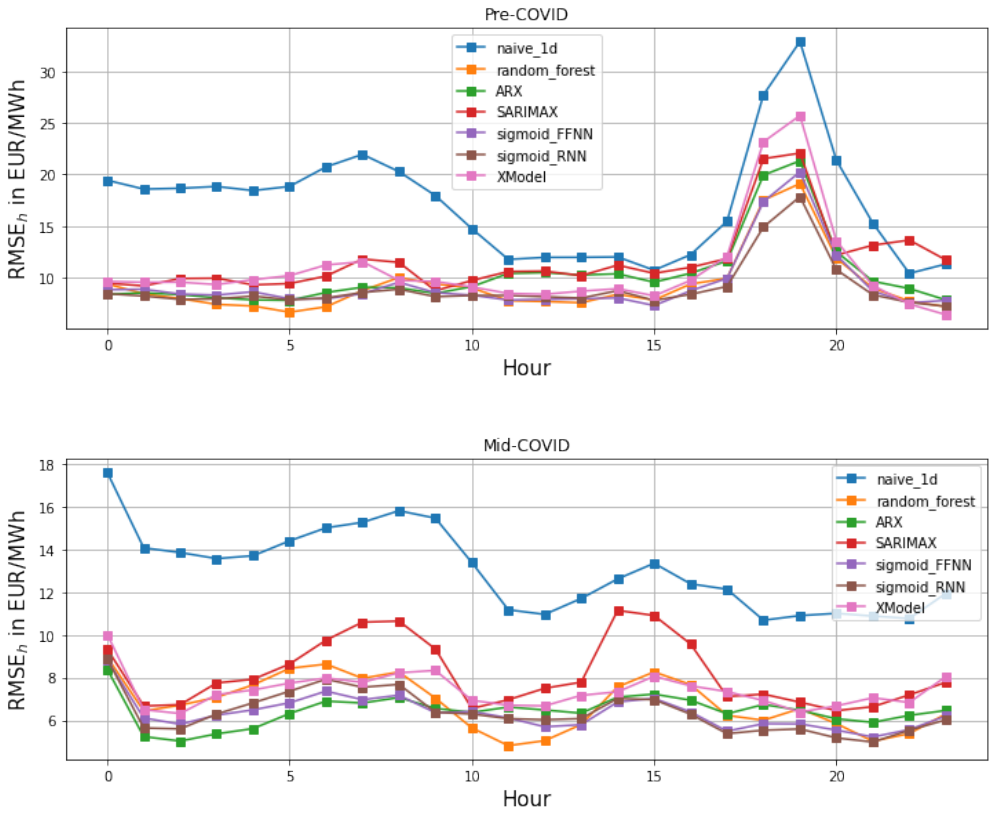


Figure 32: RMSE for pre-COVID -v- mid-COVID – grouped by hour

Given that the data shows a decreasing trend in mean price levels and in volatility, it would not be prudent to set a single price level as an upper/lower bound for the entire period. Instead, we uses a 7-day rolling interval to determine extreme prices. More specifically, the upper and lower bounds are given by the 90th and 10th percentile, respectively, over a 7-day window. Prices that fall outside this interval are considered to be price spikes. Figure 33 shows a plot of the DAM prices in the walk-forward validation period and the corresponding rolling interval.

It may be of interest to compare performance between price spikes and non-extreme prices, but we will instead look at a performance comparison between price spikes in the positive direction and price spikes in the negative direction, as it is directly relevant to the COVID performance analysis.

Figure 34 shows the distribution of the positive and negative price spikes across the hours of the day. For example, Figure 34a indicates that 10% of the positive price spikes occur at hour 19. So then it would seem that the poorer performance of all the models in hours 18 and 19 (in Figure 32) is due to the positive price spikes (not so much because of the negative price spikes).

