SDTMSA project

Author: Pavel Shumkovskii

---

1. Introduction

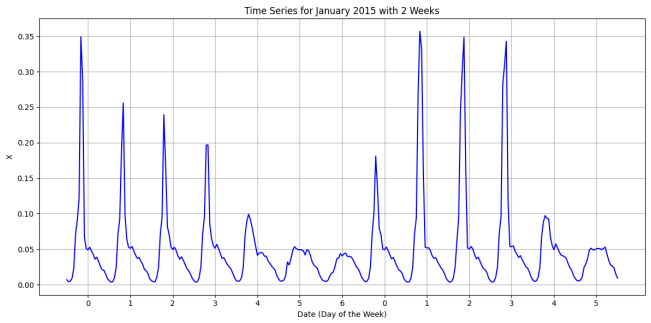
The objective of this project was to gain a deeper understanding of the dataset and explore its hidden patterns and characteristics. The data, recorded on an hourly basis, exhibits distinct seasonal variations. However, these seasonal trends are neither stable nor strictly linear, making the analysis more challenging and intriguing. Understanding such patterns is crucial, as time series data is commonly used in various real-world applications, such as forecasting electricity consumption, demand prediction, and financial market analysis.

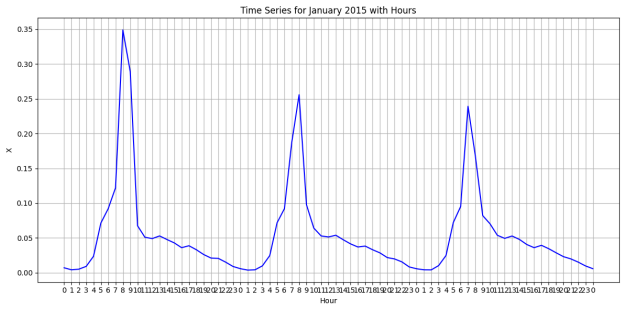
Upon an initial examination, it became evident that the dataset does not fully resemble real-world data in its natural form. Instead, it appears to be synthetic, possibly generated for research or experimental purposes.

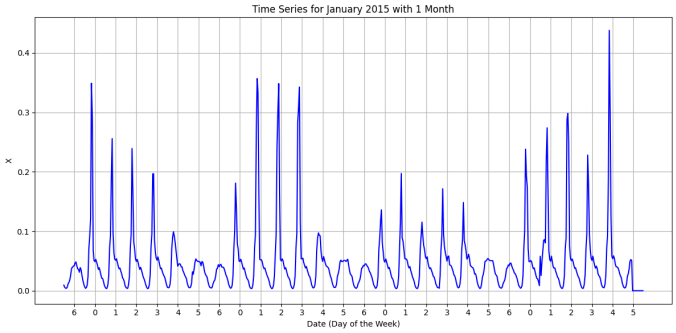
The project was implemented using Python, leveraging various libraries and statistical methods to preprocess the data, assess stationarity, and conduct in-depth analyses using tools such as wavelet transformations and the Hurst exponent. By applying different modeling approaches, including ARIMA, UCM, and deep learning models like CatBoost and LSTM, the project aimed to identify the best predictive method and derive meaningful insights from the data.

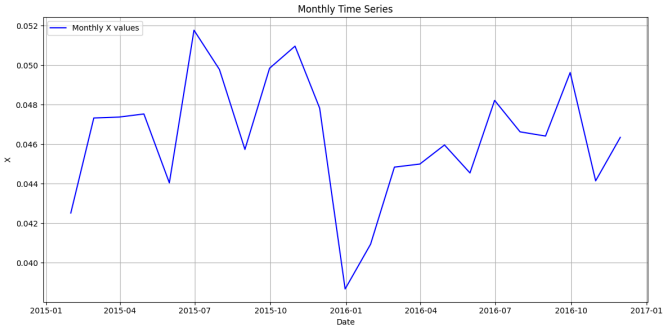
2. Preprocessing of Data

For the initial preprocessing, I have examined smaller portions of the data to investigate if there are any discernible patterns related to the time of day, week, or month.









Upon analyzing the data, it is observed that peak values typically occur around 8 in the morning, although this timing tends to drift over time. Additionally, weekly patterns show that peaks may emerge on different days of the week, such as Monday, Tuesday, and so on, indicating variability across weekdays. When examining monthly data, a potential structural break is noticeable at the beginning of 2016, which might indicate a shift in the underlying patterns or behavior of the data.

