Politecnico di Milano

FINANCIAL ENGINEERING

A.Y. 2023/2024

Final Project: Energy Price Forecasting

Group 11

Authors:

Sara Cupini
Davide Pagani
Francesco Antonio Panichi

Contents

1 Abstract

Since the inception of competitive power markets, electricity price forecasting (EPF) has gradually become a fundamental process for energy companies' decision making mechanisms. While the bulk of research has concerned point predictions, in recent years the uncertainty of future supply, demand and prices has increased. As a natural consequence, probabilistic electricity price forecasting is now more important for energy systems planning and operations than ever before.
In this paper we explore both statistical models and neural networks, and propose an interval prediction model.
This body of work stems from a context of competition among peers: 5 groups of students all present a model of their choosing, in the hopes of outperforming the others when predicting the prices for an entirely new, year-long test set. The metric of judgement is the *Delta Coverage*, and no limits are posed with respect to the kind of model that can be implemented. In order to prepare for such competition we are given a temporary test set, made up of the year 2018, on which we can evaluate various models, and ultimately choose our final strategy.

2 Introduction

The first step of our project is to analyze the data, to fix any issues regarding missing/unreasonable values. In section 4, we look for correlations among the variables, and possibly some patterns that indicate seasonality components. These analyses are exclusively conducted on the training set (which covers the period from 3-1-2015 to 31-12-2017) to avoid overfitting on the test set. In section 5 we analyze the effect that 3 scalers have on our data, and choose the best performing one to use in the following models.

Then, we present the evaluation metrics in section 6 that will be applied to the forecasts for 2018, the year corresponding to our test set. After, we explain the two approaches to interval prediction: the first consists in regression directly on the quantiles; the second on prediction of the median, and a subsequent construction of the intervals using the conformal prediction algorithm.
A remark is also made on their behaviour with respect to the main evaluation metric, the delta coverage.

At this point we introduce a simple benchmark (section 8), which we aim to outperform with the final model.

The first family of models considered is that of neural networks. In section 9 we talk of NN algoritms that directly produce prediction intervals, while in section 10 we consider point prediction NN, both of the dense and recurrent type. We go more in depth on the models 'Simple RNN', 'LSTM' and 'GRU' in the subsection 10.1.

At this point (section 11) we introduce a new way of handling the dataset and the prediction: an hourly division, and a model for each hour of the day. We motivate the reasoning that led us to this interpretation and use it in the next models.

Then, in section 12, we analyze the perfomance Triple Exponential Smoothing algorithm. The following sections are all variations of autoregressive models: a basic ARX model in section 13; a Threshold-ARX that considers 'spiky-regimes' in section 14; models with moving average and integration terms, and seasonal components in section 15; a model with preprocessing of the outliers of the target variable in section 16.

In section 17, we present three different ensemble techniques to combine individual point forecasts that make the final prediction more robust. Finally, in section 18 we motivate our choice for the final model and show its performance on the current test set.

3 Day Ahead Prediction

The core of this project is the matter of 'Day Ahead' prediction: the forecasting is made on the electricity prices relative to the 24 hours following the training data, given as known information. So even though we have a test set of a whole year, we construct models that are calibrated on a given set and predict only the next 24 values, effectively having many little one-day test sets.

The prediction for the 1^{st} of January 2018 is made using the entire available training set. Then to predict the 2^{nd} of January we shift the training set forward, removing the oldest day and inserting the true values of the electricity price observed on the 1^{st} of January. We then recalibrate the model on this new training set and forecast the following day.
This procedure is repeated for the entire test set.

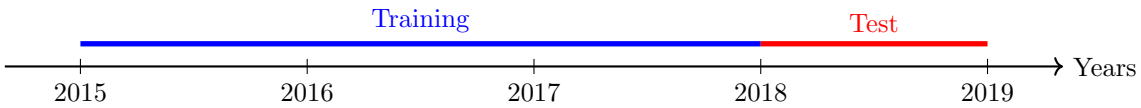
4 Data Analysis

Our dataset contains 35016 observations, which are hourly recordings from 03-01-2015, to 31-12-2018. It consists of 14 features and a target variable representing the hourly energy price in Euro of a MWh (TARG_EM_price) . Specifically, these are the variables of our dataset:

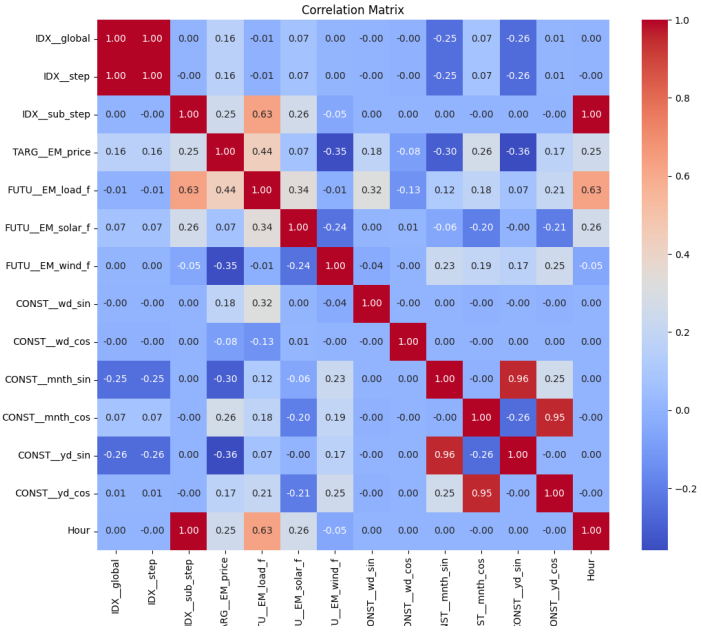
- **IDX_global**: index that represents the global observation number.
- **IDX_step**: index that represents the observation day number.
- **IDX_sub_step**: index that represents the hour of the considered observation.
- **TARG_EM_price**: the target variable. This is the price in Euro of a MWh of the considered observation.
- **FUTU_EM_load_f**: prediction of the energy load computed in the past.
- **FUTU_EM_solar_f**: prediction of the solar intensity computed in the past.
- **FUTU_EM_wind_f**: prediction of the wind intensity computed in the past.
- **CONST_wd_sin**: weekly calendar variable.
- **CONST_wd_cos**: weekly calendar variable.
- **CONST_mnth_sin**: monthly calendar variable.
- **CONST_mnth_cos**: monthly calendar variable.
- **CONST_yd_sin**: yearly calendar variable.
- **CONST_yd_cos**: yearly calendar variable.
- **Hour**: represents the hour.
- **Date**: represents the date.

Before using the dataset, we search for NaN values, and if found, we replace them with the mean of some neighbouring values or, in the case of an observation with too many null values, the row is removed. We make this control with **'df.isnull().sum()'** in Python. It can be observed that there are no problematic values.

Subsequent analysis on the dataset will be performed only on the training set part, which goes from 03-01-2015 to the end of 31-12-2017.



Firstly, let's evaluate the correlation among the variables in the dataset. The correlation coefficients are collected in the matrix below:

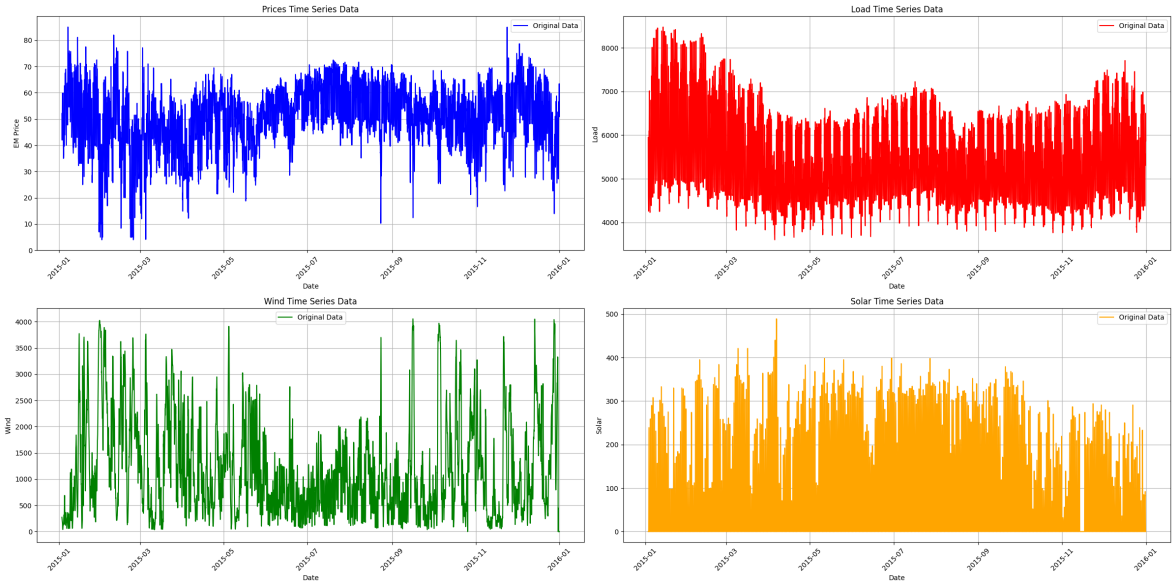


We focus on the relation between the target variable and the features. We observe a strong positive correlation (0.44) with the load feature. This makes sense from an economic point of view: when the (predicted) load is high, the demand for energy is high, thus spiking its price.
Another coefficient to analyze is the one corresponding to the wind variable. It's -0.35 since with an increase in wind activity more energy is produced from wind turbines, which is less expensive, thus causing the general energy market price to drop.
Moreover, we see that there are some highly correlated features (over 95%) like CONST__mnth_sin and CONST__yd_sin. So, we could reduce the dataset by removing one of them since they have a similar trend.