We would add that the similarity in underperformance for hours 18 and 19 (which are the hours with the highest electricity demand, corresponding to the period 17:00-19:00) between the pre-COVID and positive price spike performance is the fact that most of the positive price spikes actually

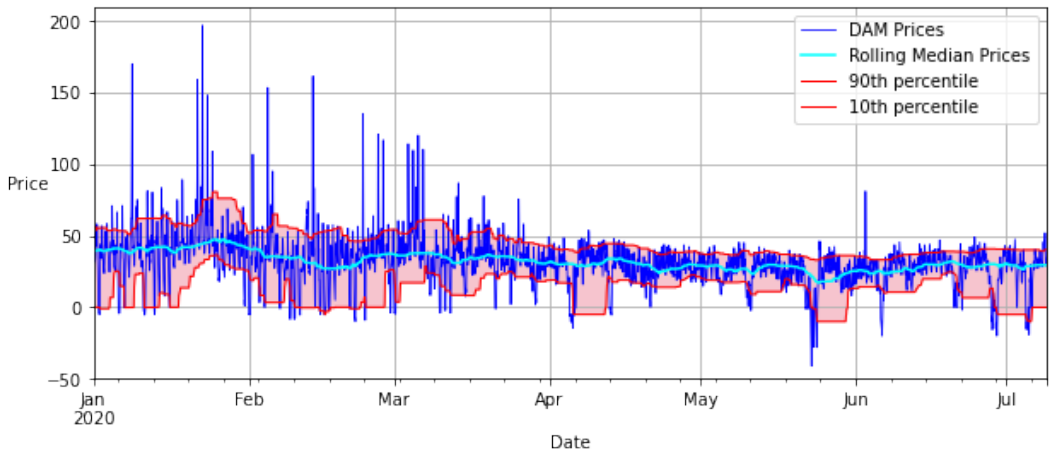
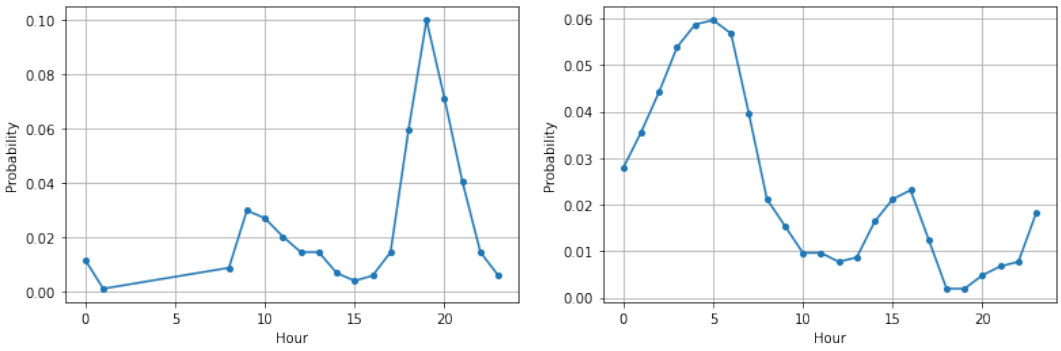


Figure 33: Seven-day rolling window of the interval for determining price spikes

occurred in the pre-COVID period (January-March), whereas most of the negative price spikes are in the mid-COVID period (April-July).



(a) Price spikes in the positive direction

(b) Price spikes in the negative direction

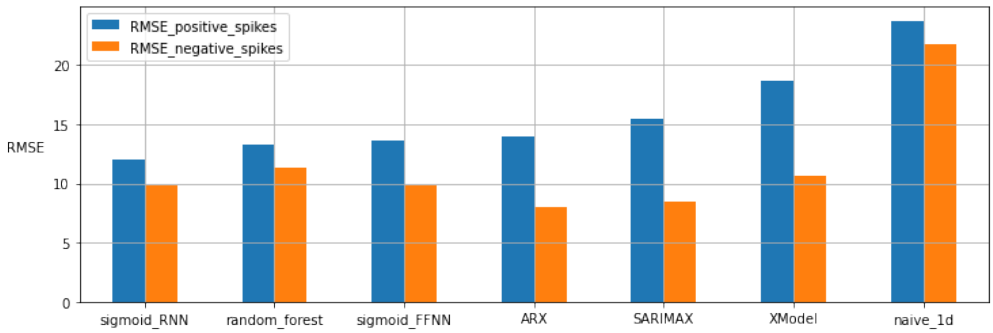
Figure 34: Distribution of price spikes across the hours of the day

Finally, we note that the performance for negative spikes is better in absolute terms but slightly worse in relative terms (i.e. with scaling) (see Figure 35). The time series methods and the X-Model are the most consistent in their performance between positive and negative spikes. On the other hand, the best models overall, the neural networks and random forest, seem to underperform more on negative spikes than on positive spikes.