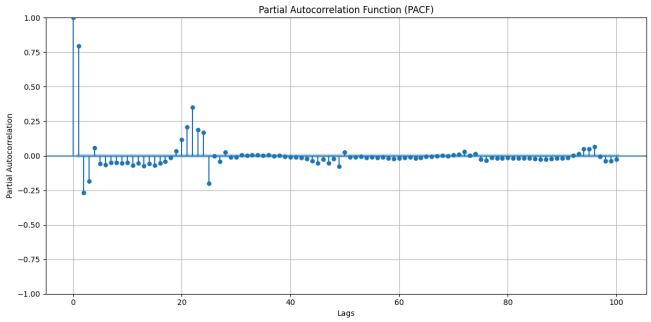
Additionally, I conducted the Augmented Dickey-Fuller (ADF) test and the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test to check for stationarity. The p-values from both tests were as follows:

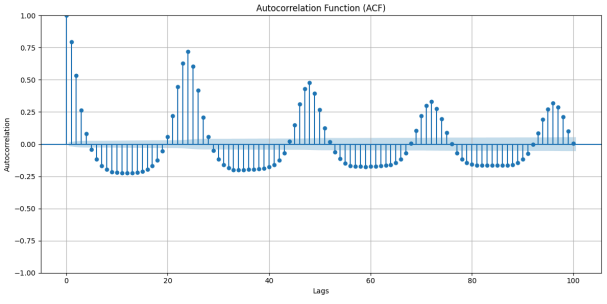
- ADF test: 0.00
- KPSS test: 0.1

Based on these results, the data appears to be stationary. However, to further confirm this, we will also assess stationarity by examining the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF).

3. Analysis of Stationarity

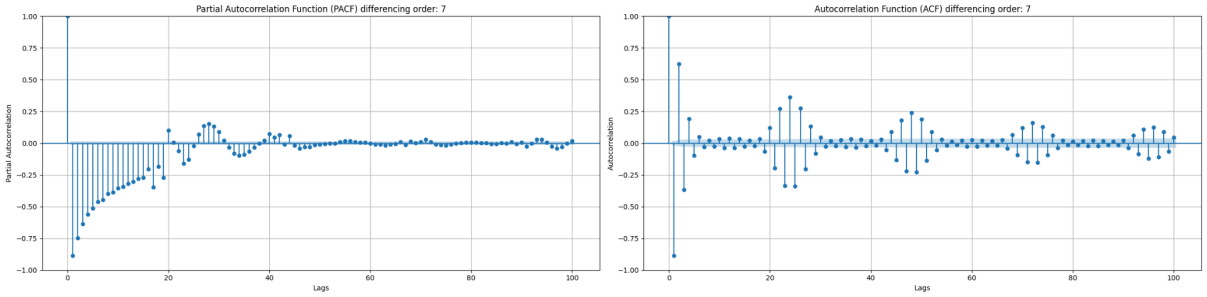
The analysis of the PACF and ACF reveals that the time series exhibits long-term memory, as nearly all lags remain significant up to the 100th lag and beyond. This suggests that past values have a prolonged impact on future observations. Additionally, the ACF shows a sinusoidal dependency across the lags, indicating periodicity or cyclical patterns in the data. This behavior further suggests that the data may not be fully stationary, and the observed long-term dependencies should be taken into account when modeling the time series.





I have experimented with different differencing techniques and log transformations on the time series, but the long-term memory component cannot be fully removed through these operations. Even after applying seven-order differencing, unusual patterns still persist in the data. Furthermore, higher-order differencing results in the loss of valuable information, reducing the time series to what resembles white noise. This indicates that while differencing

can reduce some dependencies, the inherent long-term memory and structure of the data remain prominent.



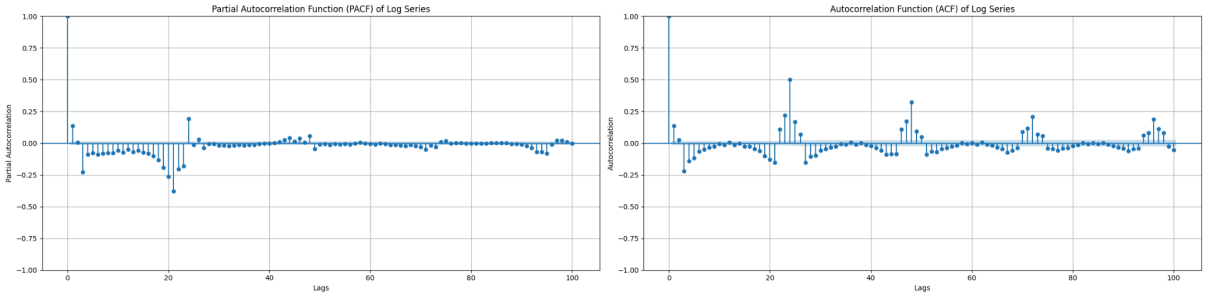
4. Deep Analysis with Advanced Tests

Motivated by the strange patterns observed in the ACF, I decided to conduct a deeper analysis and perform additional tests.

