FORECASTING SPAIN'S ELECTRICITY LOAD: A COMPARATIVE ANALYSIS OF CLASSICAL TIME SERIES, NEURAL NETWORKS, AND DEEP LEARNING MODELS

by
Evangelia Karageorgou

A Master's Thesis

Submitted to the
Department of Department of Business Organization and Administration
College of School of Business Administration
In partial fulfillment of the requirement
For the degree of
Master in Business Analytics and Data Science
at
University of Macedonia
Aug 30, 2023

Dedications

Acknowledgements

Abstract

Evangelia Karageorgou
FORECASTING SPAIN'S ELECTRICITY LOAD: A COMPARATIVE ANALYSIS OF CLASSICAL TIME SERIES, NEURAL NETWORKS, AND DEEP LEARNING MODELS
2022-2023
Dr. Konstantinos Tarabanis
Master in Business Analytics and Data Science

The objective of this research is to implement and analyze various forecasting models, including statistical, machine learning, and deep learning approaches, to forecast the electricity load in Spain from 2015 to 2020. Utilizing time-series data sourced from the Open Power System Data (OPSD) project, the study leverages the historical forecast feature of the Darts library, highlighting its retrain functionality for enhanced accuracy.

The study involved setting up and testing different models in four distinct configurations to investigate the role of past covariates and encoders on the accuracy of the forecasts. The research focused on the MSTL and AutoARIMA models for statistical analysis, while exploring the capabilities of XGBoost, LightGBM, and RandomForest models in the machine learning segment. In the realm of deep learning, NHiTS and NBEATS models were used. A detailed evaluation process was carried out, mainly using the RMSE metric to assess the performance of the various models. The results showed that deep learning models performed the best, followed by certain machine learning models, especially the LightGBM, and then the AutoARIMA model. Notably, the non-retrained versions of the models performed better than their retrained counterparts, showcasing a subtle trend in model performance. The effectiveness of including past covariates and encoders varied greatly, depending on the specific model being analyzed.

The study highlights the complex nature of electricity load forecasting and emphasizes the need for sophisticated methods in selecting and setting up models. It also suggests potential directions for future research, focusing on a deeper understanding of the retraining process, incorporating different covariates and encoders, and exploring the effects of hyperparameter tuning on model performance.

Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

List of Figures

List of Figures (Continued)

List of Figures (Continued)

List of Figures (Continued)

List of Figures (Continued)

List of Tables

List of Tables (Continued)

Table Page

List of Schemes

Chapter 1
Introduction

1.1 Background and the Importance of Forecasting in the Energy Sector

In recent decades, electricity markets have seen significant changes, mainly due to the shift from conventional to renewable energy sources. Electric power systems, intrinsically, display attributes that are time-variant, nonlinear, and dynamic. The addition of renewable energy vectors, specifically solar and wind energy, has ushered in a set of multifaceted challenges. These renewable sources, while indispensable for advancing environmental sustainability, are unpredictable. Such variability, governed by meteorological fluctuations and diurnal rhythms, superimposes an enhanced degree of unpredictability onto the electrical grid. This makes accurate electricity load forecasting even more important, as it's essential for managing the grid, making informed energy trading decisions, and planning infrastructure.

1.1.1 Background

Electricity load forecasting refers to the prediction of future electrical power demand over various time horizons. These time horizons can range from short-term (hours or days ahead) to medium-term (weeks or months ahead) to long-term (years ahead). Load forecasting plays a vital role in power system planning, operation, and control. Various methods are used for forecasting, including statistical techniques, time-series analysis, and machine learning models, each with its unique strengths and applications ([1], [2], [3]).

1.1.2 Importance in the Energy Sector

Accurate load forecasting enables utility companies to make informed decisions about power generation, transmission, and distribution. By predicting demand, utilities can optimize their generation schedules, reduce fuel consumption, and minimize operating costs. Inaccurate forecasts can lead to either underproduction, risking supply shortages, or overproduction, resulting in wasted resources.

1.1.2.1 Reliability and Stability Electricity supply must meet demand at all times to ensure the stability of the grid. Accurate forecasting helps in maintaining the balance between supply and demand, thus preventing potential blackouts or unreliability. It also aids in the effective integration of renewable energy sources, which often exhibit variable output.

1.1.2.2 Environmental Sustainability With the global push towards reducing carbon emissions, electricity load forecasting supports the integration of renewable energy into the grid. By accurately predicting demand, utilities can better harness solar, wind, and other renewable sources, reducing dependence on fossil fuels and without compromising grid stability.

1.1.2.3 Economic Benefits By predicting electricity demand, utilities can optimize generation schedules, reducing the need to run expensive peaking plants or purchase energy on the spot market. This efficiency lowers the overall cost of electricity for both providers and consumers.

1.2 Problem Statement

With the integration of renewable energy sources into the power grid, the dynamics of electricity prices and demand have become increasingly complex. [4] The variability introduced by sources like wind and solar necessitates advanced forecasting models that can capture intricate patterns and relationships. This research aims to address the challenges associated with predicting electricity prices and demand in modern electricity markets.

1.3 Objectives of the Study

The core objectives of this research are as follows:

- To develop accurate forecasting models for electricity load using advanced time series analytical methods.
- To compare the efficacy of classical time series forecasting methods with modern machine learning and deep learning techniques.
- To test the effectiveness of these models using real-world data sourced from the Open Power Systems Data of European Union countries.
- To include data on solar and wind energy generation in the models, evaluating their impact on forecasting accuracy and understanding their role within the overall system.

1.4 Scope of the Thesis

This research concentrates on the electricity load forecasting for EU countries, particularly Spain. Drawing from the Open Power Systems Data, the study evaluates forecasting models based on historical electricity load data, along with associated solar and wind generation statistics.

1.5 Methodology

The research methodology incorporates the following steps, including:

1. Data collection and preprocessing to ensure quality and consistency.
2. Exploratory data analysis to discern inherent patterns, trends, and relationships in the data.
3. Development of forecasting models using both classical, machine learning and deep learning techniques.
4. Evaluation of model performance using standard metrics and validation techniques.

Chapter 2
Literature Review

2.1 History and Evolution of Electricity Markets

The early history of electricity markets is both fascinating and complex. It is a story of technological innovation, economic transformation, and societal change [5]. The story of electricity begins with the scientific discoveries of the 18th and 19th centuries. Pioneers like Alessandro Volta, who invented the first true battery [6], and Michael Faraday (and Joseph Henry [7]), who discovered electromagnetic induction, laid the groundwork for the practical use of electricity. Thomas Edison's invention of the incandescent light bulb in 1879 marked a turning point. His work, along with that of contemporaries like Nikola Tesla and George Westinghouse, lead to the age of electric lighting [6]. The need to supply electricity to homes and businesses led to the creation of the first electric utility companies.

2.1.1 Early Beginnings

Electricity load forecasting began as a necessity to match electricity generation with consumption. During the initial stages of electrification in the late 19th and early 20th centuries, the management of electricity was a relatively simple task. The demand was predictable, and the supply was managed manually. However, as the industrial revolution progressed and electricity became more widespread, the complexity of managing electrical loads increased significantly.

2.1.2 Vertically Integrated Utilities

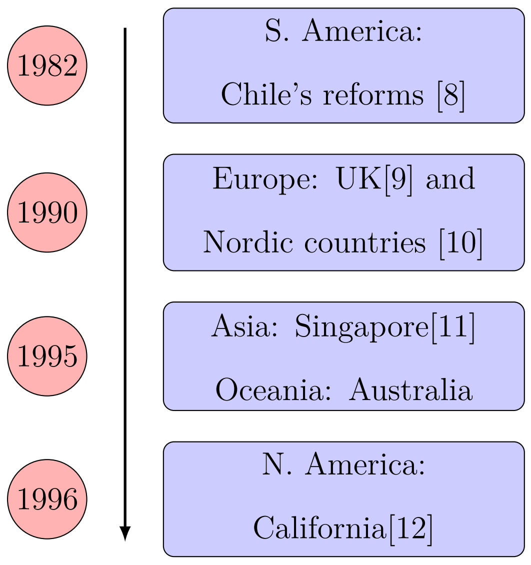
Throughout the larger part of the 20th century, many regions relied on vertically integrated utilities. In this structure, utilities were responsible for all facets of power supply, from generation to transmission and distribution. These utilities operated under rates often regulated by government entities, ensuring that electricity prices remained fair and accessible.

2.1.3 Deregulation and Liberalization

As the 20th century approached its latter stages, a wave of deregulation swept across many nations. This involved dismantling the existing vertical integration, thereby permitting distinct companies to oversee generation, transmission, and distribution individually. The overarching ambition was to instill competition within the sector, aspiring for enhanced service quality and more competitive pricing structures.

Figure 1

Liberalization across continents


1982
S. America: Chile's reforms [8]
1990
Europe: UK[9] and Nordic countries [10]
1995
Asia: Singapore[11] Oceania: Australia
1996
N. America: California[12]

2.1.4 Introduction of Wholesale Markets

The evolution of electricity markets continued with the development of wholesale markets. These markets play a crucial role in the modern electricity landscape, enabling the buying and selling of electricity in bulk quantities. These platforms facilitated bulk trading of electricity, predominantly through competitive auctions or bidding. Key features of wholesale markets are spot markets, foward markets and ancillary services.

2.1.5 Renewable Energy and 21st century

At the start of the 21st century, there was a noticeable shift toward the use of renewable energy sources, primarily solar and wind energy. This shift arose from concerns over escalating CO2 emissions and their role in global warming ([13], [14], [15], [16]). The rise of renewable energy was also supported by various government policies and incentives. These policies were aimed at reducing dependence on fossil fuels and promoting the use of clean energy sources [17]. Also, international treaties like the Paris agreement further emphasized the global shift towards renewable energy with the main goal of reduing emissions. However, because of their unpredictable nature, harnessing and integrating these energy sources presented complications in their seamless integration with power grids. Solar and wind energy, unlike fossil fuels, are dependent on environmental factors such as diurnal and weather patterns [18], [19]. As the push for cleaner energy grows, accurately predicting power needs becomes crucial, highlighting the need for further research in this domain.

2.2 Traditional Time-Series Methods and the Advent of Machine Learning

The challenge of forecasting electricity load has been approached through various methodologies. These methodologies can be broadly categorized into traditional time-series methods and modern machine learning techniques. This section delves into both, exploring their evolution, underlying mathematical concepts, and their application in electricity load forecasting.

2.2.1 Traditional Time Series Methods

2.2.1.1 Early Methods In the early days of electricity markets, load forecasting was primarily done through basic statistical methods and past trends. Techniques such as moving averages, exponential smoothing, and regression analysis were used to project future demand. These early methods, while providing a starting point, couldn't account for complex patterns, seasonal variations, and sudden changes in demand. Lack of real-time data and computational constraints further hindered the accuracy of these forecasts

2.2.1.2 Time Series Analysis With the advancement of computational power and data availability, it was possible to move on to more sophisticated forecasting techniques. Time series methodologies such as ARIMA (Autoregressive Integrated Moving Average) [20], which identifies autocorrelations in sequential data, and Exponential smoothing ([21], [20]), which focuses on trend and seasonality description, gained traction. Both models proved effective in complex intricate load patterns and addressing seasonal shifts in electricity usage [22] and are still being used today .

2.2.1.3 The Evolution of Electricity Load Forecasting With the advent of big data and machine learning, the field of electricity load forecasting has experienced a revolution in recent decades. The traditional statistical methods have been complemented, replaced by more advanced techniques, or even ensembled by machine learning models.

2.2.2 Neural Networks and Machine Learning

The emergence of neural networks and machine learning techniques marked a significant shift in load forecasting. Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), and other machine learning models provided powerful tools to model non-linear relationships and complex interactions ([23],[24],[25],[26],[27],[28],[29]). Hybrid models emerged to capitalize on the complementary strengths of different forecasting paradigms ([30], [31]).

2.2.3 Hybrid Models

Recognizing the merits and constraints of different forecasting paradigms, hybrid models emerged as a sought-after alternative. The duality of traditional statistical methods and machine learning techniques, which its have its merits and constaints, has paved the way for innovative hybrid models in electricity load forecasting as a sought-after alternative. These models, built on the premise of combining the strengths of diverse forecasting techniques, have risen in prominence. By weaving together the deterministic patterns recognized by statistical methods with the intricate non-linear relationships captured by machine learning, hybrid models offer a robust and versatile approach. For example, by mixing an ARIMA method, which is good at identying regular patterns and seasonal trends with a neural network, which is good at capturing non-linear relationships, a hybrid model can be created that is more accurate than either of its components ([32], [33]).

Chapter 3
Theoretical Background

Before proceeding, Appendix section .4 contains a lot of useful descriptive statistics and statistical tests that are used throughout this chapter.

3.1 Introduction to Time Series Data

Time series data is a sequence of observations collected or recorded at specific time intervals.