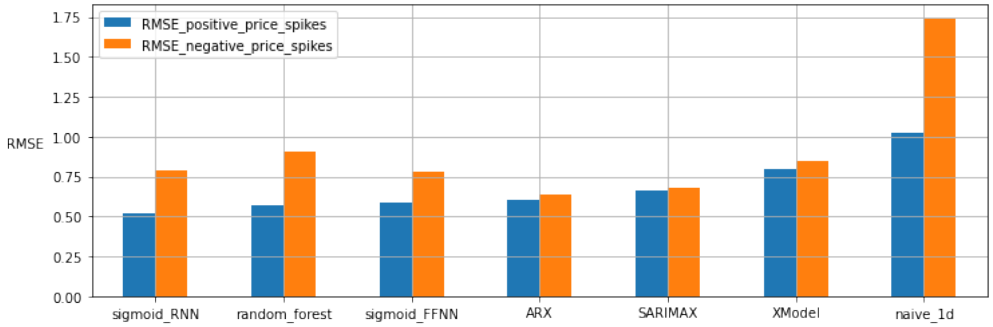
This underperformance with negative spikes relative is no surprise because they seem to occur only in the later periods in the price data. Recall that models are trained on past data. If the unseen data follows a different pattern to the training data, then the models are likely to underperform. Nevertheless, the underperformance of the models for negative spikes relative to the positive spikes is not as large as with the `naive_1d` benchmark.

We would also make the same observation about the underperformance of mid-COVID forecasts relative to pre-COVID forecasts. The mid-COVID conditions occurred only in the last 5 months of

data, out of 20 months of data in the available period. This means that during the walk-forward validation, especially in earlier iterations, most of the training set reflects pre-COVID conditions even though the forecast is for a mid-COVID date. However, the underperformance does not seem significant either, indicating that much of the underlying dynamics in the prices has not changed, resulting in forecasts that still soundly outperform the `naive_1d` benchmark.



(a) Unscaled



(b) Scaled

Figure 35: RMES for positive price spike forecasts -v- negative price spike forecasts

4.3.4 Variable Importance

We would like to address the following question: "Which predictors/features contain the most information that would be relevant for predicting DAM prices?" Recall from Sections 3.3.4 and 4.2.3 that the random forests model allows us to extract a summary of variable importance by measuring the average overall reduction in RSS when splitting each predictor in a decision tree (see Figure 36).

The implementation for calculating feature importances in `sklearn` also normalises the mean overall reduction in RSS, so the values in the table in Figure 36 are interpreted as follows: On average, for each day-ahead forecast, $\text{VI}_p\%$ of the reduction in total RSS is due to a decision tree split in the corresponding predictor.

Lagged values of the DAM prices amount to more than 50% of the overall feature importance, which might indicate an over-reliance on past values of the DAM prices as a predictor. In Section 4.4.3, we conduct an analysis of random forest performance with respect to the number of features considered at each decision tree split (recall from Section 3.3.3 that random forests have this feature to minimise

over-reliance on ‘strong’ predictors).

It is noteworthy that none of the lagged BM prices are in the top 10 variables in Figure 36. In fact, all BM price variables together accounted for only 4.1% of the overall RSS reduction.

	VI_p
EURPrices-24	27.2
Wind-0	16.3
EURPrices-48	8.8
EURPrices-168	7.3
EURPrices-120	5.1
Demand-0	5.1
EURPrices-144	3.8
EURPrices-72	3.5
Demand-168	3.1
EURPrices-96	2.9