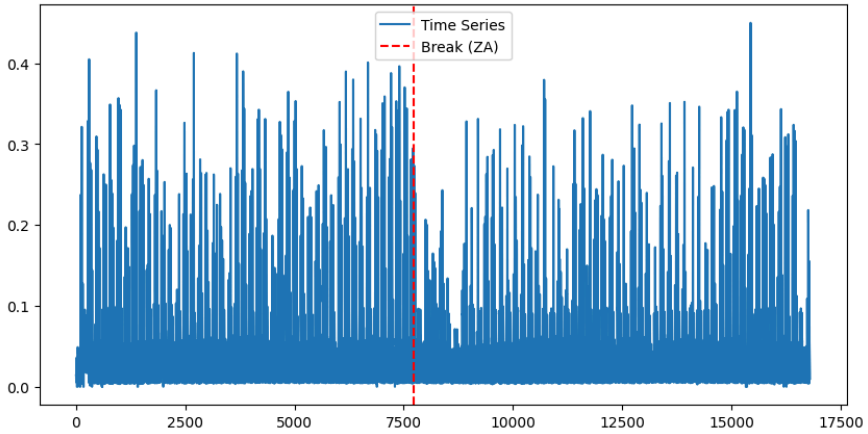
Firstly, I calculated the Hurst exponent, a statistical measure used to assess the long-term memory or dependence within a time series. The Hurst exponent ranges from 0 to 1: values closer to 0.5 indicate a random walk or no correlation, while values closer to 1 suggest persistent long-term dependence, and values closer to 0 indicate anti-persistence. The result of the Hurst exponent test was 0.9, which indicates strong long-term dependencies in the data, suggesting that future values are highly correlated with past values.

To address these long-term dependencies, I applied fractional differencing with an order of 0.4 (which is calculated 0.9 - 0.5). This method allows for capturing long-term memory while maintaining the underlying structure of the data, in contrast to traditional differencing techniques that can lead to the loss of information.

After applying fractional differencing to the time series with an order of 0.4, the Hurst exponent significantly decreased to 0.1. This suggests a reduction in the long-term dependencies within the data. However, despite this transformation, the strange patterns observed in the ACF still persisted, indicating that fractional differencing, while reducing the influence of long-term memory, did not fully resolve the underlying issues with the autocorrelation structure.



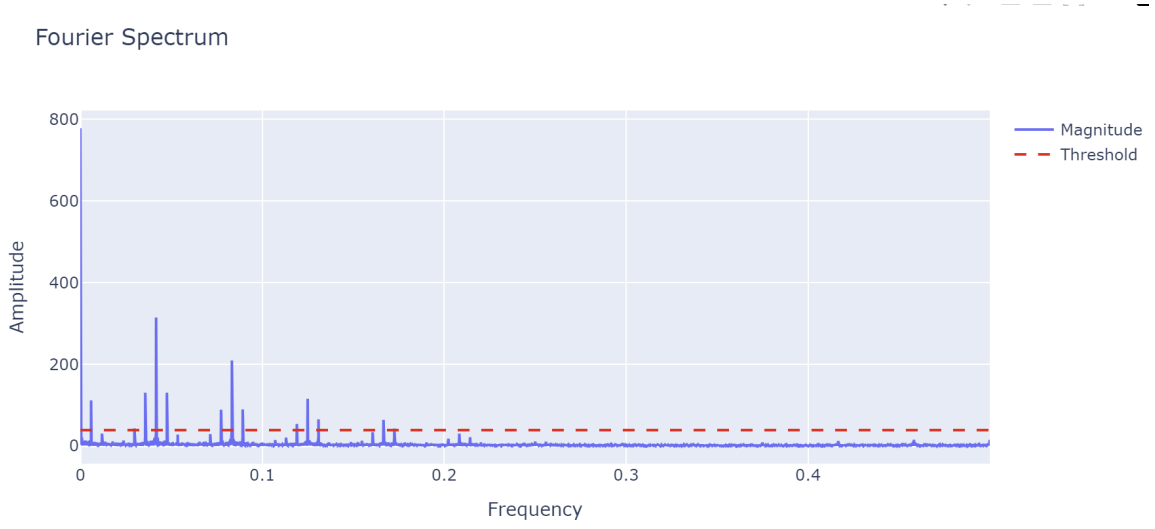
I also conducted an analysis of potential structural shifts in the data, considering that the strange patterns observed could be due to structural changes. Upon visual inspection, the Zivot-Andrews test indicated a structural break in January 2016. However, both parts of the data (before and after the break) exhibited similar long-term patterns, suggesting that the underlying dependencies remained consistent across the break.