We then evaluate the mean and the standard deviation for each column, except 'Hour', Date' and the IDX variables. We show the results in the table below.

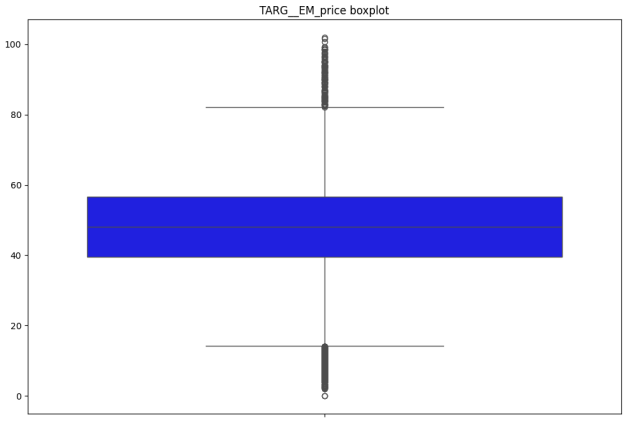
Mean and standard deviation for each column		
Variable	**Mean**	**Standard Deviation**
TARG__EM_price	47.44	14.23
FUTU__EM_load_f	5623.92	965.96
FUTU__EM_solar_f	93.75	121.80
FUTU__EM_wind_f	1304.06	982.68
CONST__wd_sin	-0.0016	0.7075
CONST__wd_cos	0.0004	0.7067
CONST__mnth_sin	-0.0055	0.7061
CONST__mnth_cos	-0.0035	0.7080
CONST__yd_sin	-0.00003	0.7074
CONST__yd_cos	-0.0009	0.7068

Afterwards, we also plot the evolution in time of the price, load, wind and solar time series. We chose to display only an year worth of observations to have easier to interpret graphs. We notice that the wind and solar features are the ones that have the more spiky behaviours, with the latter showing an high number of zeros values, which probably mean hours without solar activity.

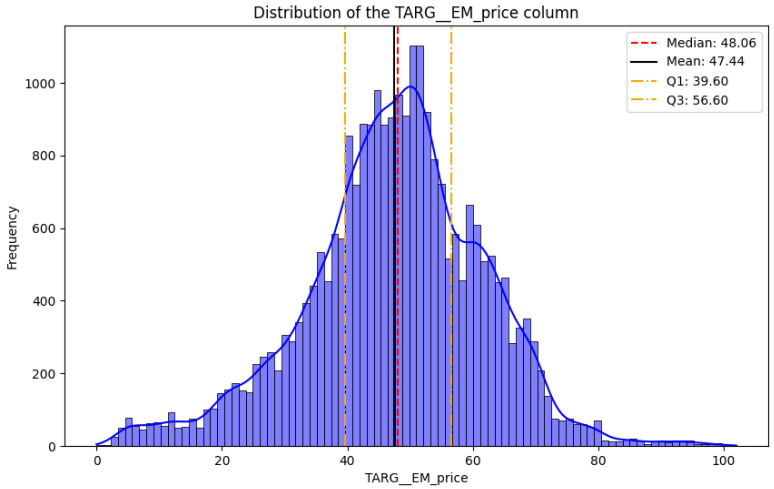


Coming back to the table above, due to the fact that the standard deviation is often bigger than the mean and the average values are very different, we'll preprocess our data to transform all the features to a similar scale.

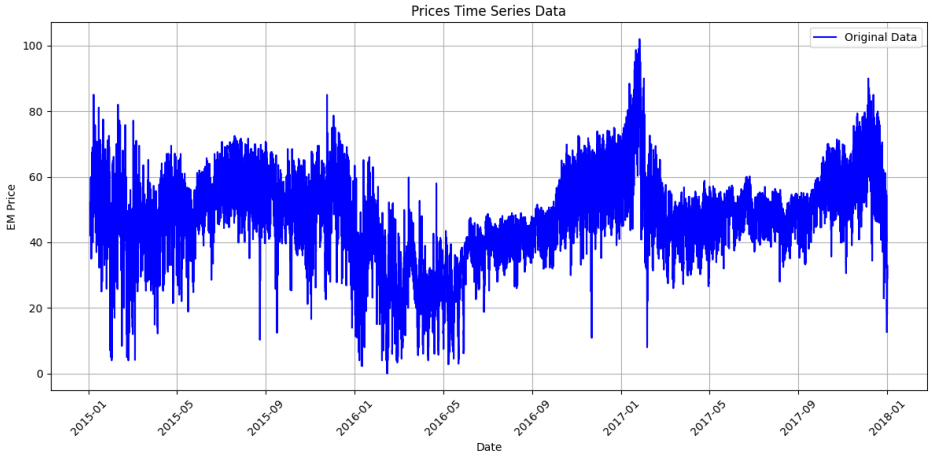
We now start an analysis on our target variable, the 'TARG__EM_price' column. Firstly, we study the outliers and we find that they are 3% (794 observations).
We defined the outliers with the interquartile rule. We compute the IQR as difference between third and first quartile (IQR = Q_3 - Q_1), and then the outliers are the values found outside of the range [Q_2 - 1.5 $*$ IQR, Q_2 + 1.5 $*$ IQR], with Q_2 being the median. We can show graphically the results using the box-plot below.



To continue our study of the target variable, we plot here below an histogram of the discretized distribution of the price. While we could be tempted to say it looks like a Gaussian shape, the left tail is definitely heavier than the right one and than a typical Gaussian tail, and the distribution is not symmetrical, but is skewed to the left. Moreover, we observe some really high prices, around the 90 euros mark. These high outliers are referred to in the literature as spikes, and because of their uncommon values they contribute to the complexity of EPF.



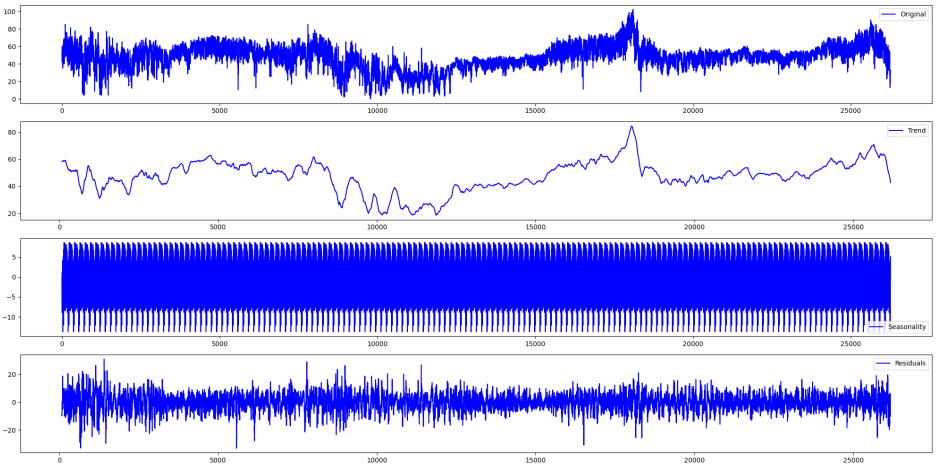
After that, let's analyze the price over the training set period.



To go into details of the prices time series, we try to perform a seasonal decomposition thanks to the **'seasonal_decompose'** Python function. We assume an additive decomposition (see [9]), with the following formula:

$$Y_t = S_t + T_t + R_t$$

where S_t is the Seasonal term, T_t is the Trend term and R_t is the Residual term, what is left of the time series after removing the two other terms. As input of the seasonal decomposition function, we assume a period of seven days, as can be expected.



We can observe a significant trend component, while the seasonality plot is too dense to be commented. The residuals look stationary, but we hold some doubts on this and we will check it better below.

The important check to be done now is about the stationarity of the target variable. Indeed, this is a theoretical assumption of most statistical models, like the auto-regressive model and more generally the whole SARIMAX model family. We use two different statistical test to verify the stationarity of our data: ADF and KPSS.

The null hypothesis of the Augmented Dickey-Fuller (ADF) test is that the time series has a unit root, that is, it has a time-dependent structure and is non-stationary. From the resulting p-value, we can reject the null hypothesis and thus this test declares that the time series is stationary.

The Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test checks whether the time series is stationary around a mean or linear trend. Contrary to ADF, its null hypothesis is that the time series is stationary. From the KPSS p-value, we see that this test is suggesting that the time series in non-stationary.