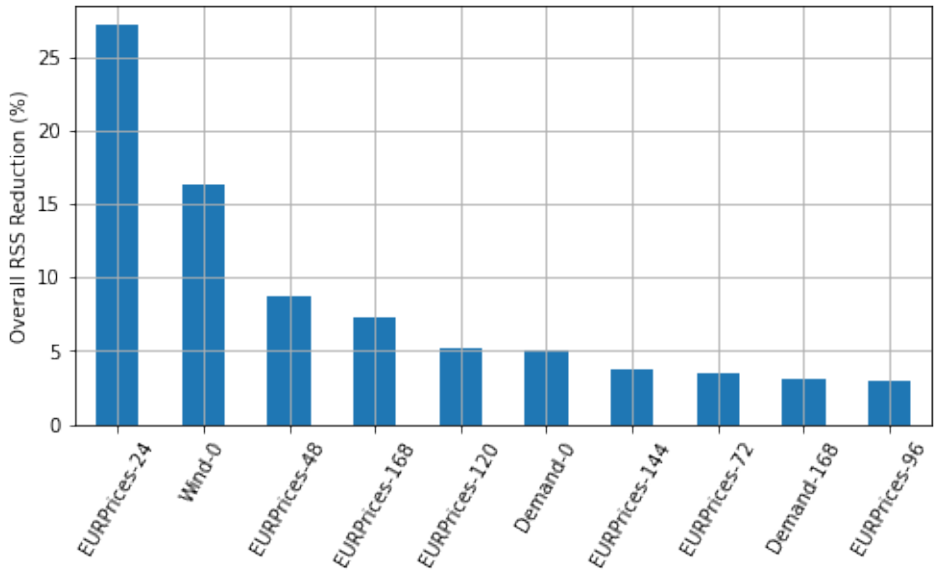


Figure 36: Variable importances

4.4 Hyperparameter Tuning

In this subsection, we will show the effect on model performance when varying the hyperparameters. Recall that a hyperparameter is a model input parameter that cannot be estimated and must be specified by the user in advance. The most optimal hyperparameter can only found by training the model with different hyperparameters (and all other parameters remaining equal) and then picking the one with the best performance.

Because we are trying out several possible hyperparameter values for each model, in order to minimise the computation time, we have restricted our walk-forward validation to only a few dates:

- 8$^{\text{th}}$ January 2020 – Date of worst performance in general for all models
- 22$^{\text{nd}}$ January 2020 – Date of the positive price spike of the greatest magnitude
- 15$^{\text{th}}$ May 2020 – Date of best performance in general for all models
- 23$^{\text{rd}}$ May 2020 – Date of the negative price spike of the greatest magnitude

For ease of comparison we have included the performance of the original model under the complete walk-forward validation for the dates above in each plot.

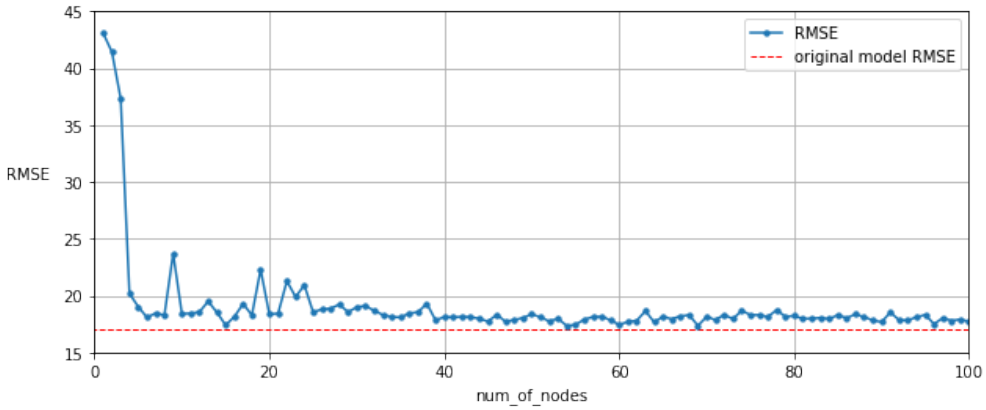


Figure 37: Plot of sigmoid FFNN RMSE against the number of neurons per layer

4.4.1 Sigmoid FFNN – Number of neurons per layer

Figure 37 shows the performance of the sigmoid FFNN with respect to the number of neurons per layer, labelled `num_of_nodes`. Recall that we chose a default of 3 hidden layers. It seems that choosing at least 4 neurons per layer already reduces the error to the best possible configuration. We do note that the performance is inconsistent between 9 and 24 neurons per layer and then settles to a minimum past 24 neurons per layer.

The original model RMSE is lower than the rest because as the calibration method was slightly different for the original model than for these models. In the original walk-forward validation, we only did a partial calibration to each new day of training data, whereas for these models, we did a full calibration to the available training data each time.