Since my training data only includes observations after 2016, this structural break does not pose a significant concern for further analysis, as it is not relevant to the part of the data being used for model development. Therefore, there is no need to address this break in the current context.

5. Seasonality analysis

From the visual analysis, it was observed that the data exhibits different seasonal patterns. To analyze these seasonalities more rigorously, I performed a Fourier spectrum analysis to identify the values of the seasonal frequencies. The resulting frequency/amplitude graphs revealed multiple peaks across different frequency levels, suggesting the presence of various seasonal components within the data. By calculating the corresponding hours for each peak as 1/frequency, I was able to interpret the data's periodicities.



According to this analysis, I identified the following peaks at the frequencies of 169, 21, 11, and 8 hours.

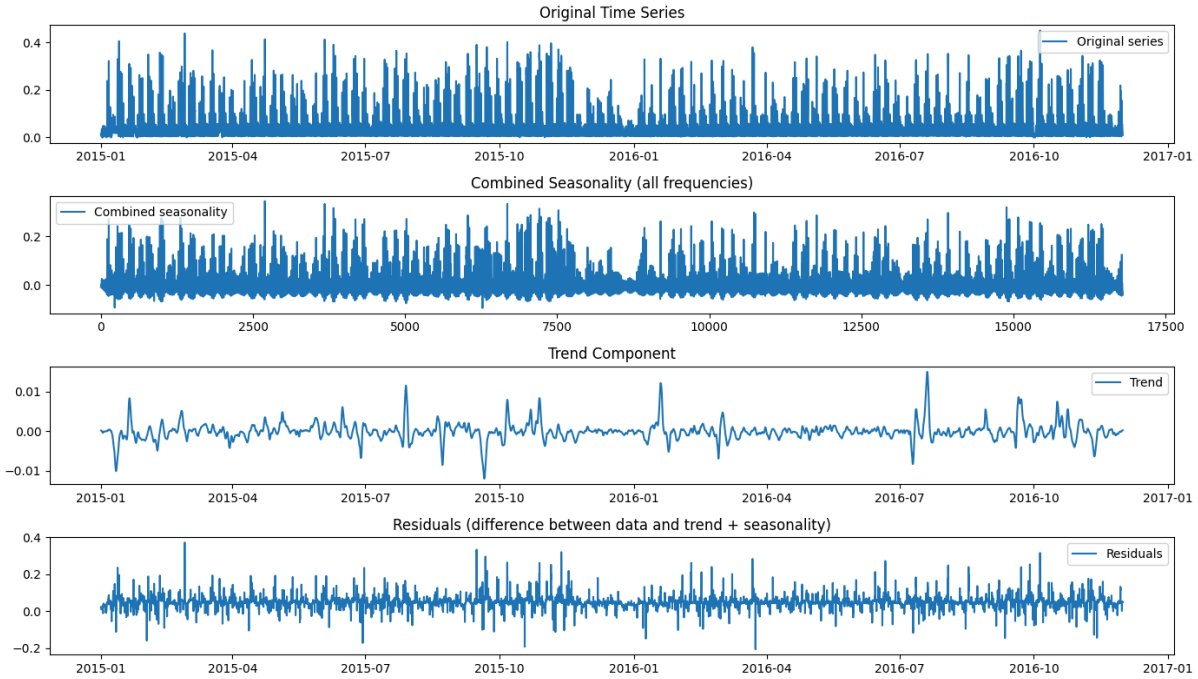
- The peak at **169 hours** corresponds to a pattern with a cycle of about **7 days**, indicating weekly seasonality.
- The peak at **21 hours** corresponds to a cycle of about **1 day**, reflecting a daily seasonal pattern.
- The peak at **11 hours** corresponds to a pattern roughly repeating every **half-day**, which might represent bi-daily fluctuations.
- The peak at **8 hours** corresponds to a cycle close to **morning and afternoon periods**, which could relate to daily work or activity cycles.