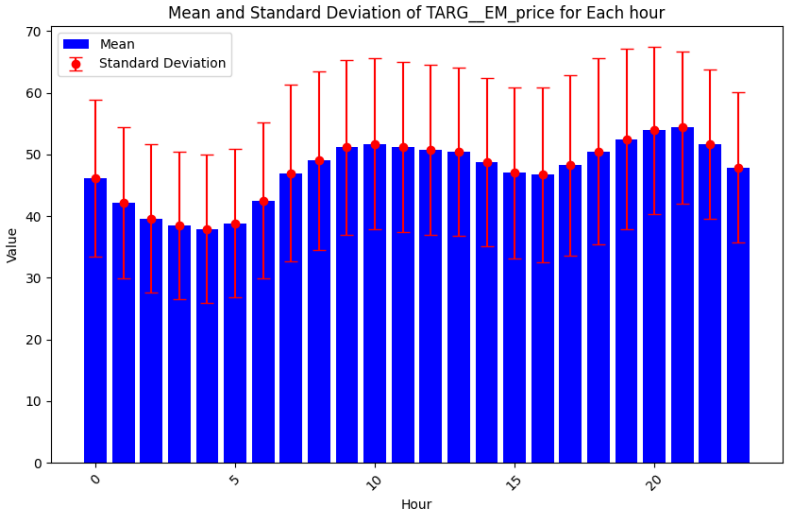
Statistical stationary test for the target		
Statistical Test	**H0**	**P-Value**
Dickey-Fuller	non-stationary	3.93×10^{-12}
KPSS	stationary	0.01

The two statistical tests therefore output opposite results: in this case it typically means that the time series is "difference-stationary", and so it could require differencing to become stationary.
We will further comment on this problem in the following sub-section, where we will analyse the stationarity of the individual hourly time series.

4.1 Analysis of the hourly price time series

We now shift the focus of our analysis to the single hourly time series of the energy price. We therefore split the dataset in 24 different sub-datasets.
This can give us further insights on the problem, as the energy price can have different dynamics in different times of the day, and it can help us select the models accounting more carefully for daily seasonalities.

To start, we compute the mean and standard deviation of the target for each hour and we show them in the plot below:



We can notice that the division made is reasonable. Indeed, mean prices vary during the day, mainly due to greater demand in the middle of the day or during the evening.

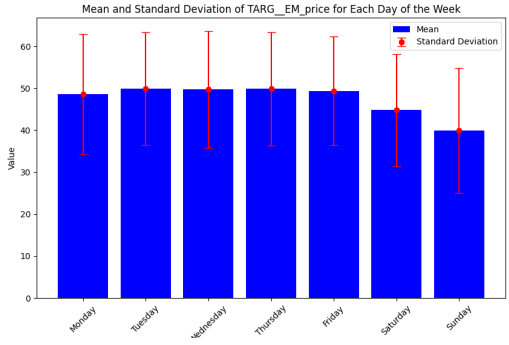
We now check the stationarity of the 24 hourly time series with the same tests as above, obtaining interesting results. Indeed, for (almost) every hour the ADF test is suggesting non-stationarity of the data, as is the KPSS.

This prompts us to perform a differencing of the each single price time series to obtain stationarity. Verifying again with the statistical tests, we see that for all the differenced time series the stationarity hypothesis can now be accepted. Therefore, if we work with a framework that builds independent models for each hour of the day, we will then probably need to perform differencing on the target variable, prior to forecasting.
On the other hand, when building models that do no split the price time series, or neural networks models, which do not need the stationarity assumption, we will probably not need differencing.

4.2 Analysis of the daily price time series

We now split the dataset into seven sub-datasets that each have all the observations for a single day of the week. This can help us better understand the price dynamics in different days of the week, and the weekly seasonalities.
Here below, we show a quick statistical analysis of the energy price over different days of the week.

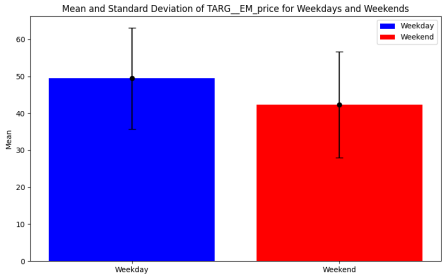
Mean and std dev for each day		
Weekday	**Mean**	**StdDev**
Monday	48.56	14.33
Tuesday	49.92	13.50
Wednesday	49.70	13.96
Thursday	49.86	13.56
Friday	49.34	12.95
Saturday	44.79	13.33
Sunday	39.97	14.89



It is clear that Sundays have a much lower price for energy and also a bit of a bigger standard deviation. Saturdays are somewhat a middle ground, while there is virtually no difference between the weekdays, except for a higher than average standard deviation on Mondays.

Taking inspiration from this last remark, let's divide our dataset into weekdays and weekend days and redo the computation on the target columns.

Mean and std dev for each day type		
Day Type	**Mean**	**StdDev**
Weekday	49.48	13.67
Weekend	42.38	14.34



From this final analysis, we observe that it could be an idea to use a dummy variable to distinguish weekdays from the weekend to replace the features 'CONST__wd_cos' and 'CONST__wd_sin', making our model lighter.

5 Preprocessing

Data preprocessing is the process of transforming raw data into a useful, understandable format, in order to optimize the efficiency of data science algorithms.
If there are input variables that have very large values relative to the other ones, these large values can dominate or skew the weights of some predictive algorithms. The result is that the algorithms pay most of their attention to the bigger values and tend to ignore the other variables.
Therefore, we scale input variables to a common range as a data preparation technique prior to fitting a model. We remark that the transformation on the test set is performed through the same scaling parameters computed on the training set.

We compare the effect on our training set of three popular scalers: *Standard Scaler*, *MinMax Scaler* and *Robust Scaler*.
StandardScaler transforms the data so that it has a mean of 0 and a standard deviation of 1. It assumes that the distribution of the features is Gaussian and does not handle outliers well.
MinMaxScaler scales features to a fixed range, like [0,1]. It's commonly used when the distribution of the data is not Gaussian or when features have different units and scales.
RobustScaler scales features using the interquartile range, so it's robust to outliers.

$$x_{\text{Std}} = \frac{x-\mu}{\sigma} \qquad x_{\text{MinMax}} = \frac{x-x_{\min}}{x_{\max}-x_{\min}} \qquad x_{\text{Robust}} = \frac{x-median(x)}{Q3(x)-Q1(x)}$$

Here reported are two metrics that guided our decision making process, namely the standard deviation and the interquantile range, for the main variables representing the energy prince, the wind and solar activity and the predicted load.

Comparison of Different Scalers					
Scaler	**Metric**	**Price**	**Load**	**Solar**	**Wind**
Original Data	Std Dev	14.23	965.95	121.80	982.66
	IQR	17.00	1589.00	194.00	1364.25
Standard Scaler	Std Dev	1.00	1.00	1.00	1.00
	IQR	1.19	1.65	1.59	1.39
MinMax Scaler	Std Dev	0.14	0.18	0.23	0.22
	IQR	0.17	0.30	0.37	0.30
Robust Scaler	Std Dev	0.84	0.61	0.63	0.72
	IQR	1.00	1.00	1.00	1.00

In the end we choose to move forward with the *MinMax Scaler*, as it was the one with the lowest standard deviation on the training set.

6 Evaluation Metrics

We present here a number of evaluation metrics that can be used in EPF, to evaluate point predictions and interval predictions.

6.1 Competition Metric: Delta Coverage

Delta Coverage is an evaluation metric for probabilistic forecasting. It determines the fraction of observations that fall outside the confidence interval (CI) within a given nominal level α.
Let LB_t and UB_t be the lower and upper bounds for a given α-CI and y_t the actual consumption at time t, the indicator I_t takes two values: 1 if the actual consumption falls in the predicted CI and 0 otherwise.
The empirical coverage EC_α is the Out-Sample mean of the indicator for a given nominal α. In practice, the closer EC_α is to the nominal level, the better it is.
More in detail, the formula of this error metrics is the following:

$$\Delta C = \frac{1}{\alpha_{max} - \alpha_{min}} \sum_{a=\alpha_{min}}^{\alpha_{max}} |EC_a - a| \qquad \alpha_{min} = 90\%,\ \alpha_{max} = 99\%$$

To avoid over-fitting our model on this measure and to have a clearer vision of its behaviour, we also look at other metrics.

6.2 Additional Metrics: Point Forecasting

- **Mean Absolute Error (MAE)**

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

- **Root Mean Squared Error (RMSE)**

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **Symmetric Mean Absolute Percentage Error (sMAPE)**

$$\text{sMAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{\frac{|y_i|+|\hat{y}_i|}{2}} \times 100$$

6.3 Additional Metrics: Interval Forecasting

- **(Average) Winkler Score**
  The Winkler's score is a scoring rule that judges the goodness of prediction intervals estimated in a probabilistic forecasting exercise. It is defined as:

$$\begin{cases} \delta_n (= \hat{U}_n - \hat{L}_n) & \text{if } y_n \in [\hat{L}_n, \hat{U}_n] \\ \delta_n + \dfrac{2}{1-\alpha}(\hat{L}_n - y_n) & \text{if } y_n < \hat{L}_n \\ \delta_n + \dfrac{2}{1-\alpha}(y_n - \hat{U}_n) & \text{if } y_n > \hat{U}_n \end{cases}$$

  Its formula is such that it rewards narrow prediction intervals, as can be seen by the component δ_n, and, in case of an observation that misses the interval, gives a penalty which depends from α and the distance from the interval. Therefore, when we compare the Winkler score for a given quantile level between two different models, we prefer to have a lower score.
  We will then also compute the mean over the different hours and different quantile levels, to obtain a single value which is easier to compare between models.