4.4.2 SARIMAX – Maximum number of training iterations

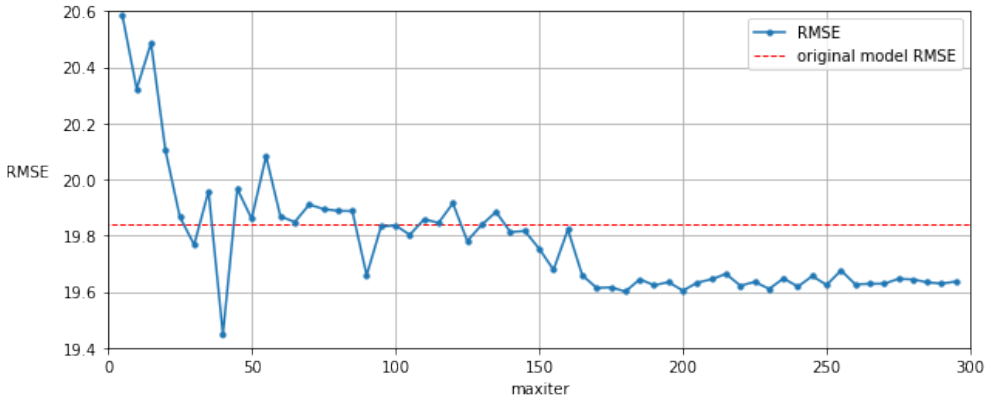


Figure 38: Plot of SARIMAX RMSE against the maximum number of iterations

Under the original SARIMAX model, we set the maximum number of iterations, labelled `maxiter`, to 100, which results in an RMSE of 19.84 EUR/MWh (see red dashed line in Figure 38). The model performance is very erratic but settles at a minimum after 165 maximum iterations. It is

interesting that the best performance of all was achieved at 40 maximum iterations, but we would say that this is likely to be a coincidence, and one could run a walk-forward validation for the entire period with `maxiter = 40` to check this.

Finally, we note that the `maxiter` setting for the original model was definitely not the most optimal choice, and this might have to do with the SARIMAX having similar hourly RMSE values to the other models but greater hourly MAE values.

4.4.3 Random Forest – Number of features considered at each split

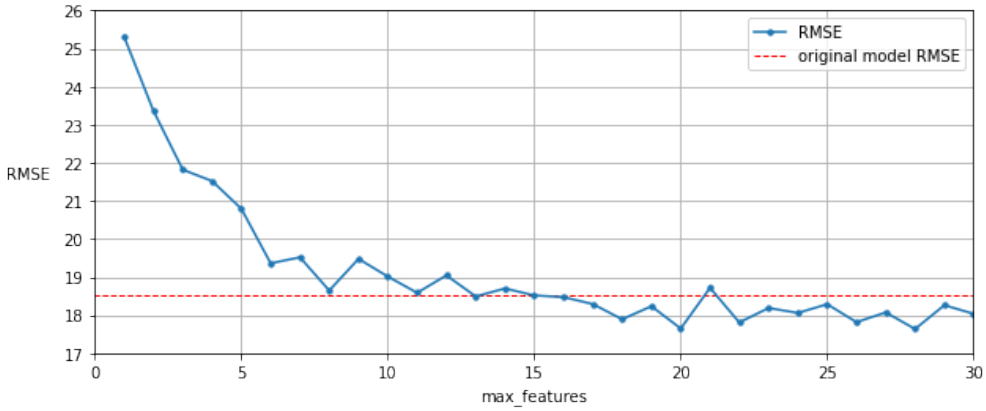


Figure 39: Plot of random forest RMSE against the number of features considered at each split

As we have mentioned in Section 4.3.4, lagged values of the DAM prices accounted for more than 50% of the mean overall reduction in RSS, which might be an indication that the number of features being considered at each tree split (`max_features = 15`) might be too large, and other values might be more optimal. Figure 39 shows that this observation was unfounded. In fact, it seems that our model might benefit from using a larger value of `max_features`.

Note that the total number of features available is 30 and that a random forest with `max_features = 30` is equivalent to a bagging algorithm. The random forest performance flattens out for `max_features` $\geq$ `18` because at this point, the strongest predictors are already being used at each split, so adding more predictors for consideration will have little or no effect on the final predictor that is chosen by the training algorithm.

5 Conclusions

We have presented one approach to the problem of electricity price forecasting in the day-ahead market. Although the available data has consisted entirely of time series data (DAM prices, BM prices, wind forecast and demand forecast), we have shown that, with the right preprocessing steps, it is possible to use methods not explicitly designed for time series and still achieve a forecast accuracy as good as (or better than) the traditional time series methods.