These seasonal components were considered when further analyzing the data and building models, as they reflect natural recurring patterns in the underlying system.

I decided to apply a custom approach to clear the data from multiple seasonal patterns. The approach works by decomposing the data iteratively based on different seasonal periods, and it is realized in the following way:

1. I first created a copy of the original data to work with and initialized an empty list to store the seasonal components.
2. Then, for each of the seasonal periods identified (such as the weekly, daily, bi-daily, etc.), I applied the **STL decomposition** (Seasonal and Trend decomposition using Loess). The STL decomposition is applied with the "robust" option to handle outliers effectively. This decomposes the data into seasonal, trend, and residual components.
3. After decomposing for each seasonal period, I updated the residual data by subtracting the seasonal component for that period.
4. Once all seasonal components were extracted, I summed them to create a combined seasonality pattern, and the trend was taken from the final decomposition.
5. Finally, I calculated the residuals by subtracting both the combined seasonal components and the trend from the original data.

The result of the decomposition can be seen in the graph, which shows the individual seasonal components, the trend, and the residuals.



Despite the decomposition process, the residuals appear to be distributed like white noise, with no clear patterns. However, their values are quite high, suggesting that the decomposition might not have fully removed all the complex patterns or dependencies in the data. This indicates that further refinement or different approaches might be necessary to achieve a cleaner separation of the seasonal and trend components.

6. Modeling Approaches

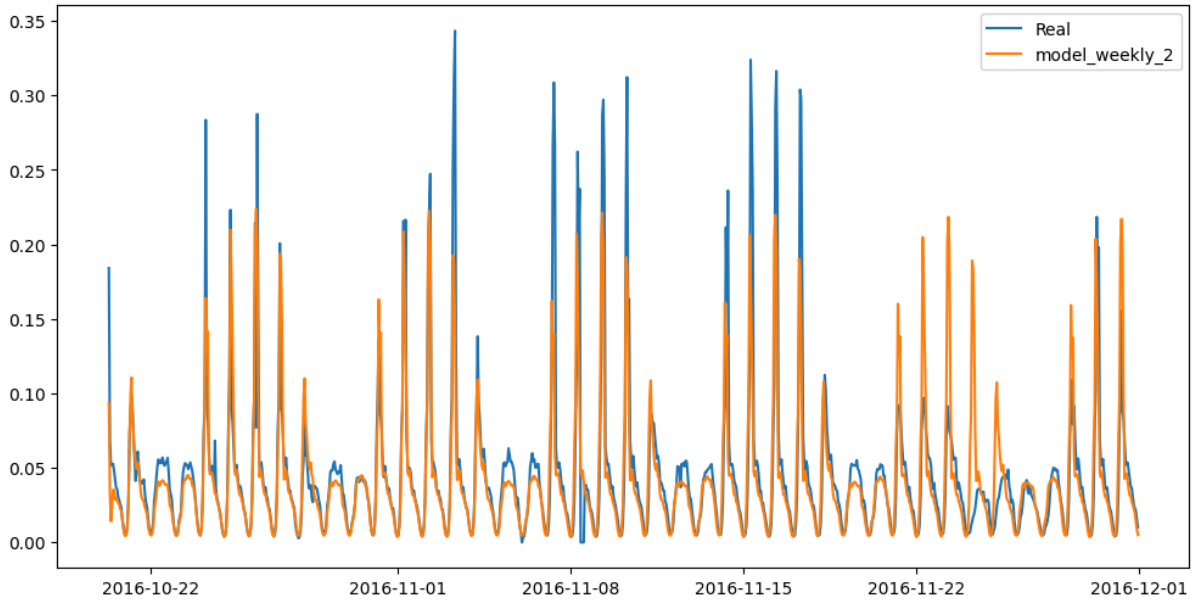
6.1 ARIMA Model

I was certain that modeling this time series with ARIMA alone wouldn't yield perfect results due to the complex seasonal patterns. So, I decided to test different seasonal orders in SARIMAX to better capture the underlying structure of the data.