- **(Average) Pinball Loss**
  The Pinball loss measures the discrepancy between the predicted quantiles and the actual values of the true data. To compute it, the following formula is used:

$$P(\alpha, \hat{q}_{\alpha,n}, y_n) := \alpha[y_n - \hat{q}_{\alpha,n}]\mathbb{1}_{(y_n \geq \hat{q}_{\alpha,n})} + (1-\alpha)[\hat{q}_{\alpha,n} - y_n]\mathbb{1}_{(y_n < \hat{q}_{\alpha,n})}$$

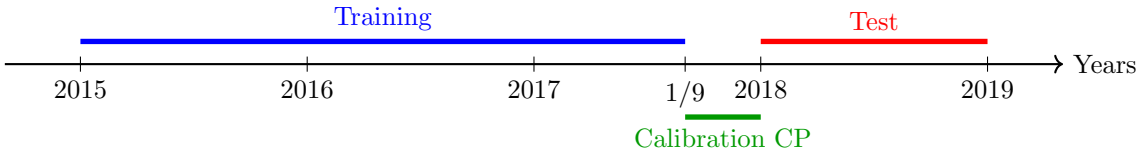
  where $\hat{q}_{\alpha,n}$ is the quantile of level α and y_n is the true value of the target feature. The Average Pinball Loss is the mean of the Pinball Loss over the set of predicted quantiles.
  We will then also compute the mean over the different hourly average pinball losses, to obtain a single value which is easier to compare between models.

7 Approaches to the Prediction Intervals

7.1 Conformal Prediction: method

Conformal Prediction is a powerful algorithm that allows data scientists to quantify the uncertainty of predictive models. It is designated to predict intervals based on errors, features weak assumptions on data characteristics and is versatile with regards to the underlying point prediction model [1].
Conformal Prediction yields valid prediction intervals that meet the designated confidence level $1-\alpha$, defined by the user.

First, we begin with a fitted point prediction model which we will call $\hat{f}$. Then, we will create prediction sets using a small amount of additional calibration data, equal to the last 4 months of the training set.



Using $\hat{f}$ and the calibration data, we seek to construct a prediction interval $\mathscr{C}(X_{\text{test}})$ that is valid in the following sense:

$$1-\alpha \;\leq\; \mathbb{P}(Y_{\text{test}} \in \mathscr{C}(X_{\text{test}})) \leq 1-\alpha+\frac{1}{n+1}, \tag{1}$$

where $(X_{\text{test}}, Y_{\text{test}})$ is a new test point from the same distribution, n is the cardinality of the calibration bag, and $1-\alpha \in [0,1]$ is the wanted confidence level.
In other words, the probability that the conformal prediction interval contains the correct label is

almost exactly $1-\alpha$; we call this property *marginal coverage*.

To construct $\mathscr{C}$ from $\hat{f}$ and the calibration data, we perform a simple calibration step.
First, we set the Conformal Score $S_t = |y_t - f(x_t)|$ to be the absolute error. This measures the test samples conformity with respect to the calibration bag, and it is high when the prediction is wrong.

Then, we sort the conformity scores in ascending order: $S_{(1)} < ... < S_{(k)} < ... < S_{(n)}$. Looking at the empirical distribution of conformity scores, we know that:

$$\mathbb{P}(S_t < S_{(k)}) = \mathbb{P}(S|y_t - f(x_t)| < S_{(k)}) = \mathbb{P}(y_t \in f(x_t) \pm S_{(k)}) = \frac{k}{n}$$

Then, we take $S_{(k)}$ from the empirical quantiles of ranked conformity scores as:

$$S_{(k)} = \begin{cases} S_{\lceil (n+1)(1-\alpha)/n \rceil} & \text{if } \lceil (n+1)(1-\alpha)/n \rceil \leq n \\ \infty & \text{otherwise} \end{cases}$$

This way, we can obtain a prediction interval of any wanted level $1-\alpha$, as $\mathscr{C}(X_{\text{test}}) = f(X_{\text{test}}) \pm S_{(k)}$.
Remarkably, this algorithm gives prediction sets that are guaranteed to satisfy property 1, no matter what point prediction model is used or what the distribution of the data is.

The only hypothesis made on the data is that of *exchangeability* for the distribution: the joint probability distribution of a sequence of random variables remains unchanged under any permutation of the indices.

7.2 Conformal Prediction and Quantile Regression: a comparison

We now present another method for interval prediction, briefly explaining its main characteristics.
Then we compare its performance and that of the Conformal Prediction algorithm with respect to the competition's evaluation metric, the Delta Coverage.

Quantile regression estimates conditional quantiles of the response variable given certain values of predictor variables. Instead of predicting a single mean value, it predicts a specified quantile (e.g. the 10^{th}, 50^{th}, or 90^{th} percentile).
This is done by minimizing a loss function that asymmetrically penalizes over -and under- predictions. The loss function is typically a quantile loss function, such as the pinball loss, which penalizes deviations of the predicted quantiles from the actual quantiles.

Conformal prediction requires the exchangeability of data points, which is a less strict assumption compared to the i.i.d. assumption required by quantile regression.

Conformal prediction is a valid interval prediction method, as it focuses on ensuring that the constructed prediction intervals achieve a specified coverage probability.
Quantile regression, on the other hand, is an adaptive interval prediction method, as it adjusts the width of the prediction intervals based on the uncertainty or variability in the data. Adaptive methods aim to provide more informative and potentially narrower intervals where the model is more confident and wider intervals where there is higher uncertainty.

Since our main evaluation metric is the Delta Coverage, which favours prediction intervals that match their expected nominal coverage, valid methods have a significant advantage because of the way they build the prediction intervals.
For this reason we will concentrate our efforts into developing a robust point prediction algorithm, which will then be combined with conformal prediction to obtain the probabilistic forecast we are looking for.

8 Naïve Benchmark

A benchmark model is a baseline or reference model used to compare the performance of other models, aiding in the evaluation of the effectiveness and efficiency of new models or algorithms.

The objective is to implement models that surpass the benchmark in accuracy of the prediction.

As benchmark for point predictions, we implement a naïve method: the output price for a given hour of the test day is the mean of the prices relative to that same hour of the seven preceding days.

After applying the conformal prediction approach to the point predictions, here is the evaluation of this model's performance on the test set:

Naïve Benchmark + CP					
Delta Coverage	**RMSE**	**MAE**	**sMAPE**	**Pinball score**	**Winkler score**
0.6963	7.2877	5.0353	1044.9552	0.6390	35.3035

9 Interval-based Neural Networks

9.1 Distributional Models

During the laboratories we analyzed two models for probabilistic price forecasting, both based on distributional neural networks.
The model structure is based on a deep neural network that contains a so-called probability layer. The network's output is a parametric distribution with 2 or 4 parameters for Normal or Jhonson's SU distribution respectively. After estimating these parameters, we get the prediction intervals from the quantiles of the distribution.

We have just investigated the distribution of the target variable **'TARG_EM_price'** graphically in the section 4. One could be tempted to assume that we are dealing with Gaussian data, but given the rugged shape of the downward bell, wee see fit to perform a Shapiro test. The output p-value is equal to $5.328 \cdot 10^{-39}$, meaning that we have strong statistical evidence to refuse the Gaussianity hypothesis.

Since there aren't distributions that fit well our data, we abandon this approach.

9.2 Quantile Regression

Quantile regression neural networks can approximate multiple quantiles directly, bypassing the need to select and calibrate a specific distribution for the data and then extract quantiles from this distribution. Indeed QRNNs learn the quantiles as part of the training process.

Unlike traditional neural networks where the output layer typically predicts a single value (e.g., the mean), in QR-DNNs, the output layer predicts multiple quantiles of the response variable. Each neuron in the output layer corresponds to a specific quantile level.
As anticipated in section 7.2 quantile regression uses a loss function, which in our case is the (average) Pinball Loss, defined below:

$$\sum_n \sum_h \alpha \, [y_n^h - \hat{q}_{\alpha,n}^h] \, \mathbb{1}_{(y_n^h \geq \hat{q}_{\alpha,n}^h)} + (1-\alpha) \, [\hat{q}_{\alpha,n}^h - y_n^h] \, \mathbb{1}_{(y_n^h < \hat{q}_{\alpha,n}^h)}$$

During the training phase, the network adjusts its weights and biases to minimize the quantile loss function, effectively learning to predict the conditional quantiles of the response variable given the input features.

The code for this model was presented in class. By configuring the model with a hidden size of 512 and 2 hidden layers, each using the *softplus* activation function, we obtained the following prediction results:

QR-DNN					
Delta Coverage	**RMSE**	**MAE**	**sMAPE**	**Pinball Score**	**Winkler Score**
7.4036	5.1834	3.8564	827.7023	0.4232	17.2621

As anticipated in subsection 7.2, the Delta Coverage is significantly high, much greater than that of the subsequent models. This supports our decision to focus on integrating point prediction with the Conformal Prediction algorithm.

10 Point Neural Networks

Our starting point is the Dense Neural Network model as seen in class, i.e. a multi-layer perceptron in which the information propagates only forward.
We chose to proceed with 512 as hidden size, 2 hidden layers, each with *softplus* activation function. These hyperparameters are chosen empirically, through a critical reading of the outputs of Optuna tuner.
After applying the conformal prediction approach to the point predictions, here is the evaluation of this model's performance on the test set:

point-DNN + CP					
Delta Coverage	**RMSE**	**MAE**	**sMAPE**	**Pinball Score**	**Winkler Score**
0.3196	5.6383	4.2246	907.9141	0.486459	25.3138

10.1 Recurrent Neural Networks

Aware of the limitations of our computers with respect to hardware, we still try to implement some simple Recurrent Neural Network models.

Recurrent neural networks, or RNNs for short, are a variant of the conventional feedforward artificial neural networks that can deal with sequential data and can be trained to hold knowledge about the past.
Ordinary feedforward neural networks are best suited for data points that are independent of each other. However, if we have data in a sequence such that one data point depends upon the previous data point, as in the case of electricity price and time series in general, RNNs are recommended since they use the information of previous inputs to generate the next output of the sequence.