3.1.1 Components of Time Series Data

Time series data can be decomposed into four primary components:

1. **Trend Component**: The underlying pattern in the series.

$$T_t = f(t)$$

2. **Seasonal Component**: The regular changes at fixed intervals.

$$S_t = g(t \mod p)$$

3. **Cyclic Component**: Fluctuations due to economic or other broad cycles.
4. **Random Component**: Irregular or stochastic fluctuations.

$$\varepsilon_t \sim \mathrm{WN}(0, \sigma^2)$$

3.2 Time Series Decomposition

Time series decomposition involves separating a time series into its main elements: trend, seasonality, and residuals.

3.2.1 Additive Model

- In an additive model, the time series is expressed as:

$$Y_t = T_t + S_t + R_t$$

- In a multiplicative model, the components are multiplied together:

$$Y_t = T_t \cdot S_t \cdot R_t$$

3.2.2 STL Decomposition

STL (Seasonal-Trend decomposition using LOESS) is essential in time series analysis, as it breaks down a series into trend, seasonal, and residual components.

3.2.2.1 STL using LOESS (Locally Estimated Scatterplot Smoothing) LOESS is a method that uses multiple regression models to create a smooth curve for the time series. The LOESS estimation at point x_i is:

$$\hat{y}_i = \sum_{j=1}^{n} w_{ij} y_j$$

where:

- $\hat{y}_i$: Estimated value at x_i.
- y_j: Observed value at x_j.

- w_{ij}: Weight for observation j when estimating i, based on the distance between x_i and x_j.

The weights are typically determined using a tricubic function, and a local polynomial is fit to the weighted observations.

3.2.2.2 STL using Moving Averages Moving averages calculate the average of data points within a specific window for the time series X_t. The simple moving average with window size k is:

$$\text{SMA}_t = \frac{1}{k} \sum_{i=t-k+1}^{t} X_i$$

Variations such as the exponential moving average (EMA) give more weight to recent observations:

$$\text{EMA}_t = (1-\alpha) \cdot \text{EMA}_{t-1} + \alpha \cdot X_t$$

where α is the smoothing factor, determining the weight for the most recent observation.

These methods illuminate the patterns in a time series, with STL decomposition being a vital analytical tool for insightful analysis and accurate future predictions.

3.3 Statistical Forecasting Methods

3.3.1 Autoregressive Integrated Moving Average Models

3.3.1.1 Introduction to ARIMA Autoregressive Integrated Moving Average (ARIMA) models [[34],[20]] are denoted by ARIMA$(p,d,q) \times (P,D,Q)_S$, where p,d,q are the orders of the non-seasonal components, and P,D,Q are the orders of the seasonal components with a seasonal period S.

3.3.1.2 Components of ARIMA

3.3.1.2.1 Autoregressive (AR) Model An AR(p) model:

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots + \phi_p Y_{t-p} + \varepsilon_t$$

3.3.1.2.2 Moving Average (MA) Model An MA(q) model:

$$Y_t = c + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ldots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

3.3.1.2.3 Integrated (I) Part The differencing operator:

$$\nabla^d X_t = (1-L)^d X_t$$

3.3.1.3 Mathematical Representation of ARIMA The ARIMA model is defined as:

$$(1-\sum_{i=1}^{p}\phi_i L^i)(1-L)^d X_t = (1+\sum_{i=1}^{q}\theta_i L^i)\varepsilon_t$$

3.3.1.4 Seasonal ARIMA (SARIMA) Models The SARIMA model includes seasonal components:

$$\begin{aligned}(1-\sum_{i=1}^{p}\phi_i L^i)(1-L)^d(1-\sum_{i=1}^{P}\Phi_i L^{iS})(1-L^S)^D X_t = \\ (1+\sum_{i=1}^{q}\theta_i L^i)(1+\sum_{i=1}^{Q}\Theta_i L^{iS})\varepsilon_t\end{aligned} \tag{1}$$

3.3.1.5 Parameter Estimation and Model Selection

3.3.1.5.1 Parameter Estimation Parameters in ARIMA models are typically estimated through various methods:

- **Maximum Likelihood Estimation (MLE):** MLE is widely used for estimating the ARIMA model parameters by maximizing the likelihood function, considering the Gaussian distribution of the residuals.

- **Yule-Walker Equations:** Especially for AR models, the Yule-Walker equations provide a method to estimate parameters based on the sample autocorrelations.

- **Least Squares Estimation:** In some cases, the method of least squares is used to minimize the sum of squared residuals, providing estimations for the AR and MA parameters.

3.3.1.5.2 Model Selection Choosing the best-fitting model is an essential step:

- **Akaike Information Criterion (AIC) [35]:** AIC is a commonly used criterion that balances the goodness of fit and model complexity, favoring models that achieve a good fit with fewer parameters. The AIC for a model is defined as:

  $$\text{AIC} = 2k - 2\ln(\widehat{L})$$

  where:

  - $\widehat{L}$ is the maximized value of the likelihood function of the model.
  - k is the number of parameters estimated by the model.

- **Bayesian Information Criterion (BIC) [1978 paper by Gideon E. Schwarz]:** BIC is similar to AIC but puts a higher penalty on models with more parameters, often leading to more parsimonious models. The Bayesian Information Criterion (BIC) for a model is given by:

  $$\text{BIC} = k\ln(n) - 2\ln(\widehat{L})$$

  where:

- $\widehat{L}$ is the maximized value of the likelihood function for the model M, i.e. $\hat{L} = p(x \mid \hat{\theta}, M)$ with parameter values $\widehat{\theta}$ that maximize the likelihood.
- x represents the observed data.
- n is the number of observations or sample size.
- k is the number of parameters estimated by the model.

- **Grid Search:** This technique, a brute force approach, involves systematically exploring a range of values for p, d, q, P, D, Q, fitting an ARIMA model for each combination, and selecting the one with the best performance according to a chosen criterion like AIC or BIC.

- **Cross-Validation:** Time series cross-validation can be used to assess the predictive performance of different models on a validation set, providing a more direct measure of forecasting accuracy.

- **Bayesian Optimization**: This method uses a probabilistic model to predict the objective function's value and applies an acquisition function to decide where to sample next. It aims to balance the exploration of the parameter space with the exploitation of current knowledge.

3.3.2 Exponential Smoothing Models

3.3.2.1 Introduction to Exponential Smoothing Exponential smoothing models [[36], [37], [20],[38], [39], [40], [41]] are a popular family of time series forecasting methods that use weighted averages of past observations, where the weights decrease exponentially as the observations get older. These models are especially useful for short-term forecasts.

3.3.2.2 Simple Exponential Smoothing (SES) Simple Exponential Smoothing is suitable for univariate time series without trend and seasonality. The forecast equation is given by:

$$\hat{Y}_{t+1} = \alpha Y_t + (1-\alpha)\hat{Y}_t$$

where α is the smoothing parameter, Y_t is the observation at time t, and $\hat{Y}_t$ is the forecast for time t.

3.3.2.3 Holt's Linear Exponential Smoothing Holt's method extends SES to include a trend component. It introduces two smoothing equations, one for the level and another for the trend:

$$\ell_t = \alpha Y_t + (1-\alpha)(\ell_{t-1} + T_{t-1})$$
$$T_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)T_{t-1}$$

where α and β are smoothing parameters, ℓ_t is the level, and T_t is the trend.

3.3.2.4 Holt-Winters Exponential Smoothing The Holt-Winters method further extends Holt's method to handle seasonality. It involves three smoothing equations:

$$\ell_t = \alpha(Y_t - S_{t-m}) + (1-\alpha)(\ell_{t-1} + T_{t-1})$$
$$T_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)T_{t-1}$$
$$S_t = \gamma(Y_t - \ell_t) + (1-\gamma)S_{t-m}$$

where γ is the smoothing parameter for the seasonal component, and S_t is the seasonal factor.

3.3.3 Complex Exponential Smoothing

Complex Exponential Smoothing [42] extends the traditional exponential smoothing methods by incorporating multiple seasonal components, nonlinear trends, or other complex structures. These models can be tailored to capture intricate patterns in the data. T

3.3.3.1 Mathematical Representation The complex exponential smoothing model might include multiple equations, similar to Holt-Winters, but with additional terms to capture the complex structures:

$$\begin{aligned}
\ell_t &= \alpha f(Y_t, \ell_{t-1}, T_{t-1}, S_{t-m}, \ldots) + (1-\alpha) g(\ell_{t-1}, T_{t-1}, S_{t-m}, \ldots) \\
T_t &= \beta h(Y_t, \ell_{t-1}, T_{t-1}, S_{t-m}, \ldots) + (1-\beta) i(\ell_{t-1}, T_{t-1}, S_{t-m}, \ldots) \\
&\vdots
\end{aligned}$$

where $f, g, h, i, \ldots$ are functions that define the relationships between the components.

3.3.4 Theta Method

3.3.4.1 Introduction to the Theta Method The Theta method ([43], [44]) is a univariate forecasting approach that combines forecasts from different "θ lines." It was introduced as a simple yet effective method for forecasting seasonal and non-seasonal time series.

3.3.4.2 Theta Lines A θ line is a linear function fitted to a transformed time series. The transformation can be a difference, a moving average, or other techniques. The most common Theta method uses two Theta lines, one for the original series and one for a differenced series.

3.3.4.3 Mathematical Representation Given a time series X_t, the two Theta lines might be represented as:

$$\begin{aligned}
\Theta_0 &: Y_t = \alpha_0 + \beta_0 t + \varepsilon_{0t} \\
\Theta_1 &: \nabla Y_t = \alpha_1 + \beta_1 t + \varepsilon_{1t}
\end{aligned}$$

where ∇Y_t is the differenced series.

3.3.4.4 Combining Forecasts

The final forecast is a weighted average of the forecasts from the Theta lines:

$$\hat{Y}_{t+1} = w_0 \hat{Y}_{0,t+1} + w_1 \hat{Y}_{1,t+1}$$

where w_0 and w_1 are the weights, and $\hat{Y}_{0,t+1}$ and $\hat{Y}_{1,t+1}$ are the forecasts from the Theta lines.

3.4 Machine Learning in Time Series

3.4.1 Linear Regression Models in Time Series Forecasting

3.4.1.1 Linear Regression

Linear regression is a straightforward approach to modeling the relationship between the time series and its lagged values. The equation of a linear regression model can be represented as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \epsilon$$

where:

- y: Dependent variable (output/response).
- β_0: Y-intercept (bias term).
- $\beta_1, \beta_2, \ldots, \beta_p$: Coefficients for the predictors.
- $x_1, x_2, \ldots, x_p$: Predictor variables (features).
- ϵ: Error term.

3.4.1.2 Polynomial and Nonlinear Regression

Polynomial regression is an extension of the linear regression model that allows for a nonlinear relationship between the predictors and the dependent variable. Instead of fitting a straight line, polynomial regression models the data using a polynomial function. The generic equation for a polynomial regression can be represented as:

$$Y_t = \beta_0 + \beta_1 Y_{t-1} + \beta_2 Y_{t-2}^2 + \ldots + \beta_n Y_{t-n}^n + \varepsilon_t \tag{2}$$

where:

- Y_t: Dependent variable (output/response) at time t.
- $\beta_0, \beta_1, \ldots, \beta_n$: Coefficients for the predictors.
- $Y_{t-1}, Y_{t-2}, \ldots, Y_{t-n}$: Lagged values of the time series up to n lags.
- ε_t: Error term at time t.
- n: Degree of the polynomial.

While polynomial regression can model curvilinear patterns in the data, it's crucial to choose the right polynomial degree to prevent overfitting.

Nonlinear regression, on the other hand, is a more general form that allows for a wide range of relationships between the predictors and the response variable. This flexibility comes from incorporating nonlinear mathematical functions such as exponential, logarithmic, or trigonometric functions into the regression equation. Nonlinear regression can capture intricate patterns in the data, making it especially useful when the underlying data-generating process is inherently nonlinear.

The form of a nonlinear regression model can vary widely based on the specific nonlinear function used. An example using an exponential function might be:

$$Y_t = \beta_0 e^{\beta_1 Y_{t-1}} + \varepsilon_t \tag{3}$$

The primary challenge with nonlinear regression is the complexity of estimating the model parameters. Often, iterative numerical methods, like gradient descent or Newton's method, are required to find the best-fitting parameters.

3.4.1.3 Ordinary Least Squares (OLS) and Mean Squared Error (MSE)

Ordinary Least Squares (OLS) is a method used to fit a linear regression model by minimizing the sum of the squared differences between the observed and predicted values. In other words, OLS aims to find the coefficients that minimize the Mean Squared Error (MSE).

$$J(\boldsymbol{\beta}) = \frac{1}{2n} \sum_{i=1}^{n} \left(y^{(i)} - \left(\beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \ldots + \beta_p x_p^{(i)} \right) \right)^2 \tag{4}$$

where:

- n is the number of observations.
- $y^{(i)}$ is the actual output for the i-th observation.
- $\beta_0, \beta_1, \ldots, \beta_p$ are the coefficients to be estimated.