Through a combination of testing and visual analysis, I found that capturing weekly seasonality (with a period of 24×724 \times 724×7 hours, corresponding to a 7-day cycle) seemed to be the most effective solution for this particular series. Therefore, I chose the following seasonal order for the SARIMAX model:

- **seasonal_order=(1, 0, 1, 24*7)**
- For the non-seasonal part, I used ARIMA parameters of **order=(2, 0, 2)**

Although the computation for the coefficients with this seasonal order was very slow, the model produced fairly good results on the test sample. This confirms that the choice of weekly seasonality was a reasonable one for this time series, and SARIMAX was able to capture the key patterns despite the complexity of the data. This is the example of the prediction on the test sample with MAE **0.014.**



6.2 Unobserved Components Model (UCM)

To effectively model the time series, I performed a series of tests to determine the optimal structure for the Unobserved Components Model (UCM). After extensive experimentation, I selected a model that includes several key components to capture the underlying patterns in the data.

The first component included in the model is the **local level**, which allows the model to track long-term trends by capturing gradual shifts in the data over time. This is essential for datasets that exhibit changes in the mean level without a strict deterministic trend.

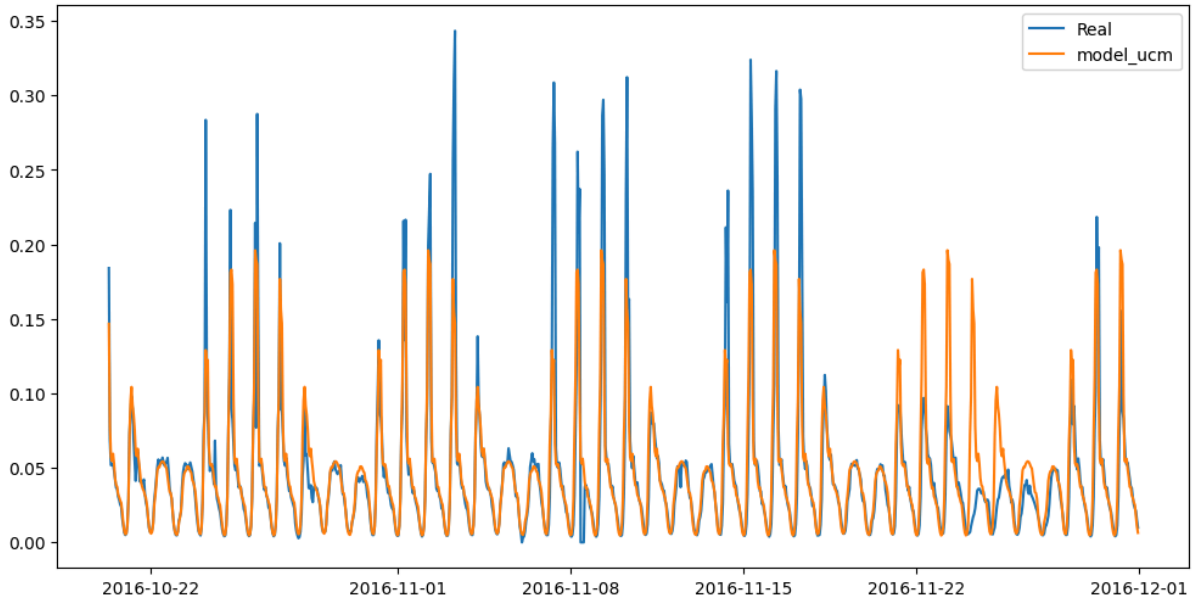
To account for recurring patterns, I incorporated a **seasonal component** with a periodicity of 168 time steps. Since the data is recorded hourly, this corresponds to a weekly cycle (168 hours = 7 days), capturing weekly seasonality. To enhance flexibility, I allowed this seasonal effect to evolve over time rather than remain fixed.

Additionally, an **irregular component** was added to account for random noise in the dataset. This ensures that short-term fluctuations, which do not follow a systematic pattern, are properly modeled rather than being misinterpreted as part of the structured components.

To improve the model's ability to capture more complex cyclical behavior, I included a **stochastic cycle component**. This allows the model to account for cycles that do not have a strictly fixed frequency but still exhibit recurring fluctuations. This is particularly useful when working with time series that may have periodicities beyond simple seasonal patterns.

Initially, I also experimented with introducing multiple seasonal components using harmonic regression techniques, but these attempts did not yield meaningful improvements. The additional harmonics failed to capture new significant seasonal effects, and the model's performance did not improve.

I came up with the quality of **0.012 MAE**, the example of the prediction on the test sample is shown on the figure:



6.3 Deep Learning Models

For deep learning models, I decided to test three different approaches:

1. **CatBoost**: I used the past 1000 lags as regressors and predicted a single value as the target.
2. **LSTM** (Long Short-Term Memory): I used 1000 past lags as input and predicted a single target value.
3. **LSTM** with 1000 lags as input and 744 target values, where 744 corresponds to the size of the test sample (representing one full day of predictions for each hour).