5.1 Model Comparison

To summarise, the sigmoid RNN was the best model in terms of overall performance. The sigmoid FFNN and random forest are comparable, with one outperforming the other under different categories. We would rank the ARX next, having performed better than the SARIMAX. The results of the X-Model and SARIMAX model are comparable.

However, when taking other important factors into account, such as training time, robustness of the model, configurability, etc., then the random forest and the FFNN would equally be the best models, with ARX just below them. The walk-forward validation took approximately one minute for the ARX and just under 9 minutes and 7 minutes for the random forest and FFNN, respectively. On the other hand, the (sigmoid) RNN took almost 4 hours, but its performance is only marginally better than the FFNN and random forest.

In addition, as we have mentioned, the `keras` RNN implementation requires that data be passed as 3-dimensional tables, which required care when reformatting the data. On the other hand, random forests and the ARX (as well as the FFNN) take 2-dimensional inputs, which is much easier to handle. Random forests (in the `sklearn` library) also have the option of extracting variable importance. This is also possible for the neural networks and ARX but would take some work to calculate and compile for analysis.

5.2 Further Development

As a possible improvement over the ARX, one could use a GARCH-type model (see Section 3.8.6 of [16]). All time series models we have used make the assumption of constant volatility in the endogenous data, while the GARCH is a generalised model that allows for a varying volatility. Recall that we mentioned other papers using combinations of different models. The ARX and SARIMAX are examples of such models, where we use a linear model with exogenous predictors and then model the residuals of the forecasts using a time series model.

We could also improve the performance of the existing models by incorporating more data, e.g. solar forecasts and other weather data, generator availabilities, holiday effects, generator fuel costs, and so on. When considering dozens of potential predictors, it would be wise to add a feature selection step to filter out predictors that would not be useful (see [32] which uses clustering algorithms).

Finally, if we decided to use more data and models, then we would need a more objective measure of model performance. Ugurlu et al. [34] use the Diebold-Mariano test to evaluate the statistical significance of the difference in performance between the models.

References

[1] ElectroRoute 2019, *How Ireland's Electricity is Traded*, <http://electroroute.com/how-irelands-electricity-is-traded/>, accessed 3rd August 2020.

[2] Ugurlu, U., Tas, O., Gunduz, U. (2018). Performance of Electricity Price Forecasting Models: Evidence from Turkey. *Emerg. Mark. Financ. Trade 2018*, doi:10.1080/1540496X.2017.1419955.

[3] Geman, H., Roncoroni, A. (2006). Understanding the fine structure of electricity prices. *J. Bus. 2006, 79*, pp. 1225–1261.

[4] Cartea, A., Figueroa, M.G. (2005). Pricing in electricity markets: Amean reverting jump diffusion model with seasonality. *Appl. Math. Financ. 2005, 12*, pp. 313–335.

[5] Janczura, J., Trück, S., Weron, R., Wolff, R.C. (2013). Identifying spikes and seasonal components in electricity spot price data: A guide to robust modelling. *Energy Econ. 2013, 38*, pp. 96–110.

[6] Hayfavi, A., Talasli, I. (2014). Stochastic multifactor modelling of spot electricity prices. *J. Comput. Appl. Math. 2014, 259*, pp. 434–442.

[7] Janczura, J., Weron, R. (2010). An empirical comparison of alternate regime-switching models for electricity spot prices. *Energy Econ. 2010, 32*, pp. 1059–1073, doi:10.1016/j.eneco.2010.05.008.

[8] Eichler, M., Tuerk, D. (2013). Fitting semiparametric Markov regime-switching models to electricity spot prices. *Energy Econ. 2013, 36*, pp. 614–624.

[9] Weron, R. (2006). *Modeling and forecasting electricity loads and prices: a statistical approach.* Chichester: Wiley.

[10] Karakatsani, N. V., Bunn, D. W. (2008). Forecasting electricity prices: the impact of fundamentals and time-varying coefficients. *International Journal of Forecasting, 24*(4), pp. 764–785.

[11] Karakatsani, N. V., Bunn, D. W. (2010). Fundamental and behavioural drivers of electricity price volatility. *Studies in Nonlinear Dynamics and Econometrics, 14*(4), art. no. 4.