Here follows a quick introduction on the implemented models:

- **Simple RNN**: in keras the *SimpleRNN* layer implements a Fully-connected layer of neurons where the output is to be fed back as the new input, hence the Recurrent adjective.
- **LSTM**: LSTM stands for Long Short Term Memory, and it's a special type of recurrent layer, implemented in keras. The basic idea of the LSTM architecture is to create an additional module in a neural network that learns when to remember and when to forget pertinent past information.
- **GRU**: GRU stands for Gated Recurrent Unit, which is a type of recurrent neural network (RNN) architecture that is similar to LSTM (Long Short-Term Memory), but with a simpler structure.

We create new classes of regressors for each of these models, and modify the function *¨build˙model¨* to change the layers from *tf.keras.layers.Dense* to *tf.keras.layers.SimpleRNN*, *tf.keras.layers.LSTM*, and *tf.keras.layers.GRU* respectively. We also integrate these new classes into the main structure of the recalibration engine.

Regarding the parameters to give in input to these layers, we limit our analysis to the choice of the activation function, which introduces non-linearity into the network, and the rate of dropout, which is the proportion of neurons that are randomly deactivated in a layer.
We choose the iperbolic tangent as activation function, and dropout rate equal to 20% as we don't have many neurons to start with (we keep dimensions low due to hardware limitations).

Before considering a run on a whole year, we choose to run our models in 3 very different months (April, August and December) to have an idea on the behaviour and to choose hyperparameters. For comparison we run the point-DNN on those same months.

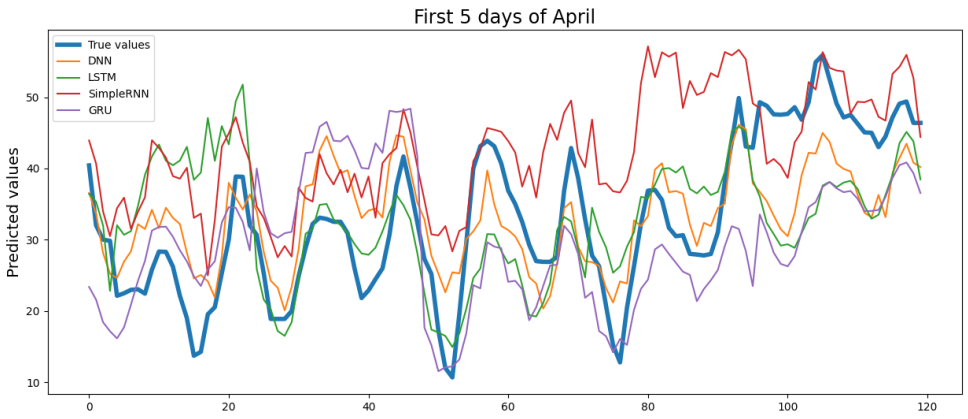
The chosen hyperparameters are: hidden size as 128, 2 hidden layers and learning ratio as 10^{-3} for both LSTM and GRU, while we increase the number of hidden layers to 2 for SimpleRNN. The main reason for this choice is the computational effort required when increasing the number of hidden layers and the hidden size.
Below we report the principal metrics for point prediction (MSE, MAE, sMAPE):

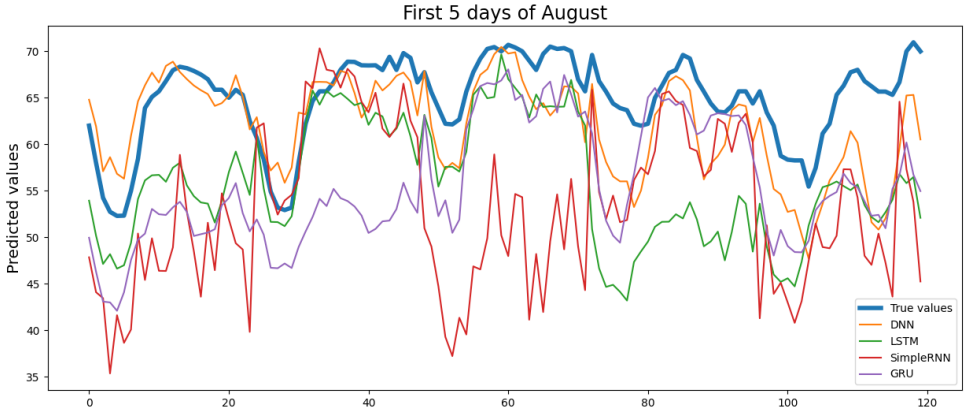
Comparison of Different Models - April			
Model	**MSE**	**MAE**	**SMAPE**
DNN	39.1467	5.0122	821.2977
LSTM	65.2459	6.0976	993.3849
Simple RNN	63.5889	6.0356	999.2155
GRU	71.5056	6.6039	1054.8995

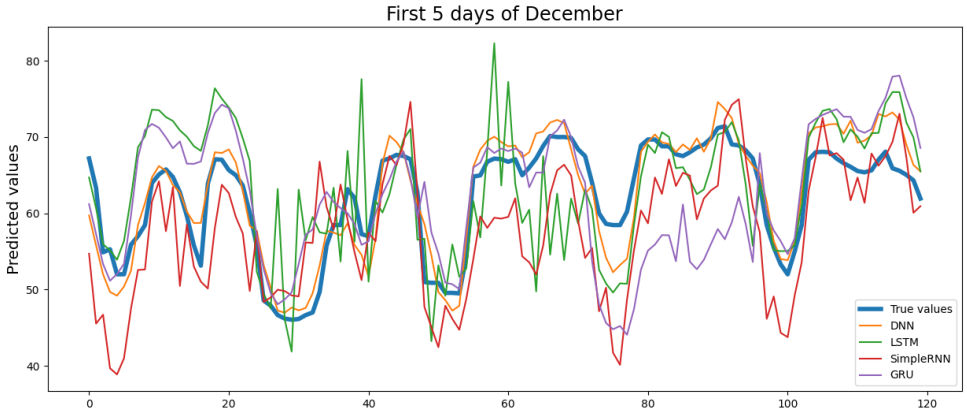
Comparison of Different Models - August			
Model	**MSE**	**MAE**	**SMAPE**
DNN	14.8599	3.0288	759.5979
LSTM	94.8390	8.1854	1949.2487
Simple RNN	87.7127	7.1438	1690.7808
GRU	115.7570	9.2699	2175.4820

Comparison of Different Models - December			
Model	**MSE**	**MAE**	**SMAPE**
DNN	12.6910	2.8573	697.5377
LSTM	46.3569	5.1871	1275.5621
Simple RNN	45.6517	5.4299	1274.5642
GRU	42.2083	4.9227	1205.8628

For a quick visualization, we also plot the first 5 days worth of predictions for each month.







Given the disappointing results, and the great computational time we abandon these RNN models, and venture into simpler statistical approaches, starting from a naive autoregressive model.

A final remark should be made on the suitability of Recurren Neural Networks in our context. RNN work best on great amount of data (order of magnitude of the tens of thousands), as otherwise the network rather memorizes the input, and leans towards overfitting.
Since we only have a 3-year time series these models don't perform to the best of their ability.

11 Hourly Division

All the models introduced thus far are set up to perform a daily prediction: the test set is always consisting of one day, and after the calibration on the training set, a prediction is made for 24 hours.

An interesting property of the energy price, is that it varies hour by hour in a consistent way between days. For instance, prices at midday behave differently than prices at midnight, but behave similarly to midday prices of other days. The reason is the strict correlation between energy price and energy demand, higher during the day and lower at night.
This holds true also for our specific dataset, as shown in section 4.1. Motivated by this analysis, we want to change the current approach to one that is better capable of recognising such patterns.

A first idea is to separately model the prices of daylight and nighttime hours. This is however quite complex, as the amount of daylight hours changes during the year and depending on the geographical location, which is unfortunately unknown to us.

Instead, we move towards an 'hourly' approach: we split the dataset into 24 sub-datasets, each containing all the data referring to a specific hour, and on each sub-dataset we calibrate a model to predict the price in the corresponding hour of the next day.

Clearly, such an approach requires 24 as many calibrations as a 'daily' model, and is therefore much more demanding from a computational point of view. This is the reason why some models, such as the Neural Network-based ones, were not implemented in a hourly fashion.

12 Holt-Winters Exponential Smoothing

The Holt-Winters method, also known as triple exponential smoothing (TES), is a relatively simple method for time series forecasting, capable of including both trend and seasonality.
This method uses exponential smoothing to encode lots of values from the past, assigning them decaying with time weights, and use them to predict "typical" values for the present and future, without using any exogenous inputs.

We chose to split the dataset into 24 smaller datasets, one for each hour, therefore fitting 24 different models, to optimize the predictions.

In our implementation of the algorithms we make use of the *statsmodels* library, which offers the ready-to-use function *ExponentialSmoothing*. The parameters to be passed are: type of trend component, type of seasonal component and the number of periods in a complete seasonal cycle.

Regarding the first two parameters, we chose 'additive' for both trend and seasonality because as we move forward with the sliding training window, we can get negative values and so the 'multiplicative' cannot be used; indeed, we fit our chosen scaler (MinMaxScaler) to the first training set, and then apply it to the whole test set, making some values negative. As for the number of periods we give 7: we are dealing in fact with daily data with a reasonable weekly seasonality.