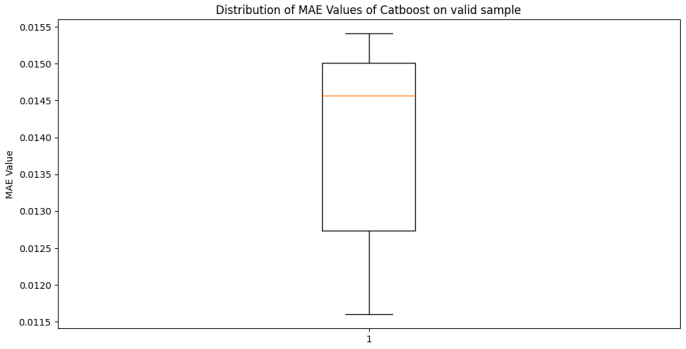
For the first two methods (CatBoost and the single-target LSTM), long-range predictions were handled in an iterative way. Specifically, I predicted the first value in the sequence, then shifted the predictor by one step, adding the newly predicted value to the input, and repeated this process until the end of the forecast horizon. This approach allows the model to generate multi-step forecasts by relying on its own predictions as inputs for subsequent steps, enabling longer-term predictions.

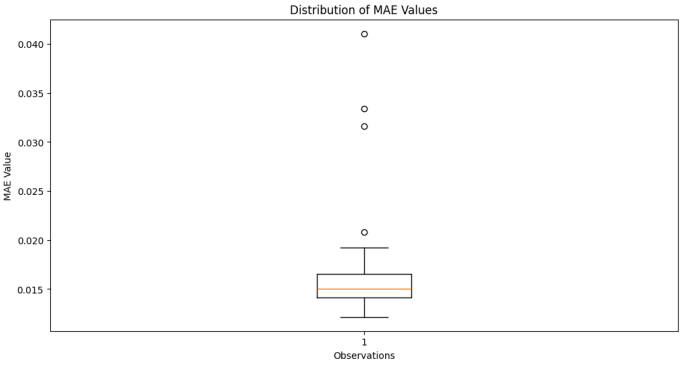
The third approach, using the LSTM with 744 target values, aimed to predict all 744 values in a single step, providing a more direct approach to forecasting the entire test sample in one go.

According to my analysis, one-shot prediction with LSTM (all points at once) was better than iterative approach, so we will compare only CatBoost results and one-shot LSTM. Hopefully,

these frameworks can use GPU acceleration (not like arima 🤣).

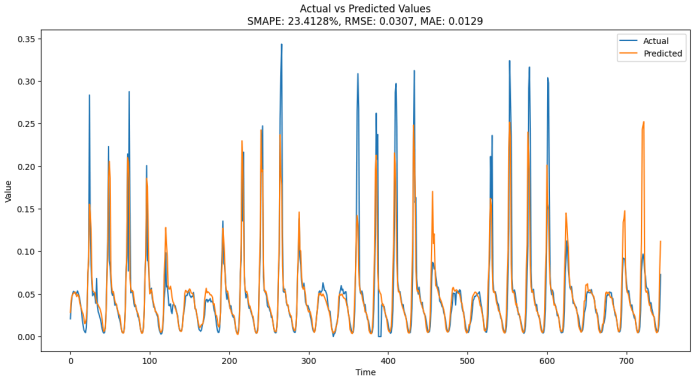
I conducted a test on the validation sample, where all 744 target points were predicted using the past 1000 lags. To evaluate the performance and compare the stability of the models, I made several partitions of the data. This allowed me to estimate how well each model performed under different conditions and assess the consistency of their predictions across various subsets of the data. By comparing the results, I was able to gain insights into the strengths and weaknesses of the two models, and determine which one provided more reliable and stable predictions. On the boxplots you can see comparison of MAE results on the validation samples (models have not seen this data before) between CatBoost and LSTM with 4 layers, 128 hidden neurons in each and 100 epochs in training.