The goal in linear regression is to find the parameters $\boldsymbol{\beta}$ that minimize the MSE, as this would imply the model's predictions are close to the true data points, on average. An important characteristic of the MSE is that it penalizes larger errors more heavily than smaller ones due to the squaring of each difference, making it sensitive to outliers.

The OLS method estimates the parameters by finding the values that minimize the cost function, providing a quantifiable measure of how far away the predictions are from the actual data points. This makes OLS a foundational method for linear regression analysis.

3.4.1.4 Gradient Descent

Gradient Descent is an iterative optimization algorithm used to find the values of parameters that minimize a given cost function. In the context of machine learning, and particularly linear regression, the cost function is often the MSE, which we aim to minimize.

For a cost function $J(\boldsymbol{\beta})$ (for instance, the MSE), the idea behind gradient descent is to update the parameters $\boldsymbol{\beta}$ iteratively in the direction of steepest decrease of J. The gradient (or the vector of first partial derivatives) of J with respect to $\boldsymbol{\beta}$ points in the direction of the steepest ascent. Hence, to minimize J, we move in the opposite direction of the gradient.

The update formula for the parameters at each iteration is given by:

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \alpha \nabla J(\boldsymbol{\beta}) \tag{5}$$

Where:

- $\nabla J(\boldsymbol{\beta})$ is the gradient of J with respect to $\boldsymbol{\beta}$.
- α is the learning rate, which determines the step size in the direction of the gradient. A smaller α can lead to slower convergence, while a larger α can lead to overshooting and potential divergence.

The gradient descent algorithm stops when the change in the cost function between iterations is below a predefined threshold, or after a predefined number of iterations.

In the specific case of linear regression with the MSE as the cost function, the gradient is given by:

$$\nabla J(\boldsymbol{\beta}) = \frac{2}{n} X^T (X\boldsymbol{\beta} - \mathbf{y}) \tag{6}$$

Here, X is the matrix of predictors with each row being a data point, and $\mathbf{y}$ is the vector of true output values.

3.4.1.5 Regularization Techniques

Regularization techniques add penalty terms to the OLS loss function. The aim is to constrain the magnitude of coefficients, thereby preventing overfitting and providing more generalized models ([45], [46]).

3.4.1.5.1 Ridge Regression (L2 Regularization) Ridge regression, also known as Tikhonov regularization, combats overfitting by adding an L2 penalty term to the OLS objective function. This penalty discourages large coefficients but doesn't force them to be zero. The objective function for Ridge regression is:

$$J(\boldsymbol{\beta}) = \text{OLS term} + \lambda \sum_{j=1}^{p} \beta_j^2$$

Here, λ is a hyperparameter that controls the strength of the penalty. When $\lambda = 0$, Ridge regression becomes equivalent to OLS regression. As λ increases, the penalty has more influence, and the coefficients tend to shrink.

3.4.1.5.2 Lasso Regression (L1 Regularization) Lasso regression, short for Least Absolute Shrinkage and Selection Operator, is another regularization technique that adds an L1 penalty to the OLS objective function. Unlike Ridge, Lasso can force some coefficients to be exactly zero, effectively selecting a simpler model that doesn't include those coefficients. The objective function for Lasso regression is:

$$J(\boldsymbol{\beta}) = \text{OLS term} + \lambda \sum_{j=1}^{p} |\beta_j|$$

Again, λ is a hyperparameter controlling the penalty's strength. As with Ridge regression, when $\lambda = 0$, Lasso reduces to OLS. As λ increases, more coefficients become zero, leading to a sparser model. Both Ridge and Lasso regressions help in handling multicollinearity, reducing model complexity, and preventing overfitting. The choice between them depends on the specific problem and the nature of the dataset.

3.4.2 Tree-based Models

3.4.2.1 Decision Trees Decision Trees are a non-parametric supervised learning method that can be used for both classification and regression tasks. They are particularly useful in capturing complex, non-linear relationships in the data without having to make many of the assumptions that linear models (e.g. ARIMA) require.

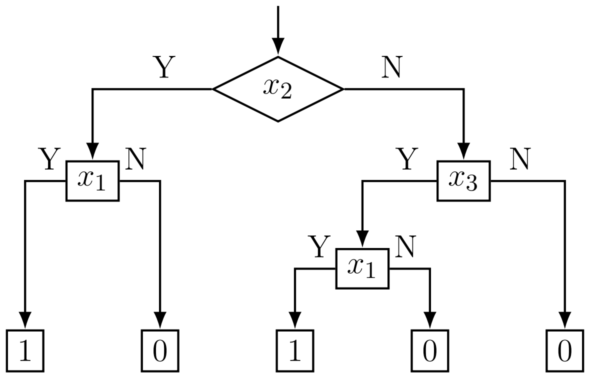
3.4.2.1.1 Structure of a Decision Tree A Decision Tree is depicted as a binary tree structure. Each internal node of this tree corresponds to a decision rule that partitions the data based on certain criteria, while each leaf node provides a prediction. These decision rules are established by choosing an attribute and a corresponding threshold value that optimally divides the data into distinct classes or ranges. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation. While decision trees are intuitive and easily visualized, they're prone to overfitting (requires mechanisms like pruning to counteract) and biased towards dominant classes (balancing the dataset is necessary prior to fitting)

In essence, Decision Tree predictions are derived from the means of particular subsets of the training data. These subsets are identified by segmenting the input data space into axis-parallel hyperrectangles. For each of these hyperrectangles, the model computes the average of all observed outputs contained within and uses this average as its prediction.

However, there's an inherent limitation: for forecasting purposes, the model invariably predicts the mean of the final training interval, rendering it ineffective. This is because Decision Trees inherently struggle with out-of-distribution data. In time series regression, every future point is, by definition, out-of-distribution, making traditional Decision Trees ill-suited for such tasks. In this section, we also present a decision tree used for evaluating a series of criteria based on variables x_1, x_2, and x_3. The decision tree helps in making decisions based on multiple conditions and provides an intuitive way to visualize the decision-making process.

Figure 2

A decision tree illustrating conditions based on features x_1, x_2, and x_3. Each node represents a condition on a feature, guiding the decision-making process.



The tree starts at the root with a decision based on the variable x_2. If the condition for x_2 is satisfied (Yes), it further evaluates x_1. Depending on the values of x_1 and x_2, the final decision is either 1 or 0. If the condition for x_2 is not satisfied (No), the tree evaluates x_3 and subsequently x_1 if necessary. Each leaf node of the tree represents a final decision outcome. However, something to notice, is that tree-based models aren't the best at extrapolation. Decision trees

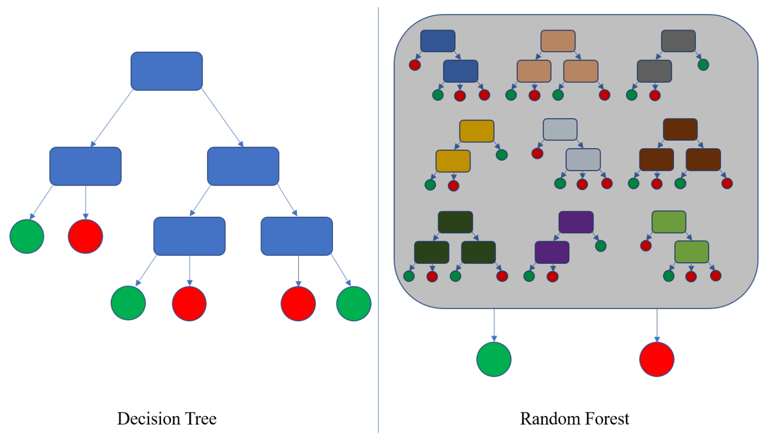
3.4.2.2 Random Forests in Time Series Forecasting Random Forests are an ensemble learning method that builds multiple Decision Trees and merges their predictions ([47], [48]). As the name indicates, a random forest entails a set of trees. The building blocks are still single trees, but instead of fitting just one tree, there are multiple trees. By leveraging the power of the ensemble, Random Forests often achieve higher accuracy and robustness compared to a single Decision Tree.

3.4.2.2.1 Building a Random Forest The process of building a Random Forest involves the following steps:

1. **Bootstrap Sampling:** Draw B bootstrap samples from the original dataset

Figure 3

A schematic illustration comparing the algorithm and complexity of decision trees and random forests. Source for image: [49]



with replacement. Each bootstrap sample forms a separate dataset used to build an individual Decision Tree.

2. **Building Decision Trees:** For each bootstrap sample, build a Decision Tree. While splitting a node, select a k random subset of features to consider for the split. This randomization helps to decorrelate the trees.

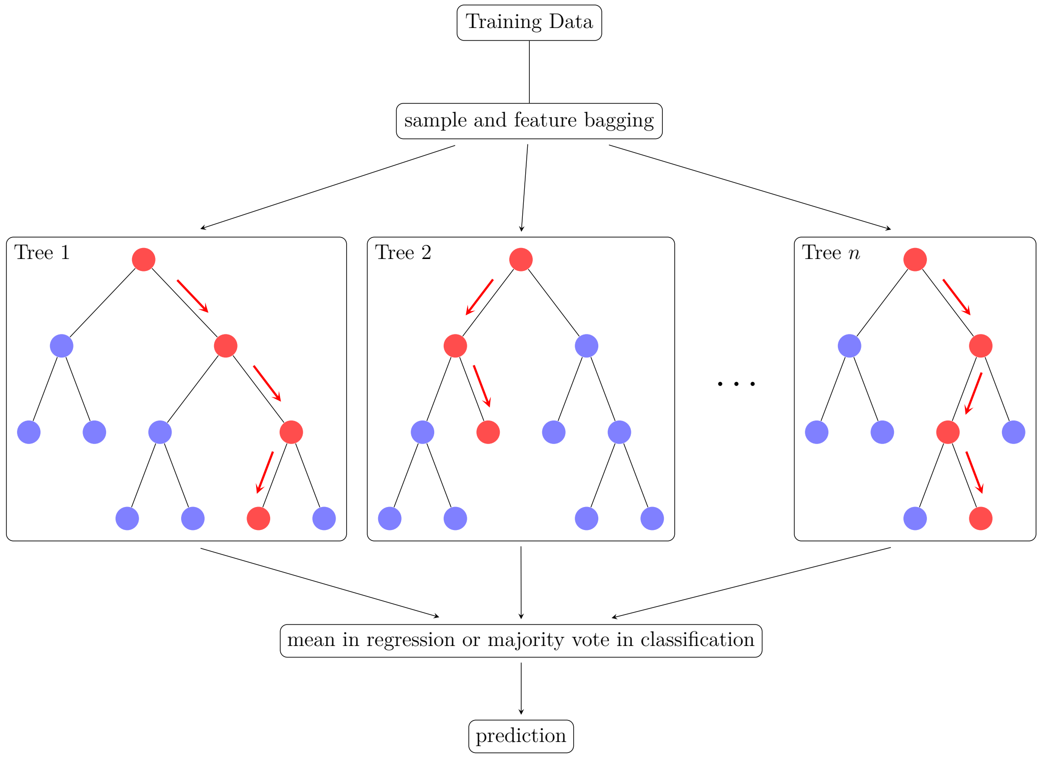
3. **Combining Predictions:** The final prediction of the Random Forest is obtained by averaging the predictions of the individual trees (for regression) or by majority voting (for classification).

Random Feature Selection k is a tuning parameter. At each split, a random subset of k features is selected out of the total p features. A typical choice is $k = \sqrt{p}$ for classification or $k = \frac{p}{3}$ for regression [47].

Prediction Aggregation For regression, the final prediction is the average of the individual trees' predictions:

Figure 4

A schematic representation of the Random Forest algorithm. It illustrates the bagging of training data and subsequent creation of multiple decision trees. The final prediction is either the mean (in regression) or majority vote (in classification) of individual tree predictions. Source for the image code: [50]


Training Data
sample and feature bagging
Tree 1
Tree 2
Tree n
mean in regression or majority vote in classification
prediction

$$\hat{y} = \frac{1}{B}\sum_{b=1}^{B} \hat{y}_b$$

For classification, the final prediction is the class with the majority vote:

$$\hat{y} = \arg\max_c \sum_{b=1}^{B} I(\hat{y}_b = c)$$

3.4.2.2.2 Advantages and Limitations of Random Forests

Advantages

- **Reduces Overfitting:** Unlike individual Decision Trees, which can easily overfit to the training data, Random Forests average the predictions of multiple trees. This ensemble approach reduces the variance and the tendency to overfit, leading to more robust predictions.

- **Handles Non-linear Relationships and Complex Interactions:** Random Forests can capture complex non-linear relationships and feature interactions, that a single Decision Tree might miss by considering random subsets of features at each split.

- **Provides Feature Importance Scores:** Random Forests provide a natural way to evaluate feature importance by calculating how much each feature contributes to overall prediction accuracy. This can be extremely useful in understanding key forecast drivers and focusing on influential variables.