[12] Misiorek, A., Trück, S., Weron, R. (2006). Point and interval forecasting of spot electricity prices: Linear vs. non-linear time series models. *Studies in Nonlinear Dynamics and Econometrics, 10*(3), Article 2.

[13] EirGrid 2020, *EirGrid Reveals Record Breaking Wind Energy Levels*, <http://www.eirgridgroup.com/newsroom/record-wind-levels-feb-20/index.xml>, accessed 3rd August 2020.

[14] SEMOpx 2020, SEMOpx_Data_Publication_Guide.zip, Market Data, Document Library Database.

[15] SEMO 2020, *SEMO Data Publication Guide*, General Publications Database.

[16] Weron R. (2014). Electricity price forecasting: a review of the state-of-the-art with a look into the future. Int J Forecast 2014;30:1030–81.

[17] Nogales, F.J., Contreras, J., Conejo, A.J., Espinola, R. (2002). Forecasting next-day electricity prices by time series models. *IEEE Trans. Power Syst. 2002, 17*, pp. 342–348, doi:10.1109/tpwrs.2002.1007902.

[18] Contreras, J., Espinola, R., Nogales, F.J., Conejo, A.J. (2003). ARIMA models to predict next-day electricity prices. *IEEE Trans. Power Syst. 2003, 18*, pp. 1014–1020.

[19] Conejo, A.J., Plazas, M.A., Espinola, R., Molina, A.B. (2005). Day-ahead electricity price forecasting using the wavelet transform and ARIMA models. *IEEE Trans. Power Syst. 2005, 20*, pp. 1035–1042.

[20] *Time Series Analysis* (1994), Hamilton, J. D. Princeton University Press, 1st edition.

[21] *The Analysis of Time Series: An Introduction* (2003) Chatfield, C. (2003). Chapman & Hall/CRC Texts in Statistical Science.

[22] Fabozzi, F. J., Focardi, S. M., Rachev, S. T., and Arshanapalli, B. G. (2014). The Basics of Financial Econometrics: Tools, Concepts, and Asset Management Applications. Appendix E. *Model Selection Criterion: AIC and BIC*. https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118856406.app5

[23] Hyndman, R. J. (2010). *The ARIMAX Model Muddle*. Hyndsight blog. <http://robjhyndman.com/hyndsight/arimax/>

[24] Kaggle Competitions 2020, <https://www.kaggle.com/competitions>.

[25] James, G., Witten, D., Hastie, T., Tibshirani, R. (2013). An Introduction to Statistical Learning with Applications in R. Chapter 8: Tree-Based Methods. Springer

[26] Ziel, F., Steinert, R. (2016). Electricity Price Forecasting using Sale and Purchase Curves: The X-Model. Energy Econ. 2016, 59, 435–454, doi:10.1016/j.eneco.2016.08.008. https://arxiv.org/abs/1509.00372

[27] Industry Guide to the I-SEM. I-SEM Project. Version 1.0. <https://www.sem-o.com/documents/general-publications/I-SEM-Industry-Guide.pdf>

[28] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 58(1):267–288.

[29] Keras. <https://keras.io/>

[30] Higham, C. F., Higham, D. J. (2019). Deep Learning: An Introduction for Applied Mathematicians. *SIAM Review. Vol. 61, No. 4, pp. 860-891*.

[31] *Deep Learning with Python* (2017) Chollet, F., Manning Publications, 1st Edition.

[32] Keles, D., Scelle, J., Paraschiv, F., Fichtner, W. (2016). Extended forecast methods for day-ahead electricity spot prices applying artificial neural networks. *Appl. Energy 2016, 162,* 218–230, doi:10.1016/j.apenergy.2015.09.087.

[33] *Understanding LSTM Networks*. (2015), Olah, C., colah's blog., accessed 3rd August 2020: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

[34] Ugurlu, U., Oksuz, I., Tas, O. (2018). Electricity Price Forecasting Using Recurrent Neural Networks.

[35] pandas. <https://pandas.pydata.org/about/>.

[36] datetime – Basic date and time types. <https://docs.python.org/3/library/datetime.html>.

[37] statsmodels. <https://www.statsmodels.org/stable/index.html>.

[38] scikit-learn. <https://scikit-learn.org/stable/>.