After applying the conformal prediction approach to the point predictions, here is the evaluation of this model's performance on the test set:

TES + CP					
Delta Coverage	**RMSE**	**MAE**	**sMAPE**	**Pinball Score**	**Winkler Score**
0.3034	7.2382	5.2747	113.3927	0.6138	33.2411

13 ARX

To improve the prediction of the previous model, we decide to consider an Autoregressive approach, that is similar to exponential smoothing but improves the forecast on a short-term prediction horizon. This technique models a time series so that the current value is a linear combination of the past p values of the time series and of the white noise ϵ_t.
Using these past lags of data, we are able to take into account the random nature and time correlations of the phenomenon under study [10].

Another improvement is adding the contribution of exogenous variables to the structure of our model. Indeed, electricity prices are influenced by external present-day factors, such as the consumers' energy demand and the intensity of solar and wind activity [10].
We will therefore add all the exogenous features present in the given dataset: the **'FUTU'** columns contribute to add explanatory value as drivers of the new price creation, and the **'CONST'** columns can help the model learn short -or long- term seasonality patterns.

The ARX implementation can be described by the following formula:

$$y_t = \sum_{n=1}^{p} \alpha_n y_{t-n} + \sum_{n=1}^{r} \beta_n x_{nt} + \epsilon_t$$

To implement this approach, we exploit the code presented in class. Moreover, we add to the Optuna routine the calibration of the hyper-parameter 'steps_lag_win', which in the ARX case represents the autoregressive lag p. The other two hyper-parameters for this model are the learning ratio of the gradient descent and the Lasso L1 regularization term for the model's weights.
However, for their choice, we do not simply rely on the best individual Optuna result, we rather critically analyse the parameter sets that look more promising from the Optuna calibration, taking also advantage of the studies done during the previous laboratories.

As learning ratio, we set lr = 0.001, to have a trade-off between accuracy and fast convergence of the method. For the regularization term, we choose L1 = 1e-5 to try to avoid under- and over-fitting. While for the 'steps_lag_win' we have 7, after considering the Optuna results and the PACF plot, which is further commented in section 15.1.

After applying the conformal prediction approach to the point predictions, here is the evaluation of this model's performance on the test set:

ARX + CP					
Delta Coverage	**RMSE**	**MAE**	**sMAPE**	**Pinball Score**	**Winkler Score**
0.2102	9.1641	7.2359	1626.9761	0.7328	38.8065

13.1 ARX with hourly models

Now we modify this model to make it 'hourly', as explained in section 11, in such a way to keep as much of the original structure as possible.
In practice we first divide the dataset into 24 sub-sets, then with prediction horizon equal to 1 we call on a new Recalibration Engine class, which differs from the original in how it indexes the data.
After obtaining 24 point predictions, we group them in the correct order and proceed with the conformal prediction algorithm to get the prediction intervals. Below is the evaluation of this model's perfomance on the test set.

ARX hourly + CP					
Delta Coverage	**RMSE**	**MAE**	**sMAPE**	**Pinball Score**	**Winkler Score**
0.3358	5.8034	4.1893	885.9225	0.5023	26.7273

We notice that the metrics have values similar to the original point-ARX model, but given the much greater computational effort required to run this new model we decide to abandon it.

14 TARX

An extension to the ARX approach are Threshold AutoRegressive models. They are a subclass of regime-switching models, where the regime can be determined by an observable variable v_t relative to a threshold value [10].
In our case, the spiky regime is to be interpreted as a situation where the energy price is in an increasing scenario. The variable v_t is taken as the difference between the mean price of the last day and the mean price of the preceding eight days, as suggested in the literature [5], and we set the threshold T_0 at 2, signifying a clear upward shift of the price level in the previous day.

For simplicity, instead of implementing two separate models for the two regimes, we decided to add to our exogenous features a dummy variable, called **'FUTU__spiky_regime'**, that should be used from the model to react to a spike in the prices and follow it. The advantage of this implementation is that it allows us to reuse this variable in any model that leverages exogenous variables.

As a first approach, we add this variable to the 'full-day' ARX model of the previous section. Then, after applying the conformal prediction approach to the point predictions, here is the evaluation of this model's performance on the test set:

TARX + CP					
Delta Coverage	RMSE	MAE	sMAPE	Pinball score	Winkler score
0.1857	5.9523	4.5174	987.7118	0.5025	26.4105

15 SARIMAX Model Class

To further explore the class of statistical models for time series prediction, we now focus on including additional components to the ARX structure.

The first component that can be added is the moving average term, incorporating the regression of past q estimation errors to catch more accurately the time correlations of the phenomenon under study [10].

However, the ARMA model approach assumes stationarity in the time series data. Given that we aim to construct hourly models and, as indicated in section 4.1, the hourly price time series exhibit non-stationarity, we must find a way to respect this assumption.
This is where the integrated component comes into play, determining the number of times (d) the series needs to be differenced. Differencing is a technique used to achieve stationarity and, as demonstrated in section 4.1, it is sufficient to set d = 1.

With these components, not forgetting the exogenous variables, we have the ARIMAX class of models, whose order is expressed by the vector (p, d, q).

Moreover, after assuming the period of the time series' underlying seasonality, we can further model the seasonal component with other autoregressive (P), integration (D), and moving average (Q) terms. These inclusions give life to the SARIMAX model family, whose order is expressed by two vectors: (p, d, q) and (P, D, Q, S), with S being the period of seasonality chosen by the user.

Finally, here below is the complete formulation of the SARIMAX models (without considering the differencing orders d and D). Clearly, if we remove the last two summations, the formula refers to the ARIMAX class.

$$y_t = \sum_{n=1}^{p} \alpha_n y_{t-n} + \sum_{n=1}^{q} \theta_n \epsilon_{t-n} + \sum_{n=1}^{r} \beta_n x_{nt} + \sum_{n=1}^{P} \phi_n y_{t-sn} + \sum_{n=1}^{Q} \eta_n \epsilon_{t-sn} + \epsilon_t$$

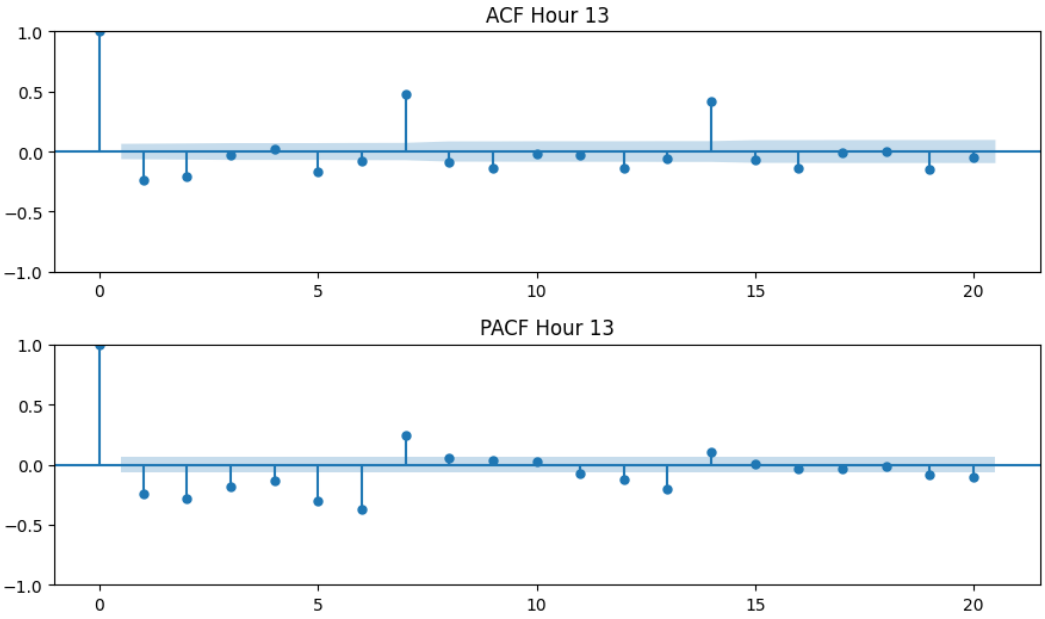
Choosing the order of the model requires an accurate study, following the Box-Jenkins methodology described in the next section.

Moreover, to reduce the model's complexity and to help with the algorithm's convergence, we just use a subset of all the exogenous variables: the forecasted energy load, wind and solar activity.
We also add the weekend dummy variable (see subsection 4.2), to retain some information about the weekdays vs weekend price behaviours, and the spiky regime dummy variable (see section 14), to help the model better understand when the price has an increasing tendency.

15.1 Box-Jenkins Method

The Box-Jenkins methodology is an heuristic method to assess the order of the auto-regressive and moving average components, seasonal and non-seasonal, for all models belonging to the SARIMAX family [2].