- **Improved Accuracy:** Random Forests frequently achieve higher accuracy by aggregating the predictions of multiple Decision Trees. They utilize the wisdom of the crowd - or ”forest”- to compensate for individual tree errors and usually outperform a single Decision Tree.

Limitations

- **Computationally More Expensive:** A Random Forest requires more computational resources to build and predict than a single Decision Tree. Longer training and prediction times may result from the need to train multiple trees and combine their predictions.

- **Less Interpretable than a Single Decision Tree:** While a single Decision Tree provides a clear and easily interpretable decision-making process, the ensemble nature of a Random Forest makes it more difficult to interpret. The insights from individual trees might be obscured in the aggregated predictions.

- **Hyperparameter Tuning Complexity:** Random Forests introduce additional hyperparameters like the number of trees and the number of features considered at each split.

Random Forests extend the capabilities of Decision Trees by building an ensemble that leverages the strength of multiple trees. Through techniques like bootstrap sampling, random feature selection, and prediction aggregation, they provide a robust and often more accurate approach to time series forecasting.

3.4.3 XGBoost

XGBoost, an acronym for "Extreme Gradient Boosting," represents an advanced gradient boosting library designed for optimal performance across distributed systems. The fundamental principle of XGBoost revolves around the gradient boosting technique, which aims to iteratively combine weak learners (commonly shallow Decision Trees) to formulate a robust learner. The library's inception can be attributed to Tianqi Chen, as detailed in the paper "XGBoost: A Scalable Tree Boosting System" [51].

3.4.3.1 Introduction to XGBoost Famed for its efficiency, flexibility, and adaptability, XGBoost has been pivotal in clinching victories in various machine learning challenges. One of its most celebrated attributes is its unparalleled speed and performance, making it a preferred choice among data scientists [51].

3.4.3.2 Mathematical Formulation In XGBoost, the objective function amalgamates a loss function (quantifying the model's data fit) with a regularization term (penalizing model complexity) [51]:

$$\mathrm{Obj}(\Theta) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where:

- $\ell(y_i, \hat{y}_i)$ denotes the loss function, contrasting the actual value y_i with its predicted counterpart $\hat{y}_i$.
- $\Omega(f_k)$ signifies the regularization term, which introduces penalties for the complexity within individual trees f_k.
- K represents the cumulative number of trees.

Further, the regularization term is expressed as:

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda\|w\|^2$$

where:

- T corresponds to the number of tree leaves.
- w is indicative of the leaf weights vector.
- Both γ and λ serve as hyperparameters, modulating the model's complexity.

3.4.3.3 Training Algorithm The gradient boosting technique employed by XGBoost [51] is characterized by the sequential addition of trees, wherein each new tree is tasked with rectifying the predecessor's errors. The steps involved in the training process are:

1. **Initialization**: Establish an initial prediction, typically the target variable's mean.
2. **Tree Building**: Iterate through the following steps:
    (a) Compute the loss function's negative gradients, often termed as pseudo-residuals.

   (b) Incorporate a new Decision Tree tailored to the negative gradients.

   (c) Refine predictions by integrating the new tree's weighted outcomes.

3. **Regularization**: Introduce regularization to temper the model's complexity.

3.4.4 LightGBM

LightGBM, as detailed in [52] and [53], is an open-source, distributed, high-performance gradient boosting framework. Originally associated with the Distributed Machine Learning Toolkit (DMLC) community and with significant contributions from Microsoft, it is renowned for its efficiency, speed, and capability to process large datasets with ease.

3.4.4.1 Introduction to LightGBM The term "LightGBM" stands for "Light Gradient Boosting Machine." It achieves its efficiency over other gradient boosting algorithms by employing a histogram-based learning method. This not only accelerates the training process but also speeds up prediction tasks.

3.4.4.2 Mathematical Formulation The objective function in LightGBM, similar to other gradient boosting frameworks, is given by:

$$\mathrm{Obj}(\Theta) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where:

- $\ell(y_i, \hat{y}_i)$ represents the loss function, which can vary depending on the specific task, such as regression or classification.
- $\Omega(f_k)$ denotes the regularization term.
- K signifies the number of trees.

The regularization term is typically represented as:

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda\|w\|^2$$

3.4.4.3 Key Features

3.4.4.3.1 Histogram-based Learning By converting continuous feature values into discrete bins or histograms, LightGBM optimizes memory usage and accelerates the training process. Such an approach enables faster computations as the algorithm primarily deals with bin edges instead of unique values.

3.4.4.3.2 Gradient-based One-Side Sampling (GOSS) GOSS retains all instances with significant gradients and conducts random sampling on a fraction of instances with smaller gradients. This approach not only expedites the computation but also addresses the potential skewed distribution of residuals, often resulting in enhanced model accuracy.

3.4.4.3.3 Exclusive Feature Bundling (EFB) EFB aggregates mutually exclusive features—those that are rarely non-zero simultaneously—into singular features, effectively reducing the data's dimensionality without significant information loss.

3.5 Deep Learning in Time Series Forecasting

3.5.1 Introduction to Deep Learning for Time Series

Deep learning, a subset of machine learning, involves the use of artificial neural networks with multiple layers (known as deep networks) to model complex relationships in data. In the context of time series forecasting, deep learning offers several advantages:

3.5.1.0.1 Automatic Feature Learning Traditional time series models often require manual feature engineering and selection. Deep learning models, on the other hand, are capable of automatically learning relevant features from raw time series data. This ability to learn temporal dependencies, recognize underlying trends, and adapt to seasonality without explicit programming renders deep learning particularly invaluable for time series forecasting.

3.5.1.0.2 Modeling Complex Patterns Deep learning models can capture complex non-linear relationships and interactions that might be challenging for traditional models. They are well-suited for handling multivariate time series and can model intricate dependencies between different time series variables.

3.5.1.0.3 Flexibility and Scalability With various architectures such as Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Transformer models, deep learning provides a flexible and scalable approach to time series forecasting. These architectures can be tailored to diverse forecasting requirements, whether they pertain to short-term predictions or long-range projections, varying data frequencies, or differing magnitudes of data scales.

3.5.1.0.4 Mathematical Foundations The core of deep learning lies in the optimization of a loss function through the iterative adjustment of model parameters (weights and biases). Given a set of input features $\mathbf{x}$ and corresponding target values $\mathbf{y}$, a typical deep learning model aims to minimize a loss function $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$, where $\hat{\mathbf{y}}$ is the model's prediction:

$$\min_{\Theta} \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$$

The optimization is often performed using algorithms like Stochastic Gradient

Descent (SGD) or its variants, updating the parameters Θ based on the gradients of the loss function:

$$\Theta_{t+1} = \Theta_t - \alpha \nabla \mathcal{L}(\Theta_t)$$

where α is the learning rate, and $\nabla \mathcal{L}(\Theta_t)$ represents the gradient of the loss function with respect to the parameters at iteration t.

The iterative nature of these optimization algorithms enables the model to refine its predictions over time, honing its forecasting capabilities. Regularization techniques, dropout layers, and batch normalization are often incorporated to prevent overfitting and improve generalization to unseen data.

3.5.2 Introduction to RNNs

3.5.2.1 The Limitations of Traditional Neural Networks Sequential processing of information is ingrained in many tasks, from natural language understanding to time series forecasting. This limitation in traditional neural networks is particularly pronounced when processing sequential data, which is abundant in tasks such as natural language understanding and time series forecasting.

This is where recurrent neural networks come into play, to address this issue.

3.5.2.1.1 Introduction to RNNs Recurrent Neural Networks (RNNs) address the limitations of traditional neural networks by introducing a loop mechanism, allowing information to persist and thereby capture time-related relationships in sequential data. This looped or recurrent structure is what gives them their name and the ability to remember.

3.5.2.1.2 Vanilla RNNs The foundational architecture of RNNs, often termed 'Vanilla RNN', maintains a hidden state that captures information over time.

This state is crucial for the RNN's ability as it is the reason it remembers past information. The hidden state at time t, denoted h_t, is computed as:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b_h)$$

where

- h_{t-1} is the hidden state at the previous time step.
- x_t is the input at time t.
- W_h, W_x are the weight matrices.
- b_h is the bias term.
- σ is an activation function, such as the hyperbolic tangent.

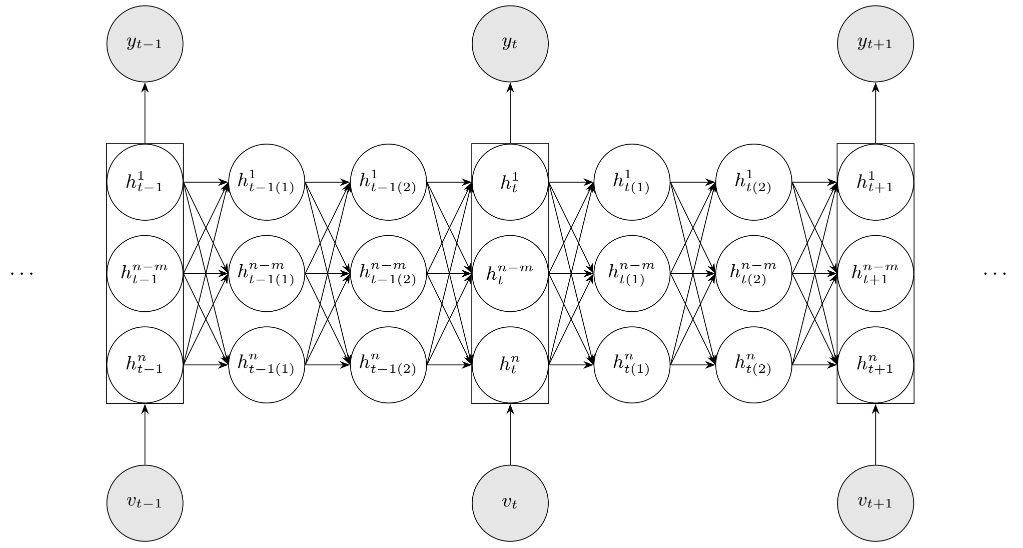
The hidden state h_t acts as a form of memory, keeping past information. As the network processes new data, this state is updated, ensuring that past context is considered in upcoming computations.

While Vanilla RNNs provide a foundation for sequence modeling, they can struggle with learning long-term dependencies due to the vanishing gradient problem [54] [55], leading to the development of more advanced RNN variants like LSTMs.

The vanishing gradient problem arises when training traditional RNNs, especially during backpropagation through time (BPTT). As the RNN processes sequential data over extended periods, it employs chained activations. If these activations involve functions like the Sigmoid, whose derivative's maximum value is less than 0.3, the gradients can diminish rapidly. When this happens, the network's weights and biases don't update significantly, making it challenging for the RNN to learn from long sequences. This concept is depicted in a simplified manner in Figure 6.

Figure 5

Unfolded architecture of an RNN across various time steps. A detailed breakdown of the RNN architecture is provided in Appendix section .4.



3.5.2.1.3 LSTM Networks "LSTMs", or Long Short-Term Memory networks, were pioneered by Hochreiter & Schmidhuber in (1997) [56]. They are an advanced RNN variant, adept at learning long-term dependencies. They were designed to circumvent the long-term dependency and vanishing gradient challenges. The LSTM introduces specialized gates that optimize its memory, retaining only pertinent information. [57] They introduce memory cells and gates to control the flow of information:

- **Forget Gate**: Controls what information to discard from the cell state.

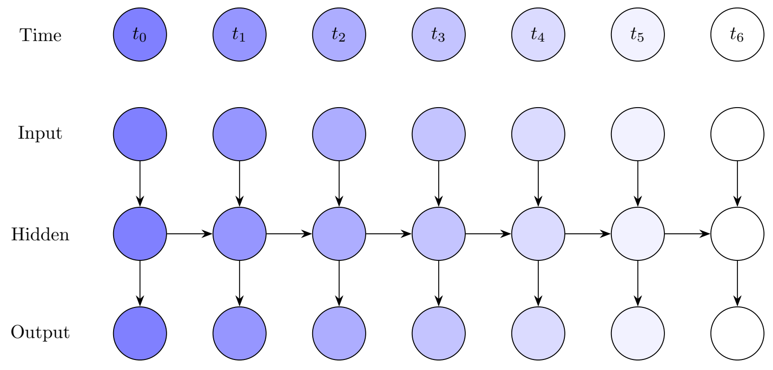
$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

- **Input Gate**: Decides what information to store in the cell state.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

Figure 6

The illustration depicts the vanishing gradient challenge in RNNs. The gradient of the nodes in the unfolded network, indicated by the shading, represents their sensitivity to initial inputs. Darker shades signify greater sensitivity. As the RNN processes subsequent inputs, the sensitivity diminishes, illustrating the network's tendency to 'forget' earlier inputs.



- **Update Gate**: Creates a candidate update for the cell state.

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$$

- **Cell State**: Combines the forget gate, input gate, and update gate to create the new cell state.