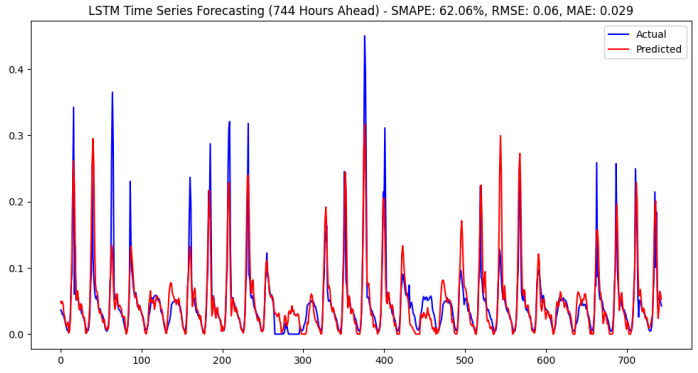




The MAE values for both approaches are close, indicating similar performance in terms of prediction accuracy. However, CatBoost seemed to be more stable overall, though it's important to note that there was a larger test sample for the LSTM, as its inference time is faster. Despite this, I decided to choose the CatBoost solution, as it not only provides good quality results but is also much easier to implement and tune compared to the LSTM. Its stability and simplicity in deployment made it the preferred choice for this task.

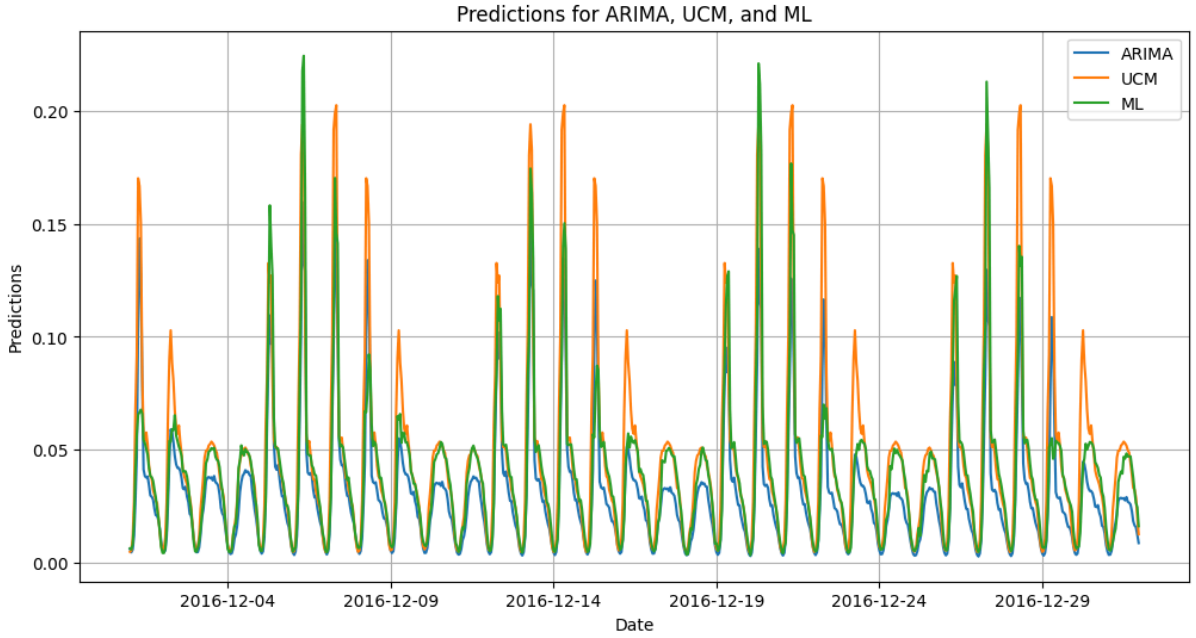
Examples of predictions (CatBoost and LSTM) are:





6.4 Combined predictions

The predictions obtained from different models demonstrate a similar underlying pattern, effectively capturing the major trends and seasonal variations present in the data. However, a key difference among them lies in the way they handle extreme values and outlier peaks. Some models tend to smooth out these peaks, while others allow for greater fluctuations, leading to variations in forecast accuracy depending on the specific nature of the test sample.



A comparative analysis of the different approaches suggests that while traditional statistical models such as ARIMA and UCM provide stable and interpretable forecasts, deep learning methods like LSTM and CatBoost show promise in capturing more complex relationships in the data. Nevertheless, further improvements could be achieved by conducting additional tests for fine-tuning model parameters and exploring alternative validation strategies. Using different validation sets could provide a better understanding of model robustness and generalization to unseen data.

Future work should focus on refining hyperparameter tuning, experimenting with ensemble approaches, and incorporating additional exogenous variables that could enhance predictive performance. By systematically evaluating these improvements, it may be possible to develop a more accurate and reliable forecasting model for practical applications.