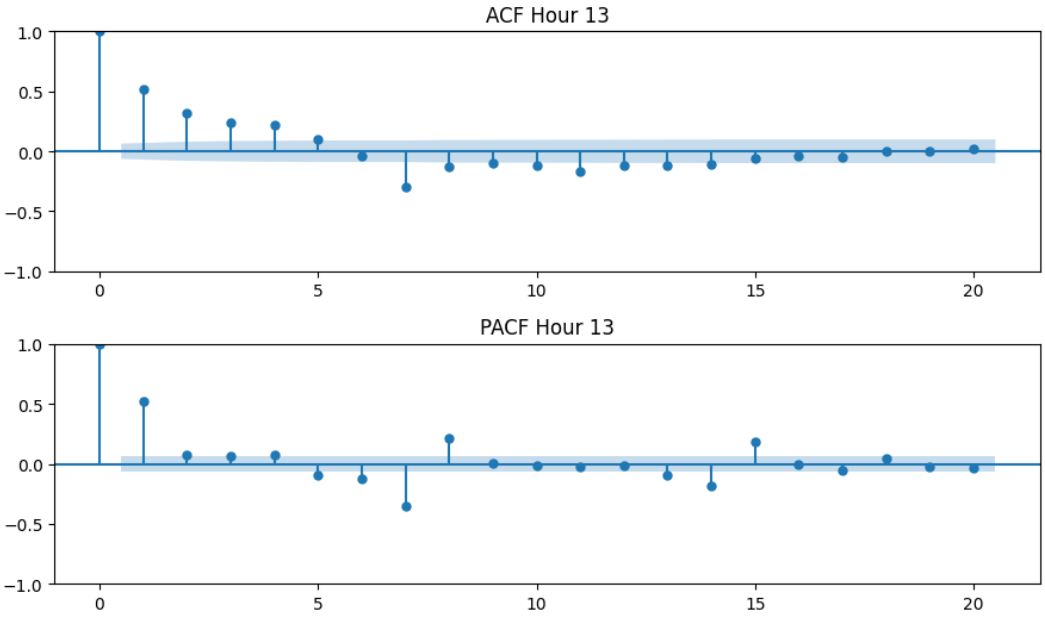
We first focus on the ARIMA components. After having decided the order of integration (aforementioned parameter d=1) thanks to the previously done ADF and KPSS test remarks (section 4.1), we plot the Auto Correlation Function and Partial Auto Correlation Function of the differenced hourly price time series. We show them below for hour 13, which is taken as representative of all the other hours.



Both the ACF and PACF plots tail off after some lags and therefore, according to Box-Jenkins, we choose the model orders looking at the first statistically significant values, always remembering to be parsimonious. So we take an order MA = 2 and AR = 3, to have a good trade-off between model complexity and amount of temporal information.

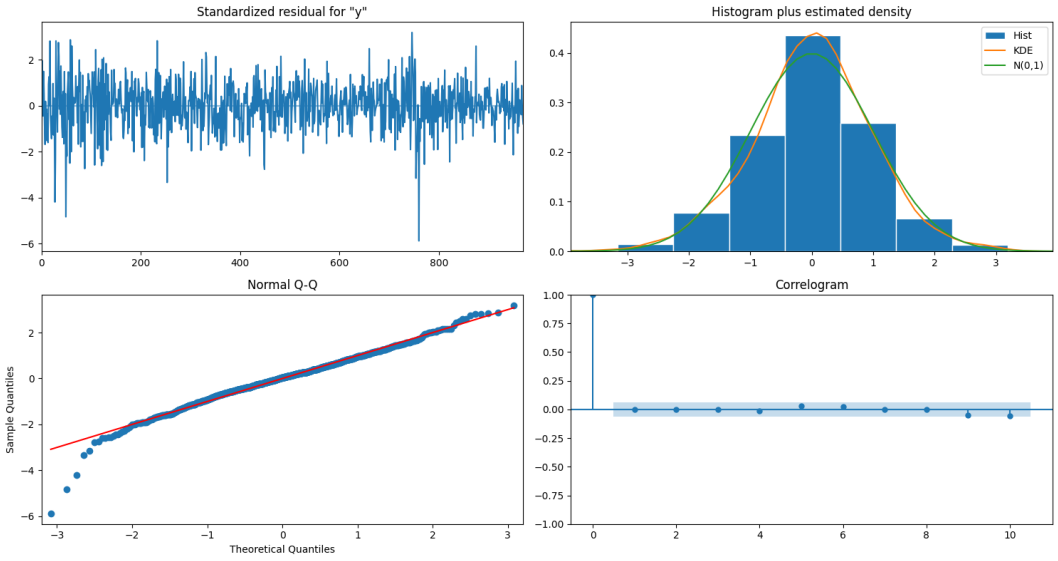
A relevant comment can be made from the ACF. Indeed, we notice that the 7^{th} and 14^{th} lags are highly significant, suggesting an underlying weekly seasonality. This makes sense from a practical point of view, as we observed in section 4.2, since prices have patterns depending on the day of the week.

To study the seasonal order in a SARIMA model, we first need to assume the period of seasonality, which from previous remarks we decide to set equal to 7 days. Next, we generate ACF and PACF plots for the time series after differencing with a lag of 7 values, setting D to 1. We also analyse the optimal orders for the remaining seasonal components, P and Q.



We note that the PACF strongly cuts off after the 1st lag and the ACF slowly tails off. When this situation happens, Box-Jenkins theory suggests us to use an order Q = 0 and P = 1, which is the lag value at which the cut-off happens in the PACF plot.

In the following subsections, we will show the results of the implementation of ARIMAX and SARIMAX models. Here, though, we briefly comment on the soundness of the model order choice, showing some model diagnostic plots after having fitted a SARIMAX model on our training set.



From the standardized residuals plot, we see that they behave quite similarly to white noise, with no obvious patterns ongoing. The curve of the (estimated) probability distribution is quite similar to a Gaussian curve and most of the data point lie on the straight line in the QQ-plot. Finally, from the correlogram we see that correlation for lags greater than 0 is statistically insignificant.
We are therefore satisfied by the model orders' choice.

15.2 ARIMAX results

We proceed to implement the ARIMAX model of order (3, 1 ,2). The reasoning behind this choice is detailed in subsection 15.1. The code utilizes the function provided within the library *statsmodels*.

The choice of exogenous variables is explained in section 15. The preprocessing is performed on the sliding training sets for both the features and the target variables.

After obtaining the point predictions, we apply the conformal prediction algorithm to obtain intervals and we evaluate the model through the metrics:

ARIMAX + CP					
Delta Coverage	RMSE	MAE	sMAPE	Pinball score	Winkler score
0.5462	5.2061	3.8401	813.2755	0.4501	23.5866

15.3 SARIMAX results

Here we implement the SARIMAX model of order (3, 1 ,2) and seasonal order (1,1,0,7). The reasoning behind this choice is detailed in subsection 15.1. The code utilizes the function provided within the library *statsmodels*.

The choice of exogenous variables is explained in section 15. The preprocessing is performed on the sliding training sets for both the features and the target variables.

After obtaining the point predictions, we apply the conformal prediction algorithm to obtain intervals and we evaluate the model through the metrics:

SARIMAX + CP					
Delta Coverage	RMSE	MAE	sMAPE	Pinball score	Winkler score
0.2531	6.1317	4.3634	911.1944	0.5379	28.5575

16 Spike Preprocessed Models

Statistical models in general are sensitive to outliers, which are a common feature of the unstable and spiky energy prices, as we observed in section 4. Therefore, to help with the prediction of the mean value of next day prices, an approach presented in the literature is to perform a pre-processing of the more spiky price values in the training set.

We follow the damping scheme suggested in [5], where a threshold T^* is fixed as the mean plus three standard deviations of the price in the training window. Any price surpassing this threshold is considered as a strong outlier, and is therefore set to $P_t = T^* + T^* \; log_{10}\left(P_t/T^*\right)$, to soften its impact on the model calibration.

This fast pre-processing step can be added to any already available model, making it more robust to outliers and possibly improving its average performance.
We therefore implement this additional step on the ARIMAX and SARIMAX models of section 15. We show here below the results on the test set, after having performed conformal prediction.

16.1 P-ARIMAX results

P-ARIMAX + CP					
Delta Coverage	RMSE	MAE	sMAPE	Pinball score	Winkler score
0.5614	5.2055	3.8389	812.9919	0.4500	23.5837

16.2 P-SARIMAX results

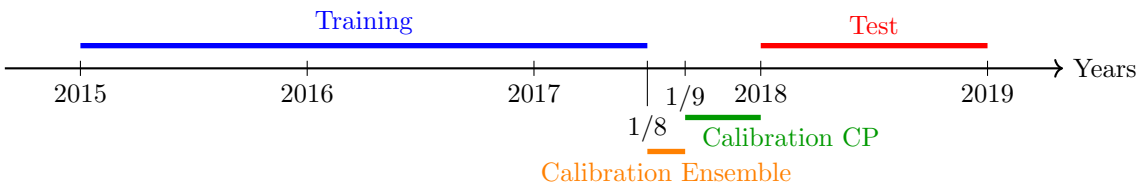
P-SARIMAX + CP					
Delta Coverage	RMSE	MAE	sMAPE	Pinball score	Winkler score
0.2643	6.1330	4.3652	911.7983	0.5381	28.5582

17 Mixed Ensemble Model

In our mixed ensemble approach, we generate a new point forecast by combining the outputs of various individual models. Mixing different models makes the final prediction more robust, as it reduces the impact of a single model's assumptions on the data.
Moreover, it can contribute in reducing over-fitting and improving the accuracy, as it can combine the different strengths of the underlying weaker learners, becoming also less sensible to outliers.
Indeed, according to [3], it is useful to combine multiple heterogeneous models in forecasting rather than exclusively using any single method.

Going into the specifics of our project, we use seven of the previously mentioned models: DNN, ARX, TARX, ARIMAX, SARIMAX, P-ARIMAX, and P-SARIMAX. To perform the ensemble, we experiment with three different forecast averaging techniques, which will be described below.

Below, we show that as calibration bag for the ensemble approach we use 28 days, as suggested by [5]. The weights are clearly re-calibrated every day, with the ensemble calibration window sliding forward daily.