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

- **Output Gate**: Decides what part of the cell state to output.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

- **Hidden State**: Computes the final hidden state based on the output gate and cell state.

$$h_t = o_t \odot \tanh(C_t)$$

3.5.2.1.4 ***Gated Recurrent Units (GRU)*** Gated Recurrent Units (GRUs) were introduced in 2014 [58] as a variant of recurrent neural networks. While they share similarities with Long Short-Term Memory (LSTM) units, particularly in using a gating mechanism, GRUs simplify the model by combining the forget and input gates into a single "update gate" and lack an output gate. This results in fewer parameters, making them computationally more efficient. The key equations are:

- **Update Gate**:

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z)$$

- **Reset Gate**:

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$$

- **Candidate Hidden State**:

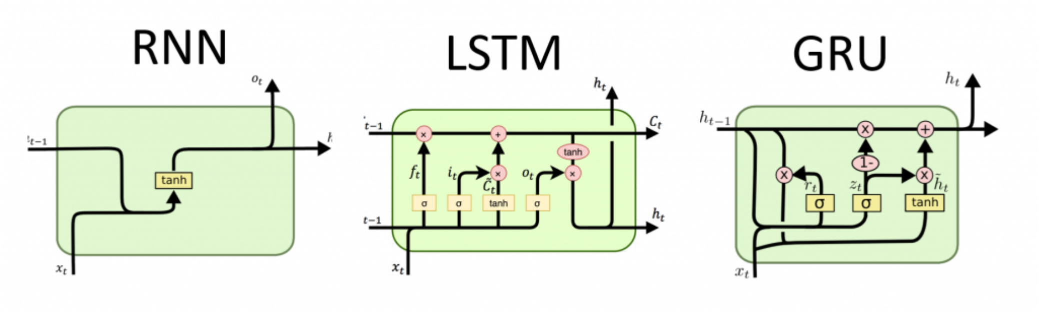
$$\tilde{h}_t = \tanh(W_h[r_t \odot h_{t-1}, x_t] + b_h)$$

- **Hidden State**:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Figure 7

Comparison of RNN, LSTM, and GRU cells. Source: [59]



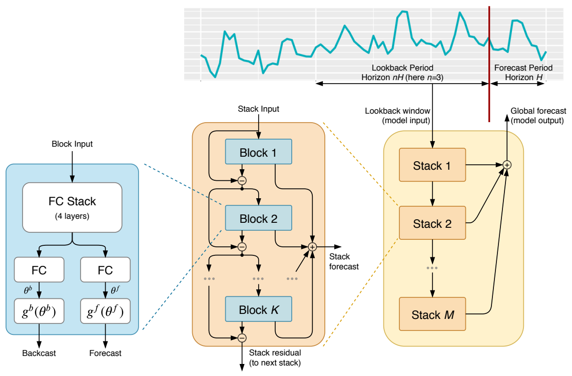
BlockRNN, as used in the Darts library, leverages RNN architectures to encode fixed-length input chunks and employs a fully connected network for producing fixed-length outputs. It supports past covariates known for a specific duration before prediction time. The library provides three primary variants of RNNs for this purpose: Vanilla RNN, LSTM, and GRU. Such a modular approach ensures flexibility and adaptability across various sequence modeling tasks [60].

3.5.3 N-BEATS Model Overview

N-BEATS [61], standing for Neural Basis Expansion Analysis for Interpretable Time Series, is a deep neural network built on a deep stack of fully-connected layers with backward and forward residual links. It uses basis expansion to improve data representation, enhancing its capability to model non-linear relationships in time series.

Figure 8

N-BEATS architecture highlighting its stacks and blocks. Each block connects feedforward networks with forecast and backcast links, focusing on residual errors unaddressed by prior blocks. The cumulative partial forecasts yield a comprehensive global forecast. Source: [61]



3.5.3.1 Basis Expansion Basis expansion amplifies data to model non-linear relationships by extending the feature set. A prevalent form, polynomial basis expansion, alters features (like squaring) to fit non-linear patterns more aptly.

3.5.3.2 N-BEATS Architecture N-BEATS, with its layered design, prioritizes simplicity, adaptability, and extensibility. It comprises multiple stacks, with each stack containing blocks that generate both forecast and backcast outputs. The model's design ensures that unaddressed residuals are forwarded for subsequent processing, guaranteeing a thorough forecast.

Figure 9

N-BEATS Basic Block, using a lookback window based on forecasting horizon H. After processing through a neural network, it estimates expansion coefficients which undergo a "neural basis expansion" to generate backcast and forecast signals. Source: [61]

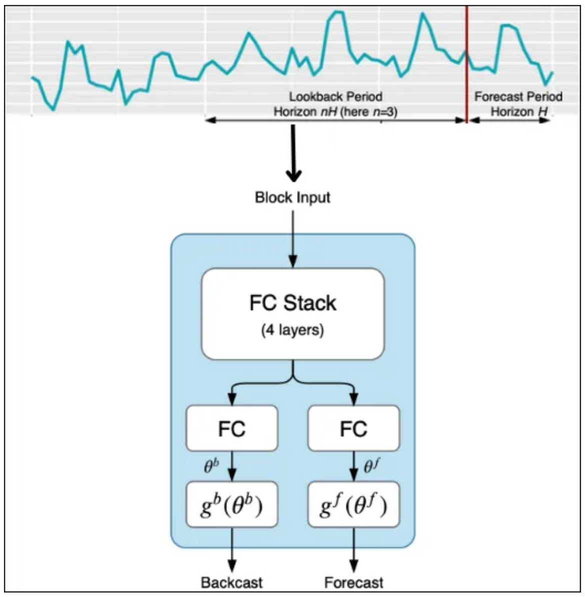
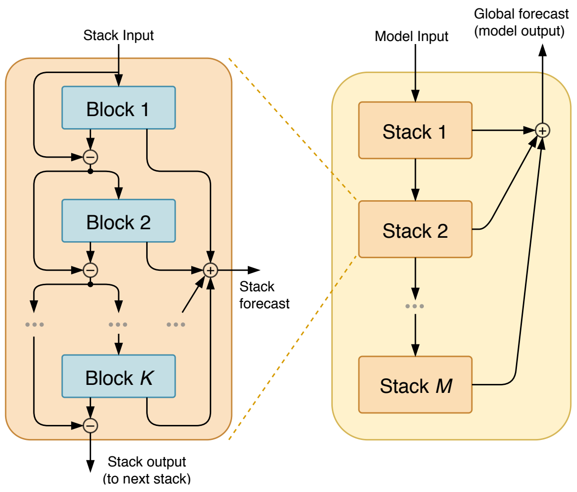


Figure 10

N-BEATS' stacking mechanism. Beginning with the true input sequence, subsequent blocks process the backcast signal from their predecessors. This doubly residual stacking culminates in an aggregated forecast. Source: [61]

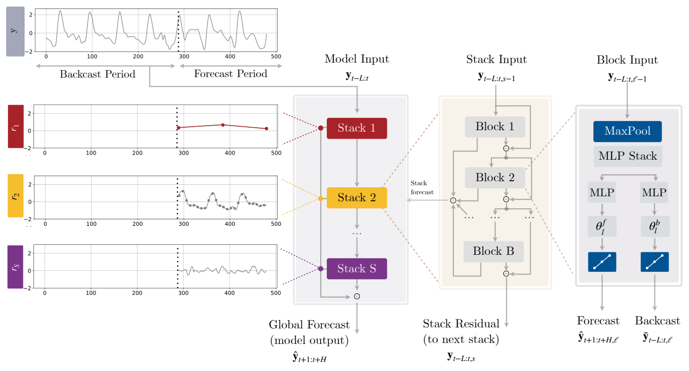


3.5.4 Introduction to N-HiTS

The Neural Hierarchical Interpolation for Time Series (N-HiTS) forecasting model [62] builds upon the foundation of the N-BEATS model. It promises enhanced prediction accuracy while optimizing computational efficiency. The model's standout feature is hierarchical interpolation, which capitalizes on multi-rate signal sampling to account for both immediate and extended effects in a time series.

Figure 11

The N-HiTS architecture, evolving from the N-BEATS approach, showcases a hierarchical design. Central to its structure are blocks, furnished with multilayer perceptron (MLP) layers, that produce backcast and forecast coefficients. These blocks are assembled into stacks, each fine-tuned to discern distinct data features through tailored basis functions. The model's potency is further augmented by a multi-rate sampling technique and multi-scale synthesis, bolstering its long-term forecasting efficiency. [62]



y
Backcast Period
Forecast Period
r_1
r_2
r_S
Model Input $\mathbf{y}_{t-L:t}$
Stack 1
Stack 2
Stack S
Global Forecast (model output) $\hat{\mathbf{y}}_{t+1:t+H}$
Stack Input $\mathbf{y}_{t-L:t,s-1}$
Block 1
Block 2
Block B
Stack forecast
Stack Residual (to next stack) $\mathbf{y}_{t-L:t,s}$
Block Input $\mathbf{y}_{t-L:t,\ell-1}$
MaxPool
MLP Stack
MLP
MLP
θ_l^f
θ_l^b
Forecast $\hat{\mathbf{y}}_{t+1:t+H,\ell}$
Backcast $\tilde{\mathbf{y}}_{t-L:t,\ell}$

3.5.4.1 Architecture of N-HiTS Similar to N-BEATS, N-HiTS comprises an array of blocks and stacks that handle both forecast and backcast operations. A salient innovation in N-HiTS is the MaxPool layer integrated at the block level, facilitating multi-rate sampling. This mechanism adjusts the kernel size to concentrate on varying temporal series effects, markedly boosting its proficiency in long-range forecasting.

3.5.4.2 Mechanism of Hierarchical Interpolation Hierarchical interpolation, pivotal to N-HiTS, fuses forecasts from diverse time scales, streamlining prediction cardinality. Each stack is governed by a unique ratio that determines the frequency of predictions, setting apart the cardinality across the board. Such an arrangement enhances computational efficiency and sharpens accuracy, particularly for expansive forecast scopes.

3.5.4.3 Fully Connected Networks and Data Propagation After the MaxPool layer, N-HiTS harnesses fully connected networks to derive forecast and backcast outputs. The backcast elucidates the block's insights and, once deducted from the input, is channeled to the succeeding block through residual links. This structure ensures comprehensive data capture, refining the model's precision.

3.5.4.4 Conclusion N-HiTS elevates the N-BEATS architecture by incorporating a MaxPool layer, enabling diverse time scale analyses for each stack. This addition empowers the model to discern both immediate and extended time series nuances. By consolidating predictions via hierarchical interpolation, N-HiTS emerges as a lightweight yet high-accuracy model, marking it as a vanguard in time series forecasting.

Chapter 4
Data Collection and Pre-processing

4.1 Data Sources

The dataset utilized in this study encompasses electricity consumption (or load), wind generation, and solar generation, sourced from the Open Power System Data (OPSD) project [63]. This comprehensive dataset is a product of the ENTSO-E Transparency platform, an initiative by the European Network of Transmission System Operators for Electricity. The data, which spans from 2015 to mid-2020, is instrumental in understanding the energy landscape and consumption patterns in Europe, particularly Spain.

The specific file extracted for analysis is `time_series_60min_singleindex.csv`. The columns of focus for the country of Spain include:

- `utc_timestamp`: Represents the Universal Time Coordinated timestamp for each data entry.
- `ES_load_actual_entsoe_transparency`: Refers to the actual electricity load or consumption for Spain.
- `ES_wind_generation_actual`: Denotes the actual wind energy generated within Spain.
- `ES_solar_generation_actual`: Signifies the actual solar energy generated in Spain.

Preliminary data analysis revealed the presence of some missing values within the dataset. To address this, a two-step imputation method was applied:

- First, missing values were forward filled, replacing any ‘NaN‘ with the previous non-missing value in the column.
- For entries where the initial values were missing, a backward fill was applied, replacing the ‘NaN‘ with the subsequent non-missing value.

This approach ensures a comprehensive dataset, leveraging actual data points to fill gaps, thereby maintaining the integrity and continuity of the data.

4.2 Data Cleaning and Transformation

Preliminary data analysis revealed the presence of some missing values within the dataset. To ensure data completeness and maintain its integrity, the following cleaning and imputation methods were applied:

- **Handling Missing Values**: A two-step imputation method was employed:
  - Missing values were forward filled, replacing any ‘NaN‘ with the previous non-missing value in the column.
  - For entries where the initial values were missing, a backward fill was applied, replacing the ‘NaN‘ with the subsequent non-missing value.
- **Data Scaling**: The data was scaled to ensure uniformity and to facilitate better model convergence. Two types of scalers were experimented with:
  - **MinMaxScaler**: This scales the data by transforming it to a range between 0 and 1.
  - **MaxAbsScaler**: This scales the data such that the maximum absolute value of each feature is scaled to unit size.

This approach ensures a comprehensive dataset, leveraging actual data points to fill gaps, thereby maintaining the integrity and continuity of the data.

4.3 Encoding and Feature Engineering

To enhance the predictive capabilities of the models and capture underlying patterns in the data, several encoding techniques and feature engineering steps were employed:

- **Cyclic Encoding**: To capture the seasonality inherent in time series data, Fourier transformations were used to create cyclic features. These are represented mathematically as:
  - Yearly Fourier Cosine: $\cos\left(\frac{2\pi\times\text{day of year}}{365}\right)$
  - Yearly Fourier Sine: $\sin\left(\frac{2\pi\times\text{day of year}}{365}\right)$
  - Daily Fourier Cosine: $\cos\left(\frac{2\pi\times\text{hour of day}}{24}\right)$
  - Daily Fourier Sine: $\sin\left(\frac{2\pi\times\text{hour of day}}{24}\right)$
- **Datetime Attributes**: Extracted temporal features from timestamps to capture potential cyclical patterns and trends at different granularities.
- **Holiday Indicator**: A binary feature using the python library [64], $h(t)$, where:

$$h(t) = \begin{cases} 1 & \text{if } t \text{ is a holiday in Spain} \\ 0 & \text{otherwise} \end{cases}$$

- **Rolling Statistics**: To capture short and long term fluctuations in the data, several rolling statistics were computed:
  - 24-hour Rolling Mean:

$$\mu_{24}(t) = \frac{1}{24}\sum_{i=0}^{23} x(t-i)$$

- 24-hour Rolling Standard Deviation:

$$\sigma_{24}(t) = \sqrt{\frac{1}{24}\sum_{i=0}^{23}(x(t-i)-\mu_{24}(t))^2}$$

- 24x7-hour (Weekly) Rolling Mean:

$$\mu_{168}(t) = \frac{1}{168}\sum_{i=0}^{167}x(t-i)$$

- 24x7-hour (Weekly) Rolling Standard Deviation:

$$\sigma_{168}(t) = \sqrt{\frac{1}{168}\sum_{i=0}^{167}(x(t-i)-\mu_{168}(t))^2}$$

- 24x7x4-hour (Monthly) Rolling Mean:

$$\mu_{672}(t) = \frac{1}{672}\sum_{i=0}^{671}x(t-i)$$

- 24x7x4-hour (Monthly) Rolling Standard Deviation:

$$\sigma_{672}(t) = \sqrt{\frac{1}{672}\sum_{i=0}^{671}(x(t-i)-\mu_{672}(t))^2}$$

- **Lags**:
  - For models such as XGBoost and LightGBM, lag features were used to provide historical context. For a given time t and lag k, the lag feature is represented as $x(t-k)$. Values used were from 1 hour to 3 days (72 hours)

Chapter 5

Exploratory Data Analysis

5.1 Statistical Test Analysis

5.1.1 Overview of Time-Series Test Analysis

This section examines the properties of three time series: `ES_load`, `ES_solar`, and `ES_wind`. Various statistical methods are employed to investigate their stationarity, mean, variance, autocorrelation, cointegration, and Granger causality.

5.1.2 Stationarity of the Time-Series: Augmented Dickey-Fuller (ADF) Test

The Augmented Dickey-Fuller (ADF) test was applied to determine the stationarity of each series. The test results strongly suggest the series are stationary, with all p-values below 0.001.

Table 1

ADF test results for stationarity of time series

Time Series	ADF Statistic	p-value
`ES_load_actual`	-17.61	3.86×10^{-30}
`ES_solar`	-9.61	1.80×10^{-16}
`ES_wind`	-16.88	1.06×10^{-29}

5.1.3 Cointegration in the Time-Series - Johansen Test

The Johansen test was employed to assess cointegration among the series. The results, as presented in the table below, strongly indicate the presence of long-term equilibrium relationships among the series.

The trace statistics are significantly larger than the critical values at all confidence levels (90%, 95%, and 99%), suggesting rejection of the null hypothesis of no cointegration. This implies the presence of cointegrating relationships among the series.

Furthermore, the eigenvalues, which measure the strength of the cointegration relationships, show decreasing magnitudes, which is expected. The associated eigenvectors can be used to form the cointegrating vectors, which represent the long-term relationships among the series.

Table 2

Johansen Test Results

Metric	**ES load**	**ES solar**	**ES wind**
Trace Statistic	35587.34	16347.89	7236.05
Critical Values (90%)	27.07	13.43	2.71
Critical Values (95%)	29.80	15.49	3.84
Critical Values (99%)	35.46	19.93	6.63
Eigenvalues	0.318	0.165	0.134
Eigenvector 1	6.16×10^{-4}	5.65×10^{-5}	-5.56×10^{-4}
Eigenvector 2	-1.58×10^{-3}	1.39×10^{-4}	-1.04×10^{-3}
Eigenvector 3	-3.42×10^{-4}	2.95×10^{-3}	5.94×10^{-5}
Eigen Statistics	19239.44	9111.84	7236.05

5.1.4 Granger Causality Test

The Granger causality test was conducted to ascertain the potential causal relationships between the variables. The tests revealed significant results that indicate predictive causality among the series.

Table 3

Granger Causality Test Results for Lags 1 and 2

	ES_load	ES_solar	ES_wind
ES_load	-	Lag 1: F: 1.71, p: 1.91×10^{-1}; Lag 2: F: 877.71, p: $< 1 \times 10^{-3}$	Lag 1: F: 189.50, p: $< 1 \times 10^{-3}$; Lag 2: F: 65.44, p: $< 1 \times 10^{-3}$
ES_solar	Lag 1: F: 86.95, p: $< 1 \times 10^{-3}$; Lag 2: F: 3599.82, p: $< 1 \times 10^{-3}$	-	Lag 1: F: 826.19, p: $< 1 \times 10^{-3}$; Lag 2: F: 505.70, p: $< 1 \times 10^{-3}$
ES_wind	Lag 1: F: 1725.62, p: $< 1 \times 10^{-3}$; Lag 2: F: 262.95, p: $< 1 \times 10^{-3}$	Lag 1: F: 4227.09, p: $< 1 \times 10^{-3}$; Lag 2: F: 579.56, p: $< 1 \times 10^{-3}$	-

The results show that ES_load and ES_solar series exhibit bidirectional causality. Similarly, bidirectional causality is observed between ES_load and ES_wind. A significant causal relationship is also noted between the `ES_solar` and `ES_wind` series.

5.1.5 Variance Inflation Factor (VIF)

The Variance Inflation Factor (VIF) was computed to evaluate multicollinearity among the predictor variables in the model. VIF quantifies how much a variable is inflating the standard errors due to multicollinearity. Generally, a VIF above 5-10 indicates a problematic amount of collinearity. The VIF values for the series suggest no significant multicollinearity issues among the variables.

Variable	**Value**
`ES_load`	5.50
`ES_solar`	1.93
`ES_wind`	3.94

Table 4

VIF values assessing multicollinearity

5.1.6 Statistical Properties of the Time Series

The Durbin-Watson statistic was utilized to detect serial autocorrelation. The results suggest a positive autocorrelation for all series. The Durbin-Watson values close to zero indicate strong positive autocorrelation in all time series. The presence of autocorrelation indicates that future statistical modeling should account for this aspect, possibly through differencing the series or using ARIMA models.

Series	**Mean**	**Var.**	**D-W**
`ES_load`	28441.60	2.14×10^7	0.002
`ES_solar`	1613.87	3.86×10^6	0.056
`ES_wind`	5559.90	1.07×10^7	0.004

Table 5

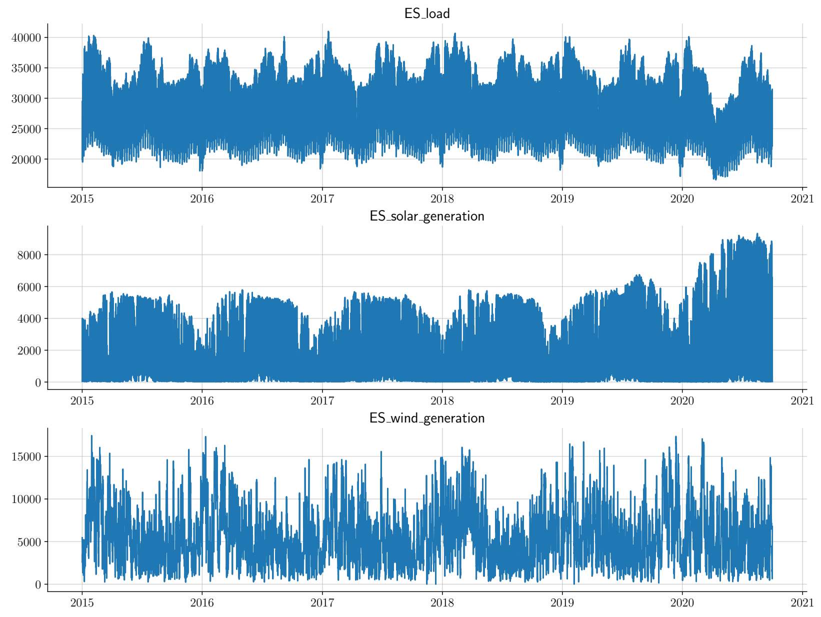
Statistical properties of time series

5.2 Visualizing the Data

5.2.1 Line Plots

Figure 12

Line Plots of ***`ES_load`***, ***`ES_solar_generation`***, *and* ***`ES_wind_generation`***



The line plots in Figure 12 provide an initial visual representation of the data's evolution over a five-year span.

Table 6

Observations from Figure 12

Characteristic	**ES Load**	**ES Solar Generation**	**ES Wind Generation**
Description	Visual Analysis	Visual Analysis	Visual Analysis
Overall Trend	Stationary	Rising Trend	Stationary
Seasonality	Monthly Patterns	Yearly Seasonality	Winter Peaks
Anomalies (2020)	Decline (COVID-19)	Surge (COVID-19)	Not Observed
Correlations	Yearly Lows	Seasonal Variations	Not Specified

5.2.2 Box Plots

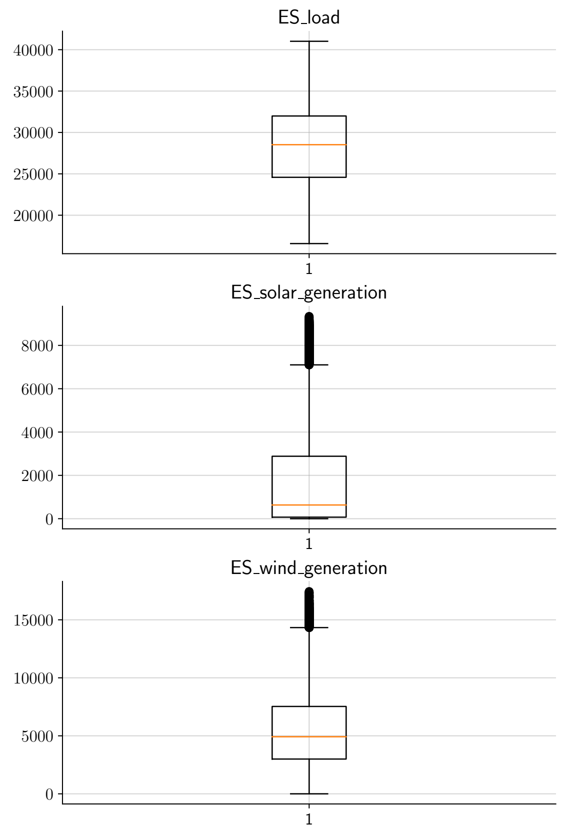


Figure 13

Box Plots

Metric	**ES load**	**ES solar**	**ES wind**
Median	28522.0	633.0	4922.0
Mean	28441.0	1613.7	5560.6
IQR	7413.0	2813.0	4533.0
Range	24440.0	9338.0	17436.0
Skewness	0.0722	1.277	0.799

Table 7

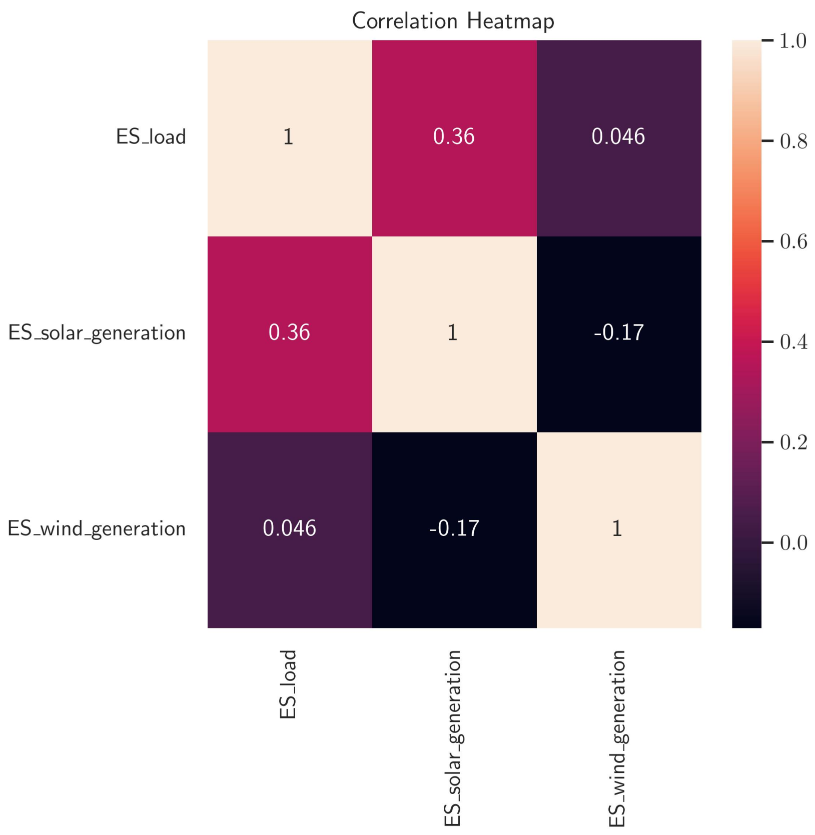
Box Plot Metrics

Figure 13 shows box plots, which provide information about the distribution and variability of the data. The box represents the interquartile range, and the whiskers extend to the minimum and maximum values.