Finally, after the ensemble calibration step, to obtain an interval prediction we apply conformal prediction a before.

17.1 Simple Averaging

This method simply consists in the arithmetic mean of all forecasts produced by the individual models. It is robust and widely used in business and economic forecasting.
The results we obtain are the following:

Ensemble - Simple average					
Delta Coverage	RMSE	MAE	sMAPE	Pinball score	Winkler score
0.3607	5.1038	3.7330	788.6629	0.4437	23.6136

17.2 Constrained Least Absolute Deviation

In the Constrained Least Absolute Deviation approach, the combined forecast is determined using the following regression (where M is the number of models):

$$y_t = \sum_{i=1}^{M} w_{it}\ \hat{y}_{it} + \epsilon_t$$

and we apply the absolute loss function $\sum_t |\hat{y}_{it} - y_{it}|$. This loss function is chosen over the more common mean squared error of ordinary least squares regression because it is more robust to price spikes, since it penalizes less errors on those outliers, making the ensemble weights more stable.
Moreover, we add two additional constraints:

$$w_{it} \geq 0 \qquad \text{and} \qquad \sum_{i=1}^{M} w_{it} = 1$$

The first constraint is a strong competitor to the Simple Average technique and almost always outperforms unconstrained regression, according to Aksu and Gunter (1992). The second one helps to further stabilize the weights and to gain a more natural interpretation of the coefficients w_{it}, which can be viewed as relative importance of each model in comparison to all other models.

A remark must be made here: by trying this ensemble approach on different test sets, we have observed that it possesses an unstable behaviour, over -and under- performing depending on the dates considered. This is therefore an aspect to be taken into account for the final model choice.

The results are the following:

Ensemble - LAD					
Delta Coverage	RMSE	MAE	sMAPE	Pinball score	Winkler score
0.5426	5.4590	3.9994	849.8040	0.4726	25.4332

17.3 Inverse RMSE

The main idea of this approach is to base the choice of the weights for each individual model on the inverse of the Root Mean Square Errors presented in sub-section 6.2. The weights are indeed defined

as:

$$w_{model} = \frac{\frac{1}{RMSE_{model}}}{\sum_{i=1}^{M} \frac{1}{RMSE_i}}$$

The rationale behind this approach is that models with smaller RMSE on the calibration set are more precise in those days, and will be therefore assigned larger weights in comparison to models with higher RMSE.

A remarkable property of this ensemble approach is its robustness to the test set: we have observed it trying different test dates, and it is a quality reported also in the literature (see [7]).

The results on our dataset are the following:

Ensemble - IRMSE					
Delta Coverage	RMSE	MAE	sMAPE	Pinball score	Winkler score
0.3866	5.1100	3.7401	790.7917	0.4443	23.6571

18 Final Model Choice and Results

At last the model we submit to the competition is not merely the best performing one among those discussed above, but rather an ensemble of a number of point-prediction models (see section 17), followed by the conformal prediction algorithm.

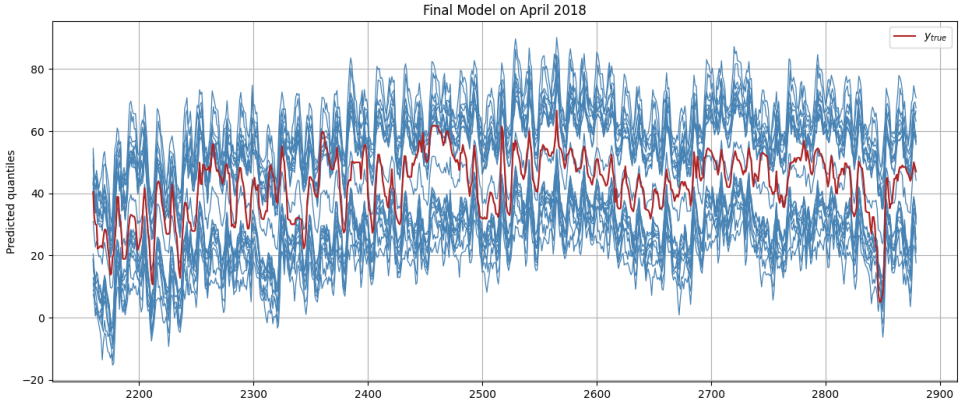
The advantages of such a strategy are many, particularly its robustness and reduced sensitivity to the composition of the test set. Additionally, an ensemble approach mitigates the impact of assumptions made by individual models, leading to more accurate predictions.

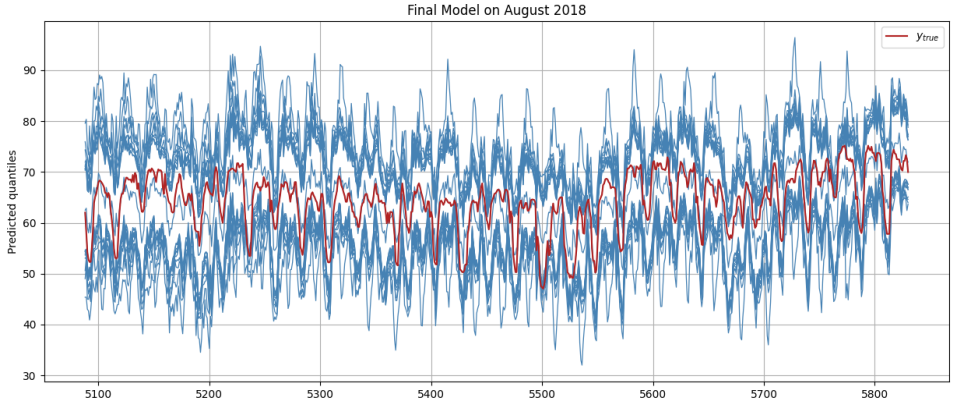
For the composition methodology, we use the Inverse Root Mean Squared Error (IRMSE) approach, motivated by its sensitivity to prediction accuracy, its robustness compared to the constrained LAD approach, and its effectiveness on the current test set.

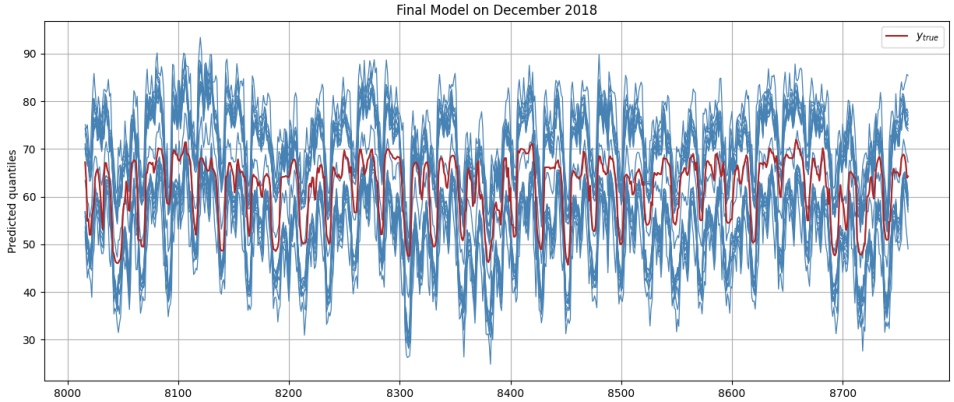
Here follows the final results of our chosen strategy for the test set of the whole 2018.

Final Model					
Delta Coverage	RMSE	MAE	sMAPE	Pinball score	Winkler score
0.3866	5.1100	3.7401	790.7917	0.4443	23.6571

Finally, we show graphically the predictions of the final model. For a clearer visual representation, to make the plots less dense, we decided to display just three individual months, in different parts of the year: April, August and December.




Final Model on August 2018
y_{true}
Predicted quantiles
90
80
70
60
50
40
30
5100
5200
5300
5400
5500
5600
5700
5800


Final Model on December 2018
y_{true}
Predicted quantiles
90
80
70
60
50
40
30
8000
8100
8200
8300
8400
8500
8600
8700

References

[1] Anastasios N. Angelopoulos and Stephen Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. 2022.

[2] George Box and Gwilym Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1970.

[3] Atom Mirakyan et al. Composite forecasting approach, application for next-day electricity price forecasting. *Energy Economics 66 (2017), 228-237*, 2017.

[4] Lago et Al. Forecasting spot electricity prices: Deep learning approaches and empirical comparison of traditional algorithms. *Applied Energy 221 (2018), 386-405*, 2018.

[5] Nowotarski et al. An empirical comparison of alternative schemes for combining electricity spot price forecasts. *Energy Economics 46 (2014), 395-412*, 2014.

[6] Roger Koenker. *Quantile Regression*. Cambridge University Press, 2005.

[7] Jakub Nowotarski and Rafał Weron. Computing electricity spot price prediction intervals using quantile regression and forecast averaging. *Computational Statistics 30 (2015), 791-803*, 2015.

[8] Jakub Nowotarski and Rafał Weron. Recent advances in electricity price forecasting: A review of probabilistic forecasting. *Renewable and Sustainable Energy Reviews 81 (2018), 1548-1568*, 2018.

[9] Baviera R. and M. Azzone. Neural network middle-term probabilistic forecasting of daily power consumption. *Journal of Energy Markets, 14.1, 1-26*, 2021.

[10] Rafał Weron. Electricity price forecasting: A review of the state-of-the-art with a look into the future. *International Journal of Forecasting 30 (2014), 1030-1081*, 2014.