5.2.3 Correlation Heatmap Analysis

Figure 14

Correlation Heatmap



The heatmap in Figure 14 visualizes the correlation matrix among various variables. Brighter colors represent stronger correlations, while darker hues indicate weaker associations. Key observations include:

- **`ES_load` and `ES_solar`**: A coefficient of 0.36 signifies a moderate positive correlation. This suggests a tendency for higher `ES_load` values to coincide with increased `ES_solar` values, although the relationship isn't robust.

- **`ES_load` and `ES_wind`**: With a coefficient of 0.047, there's a negligible positive correlation, indicating a minimal relationship between the two variables.

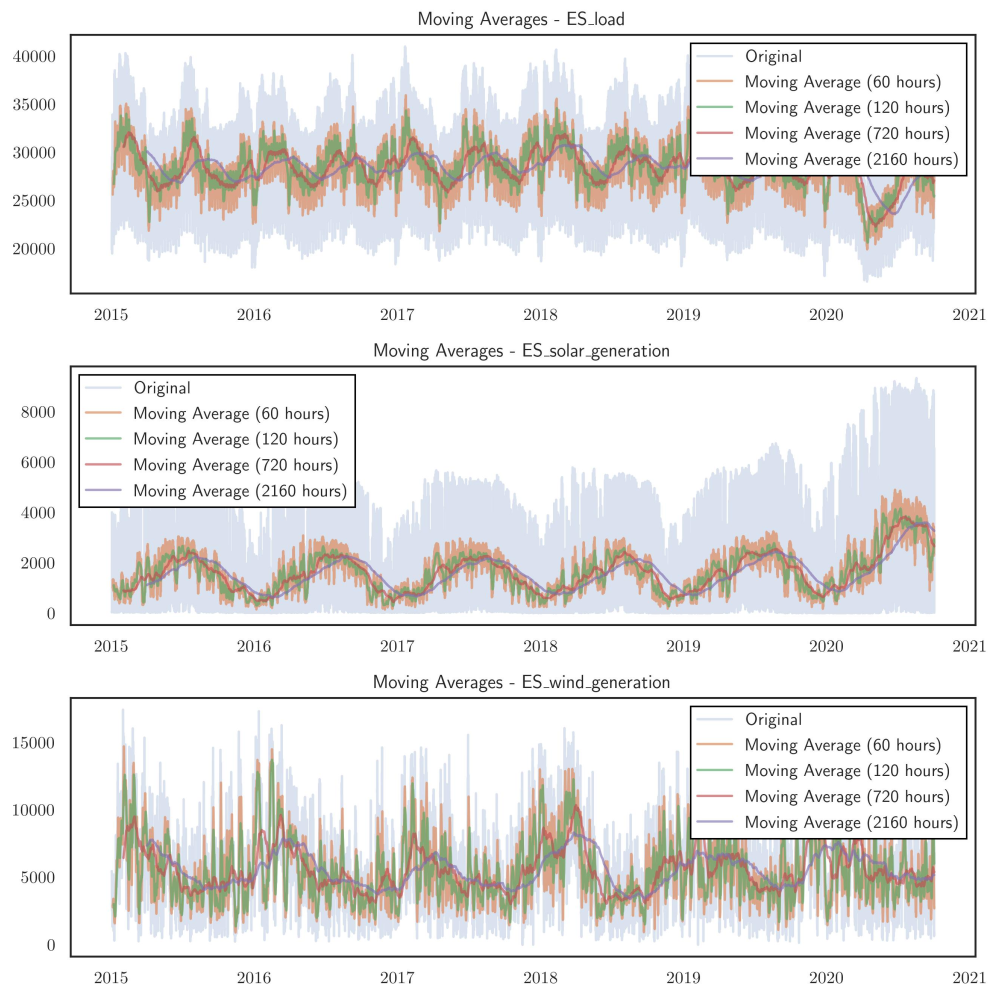
- **`ES_solar` and `ES_wind`**: A coefficient of -0.17 points to a weak negative correlation, implying that higher `ES_solar` values generally correspond to lower `ES_wind` values.

The correlation data suggests that while solar energy production aligns somewhat with energy demand, wind energy might not consistently match the load requirements.

5.2.4 Moving Average Plots in Multiple Window Size

Figure 15

Moving Averages with Multiple Window Size

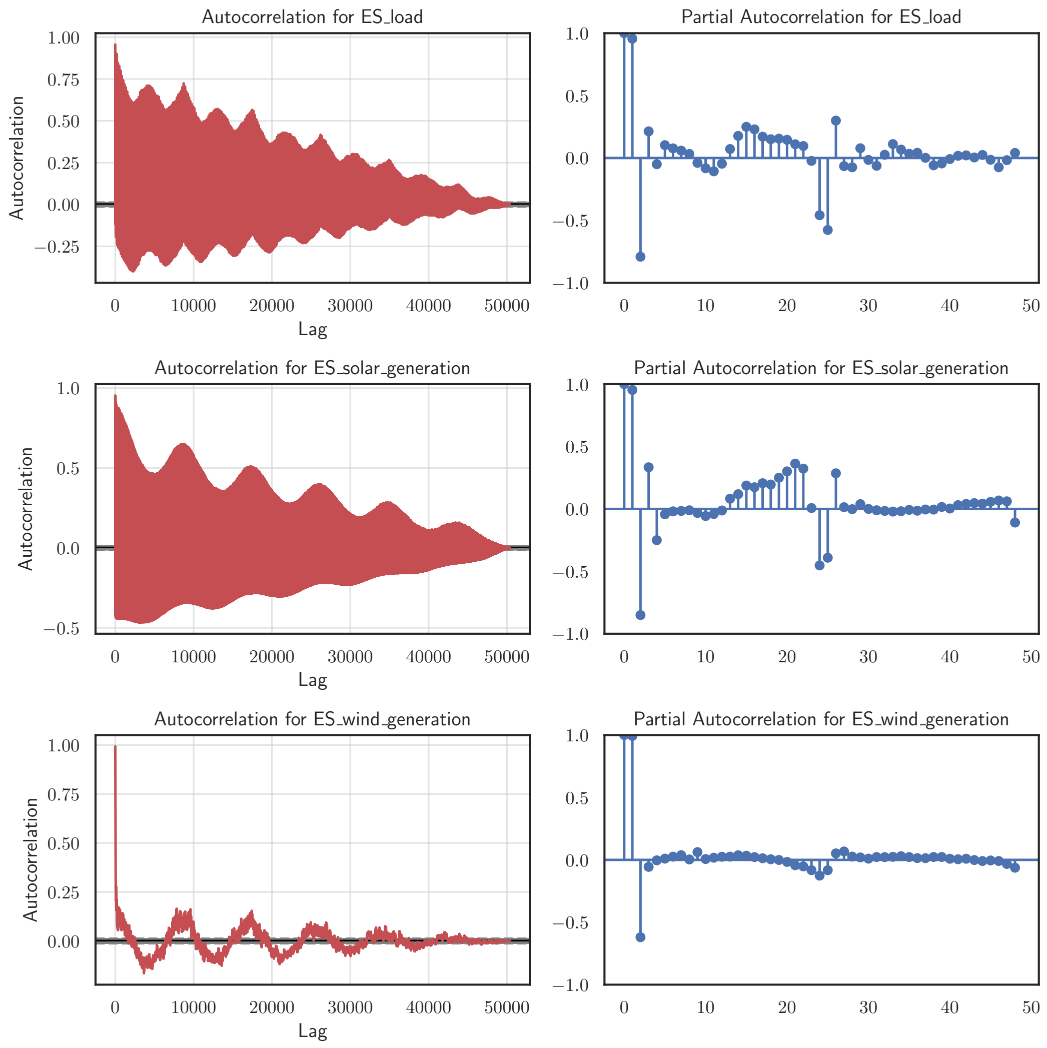


The moving average plot, as depicted in Figure 15, serves as an instrumental tool for discerning the overarching trends or patterns in time series data. By calculating the average for a designated window of observations and plotting it against the associated time points, this plot effectively mitigates short-term fluctuations and accentuates the long-term trend or pattern inherent in the dataset. Evident trends can be observed for `ES_actual`, `ES_solar`, and `ES_wind`.

5.2.5 Autocorrelation and Partial Autocorrelation Plots

Figure 16

Autocorrelation and Partial Autocorrelation Plots



Autocorrelation and partial autocorrelation plots serve as instrumental tools in discerning patterns and relationships in time series data across different lags.

Autocorrelation Plot:

- *Positive coefficients*: Denote a direct relationship between observations.
- *Negative coefficients*: Indicate an inverse relationship.
- *Slow decay*: Suggests a potential trend or long-term dependency.
- *Alternating coefficients*: Signify seasonality.
- *Rapid decay*: Highlights limited correlation between observations.

Partial Autocorrelation Plot:

- *Significant coefficient at lag k*: Implies a direct relationship at that specific lag, independent of other lags.
- *Rapid decay*: Points to minimal direct relationships across multiple lags.

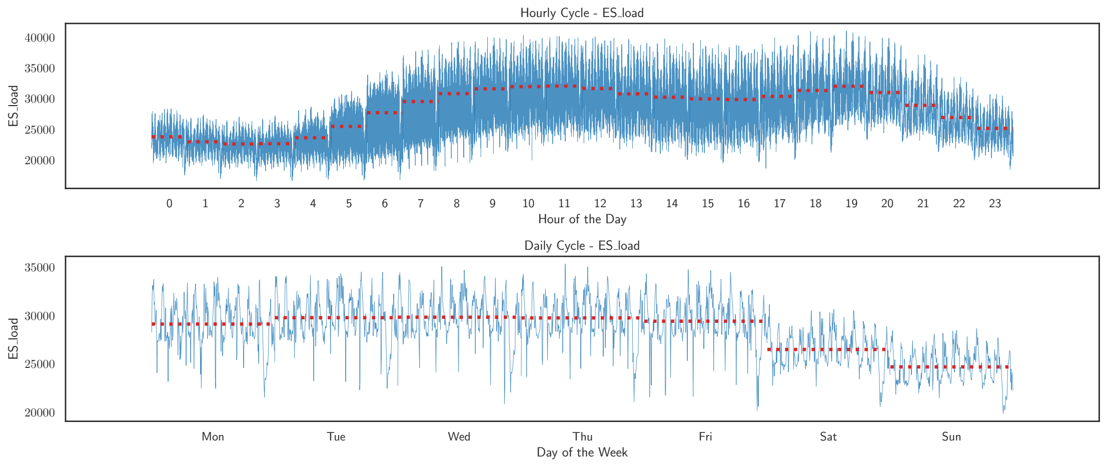
5.2.6 Hourly, Daily and Weekly Cycle Plots

5.2.6.1 Cycle Plots for Load

5.2.6.1.1 Hourly and Daily Cycle Plots for Load The hourly and daily cycle plots, shown in Figure 17, depict the variation in load over a 24-hour period and across different days of the week. The load demonstrates a bimodal distribution, with peaks observed typically in the morning and evening. These peaks coincide with the commencement and conclusion of standard work hours. During nighttime, there is a noticeable decrease in load, which can be attributed to the closure of most businesses and a reduction in residential consumption. This pattern remains consistent throughout the weekdays. However, during weekends, the load pattern

Figure 17

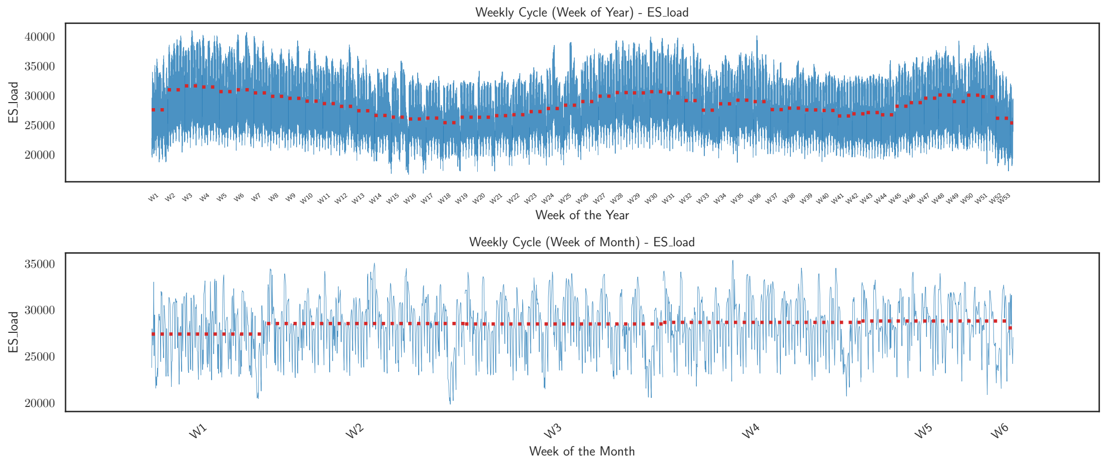
Hourly and Daily Cycle Plots for Load



shifts slightly, rising later in the morning and remaining relatively steady throughout the day. This suggests a consistent residential demand and a decline in commercial or industrial activity during weekends.

Figure 18

Weekly Cycle Plots for Load

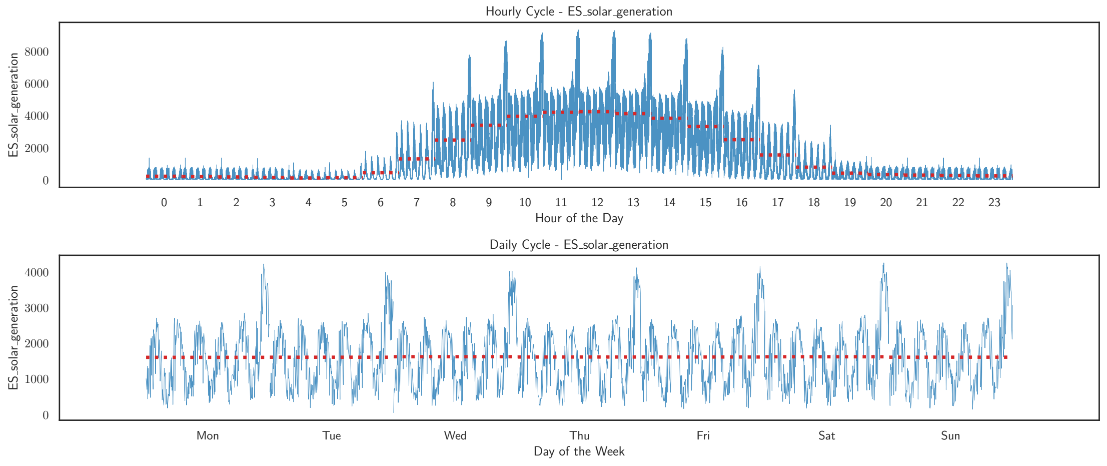


5.2.6.1.2 Weekly Cycle Plots for Load The weekly cycle plot Figure 18 illustrates the fluctuations in load across various weeks within a month and throughout the year. A consistent pattern in the weekly behavior of the load can be discerned from the graph. Similar to the daily cycles, weekdays tend to have elevated electricity demand, largely driven by commercial and industrial activities. Conversely, weekends experience a reduced demand. Variations in load based on the week of the month or the year likely indicate typical business cycles or the influence of holidays and other significant events over the course of the year.

5.2.6.2 Cycle Plots for Solar Generation

Figure 19

Hourly and Daily Cycle Plots for Solar Generation

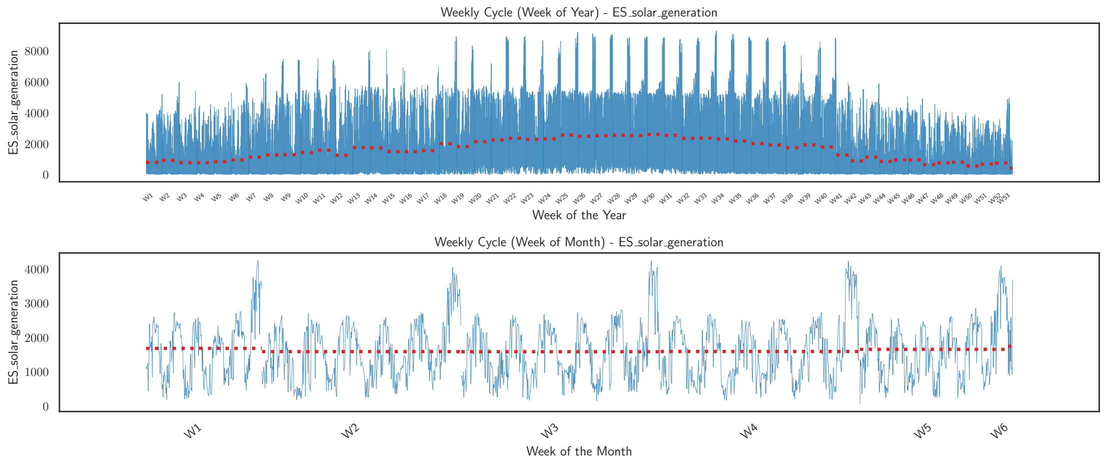


5.2.6.2.1 Hourly and Daily Cycle Plots for Solar Generation

The hourly and daily cycle plots, as depicted in Figure 19, demonstrate the fluctuations in solar generation throughout a 24-hour period and across varying days of the week. The graphical representation reveals a distinct pattern with heightened values during daylight hours and an absence of generation at nighttime. This pattern is indicative of the diurnal variation inherent to solar energy production. Additionally, the daily cycle underscores that solar generation remains relatively consistent between weekdays and weekends. This consistency is attributed primarily to meteorological factors rather than human activity.

Figure 20

Weekly Cycle Plots for Solar Generation



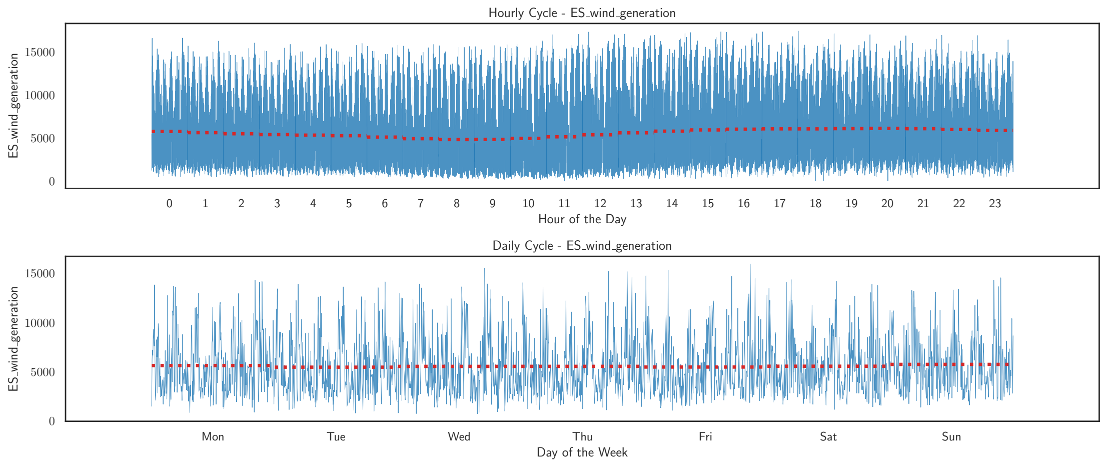
5.2.6.2.2 Weekly Cycle Plots for Solar Generation The weekly cycle plot, illustrated in Figure 20, delineates the fluctuations in solar generation throughout different weeks of both the month and year. The representation exhibits seasonal variations in solar generation, characterized by amplified values during the summer and diminished values in the winter. Such a trend is indicative of the seasonal shifts in solar irradiation. Solar panels are typically more productive during the sunnier and extended daylight hours of summer, in contrast to the shorter and less sunny days of winter.

5.2.6.3 Cycle Plots for Wind Generation

5.2.6.3.1 Hourly and Daily Cycle Plots for Wind Generation The hourly and daily cycle plots, represented in Figure 21, depict the fluctuations in wind generation over a 24-hour period and throughout the various days of the week. The illustrations reveal a consistent level of wind generation, without substantial diurnal variations or pronounced fluctuations in wind speed. Wind generation levels predominantly remain stable throughout the day, suggesting a consistent wind energy supply. A slight decline during midday might be attributed to elevated temperatures.

Figure 21

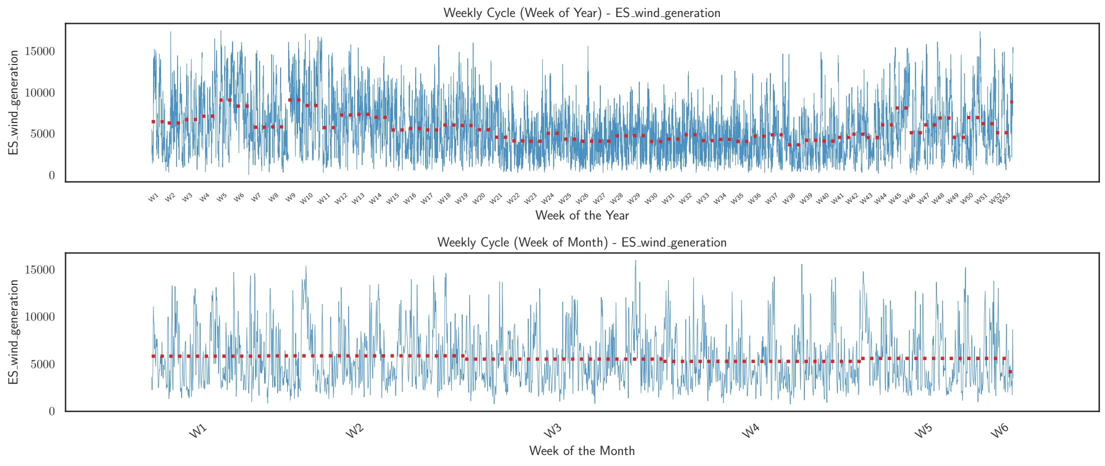
Hourly and Daily Cycle Plots for Wind Generation



Notably, the consistency in wind generation between weekdays and weekends implies that such generation is more so influenced by meteorological factors than by human activities.

Figure 22

Weekly Cycle Plots for Wind Generation



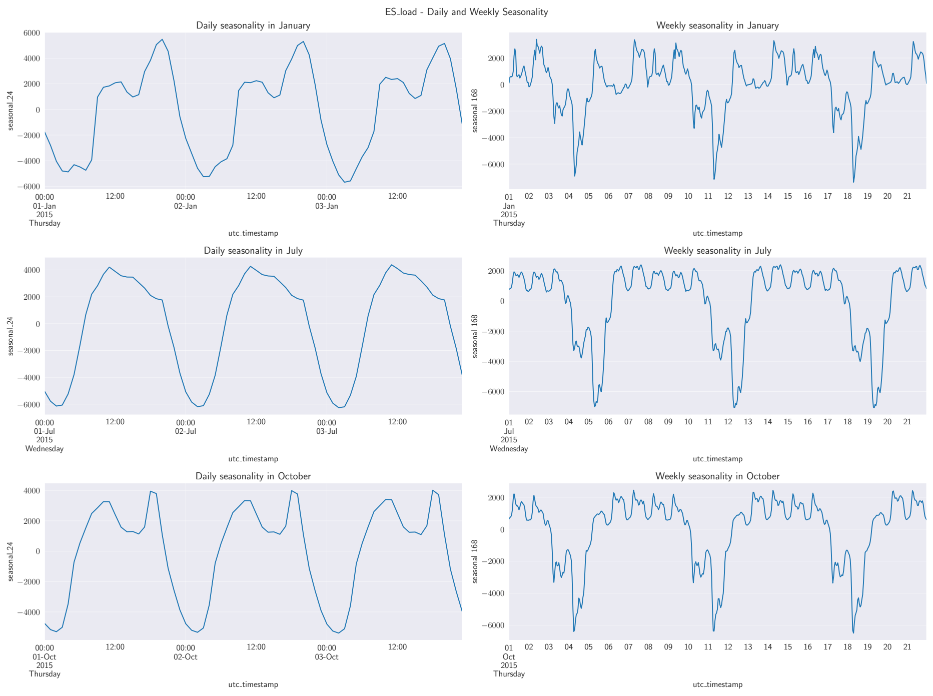
5.2.6.3.2 Weekly Cycle Plots for Wind Generation The weekly cycle plot showcased in Figure 22 elucidates the fluctuations in wind generation throughout different weeks of the month and the year. Distinct seasonal patterns emerge, with elevated values predominantly observed during the autumn and winter months. The surge in average wind speed during these colder periods is less about the temperature itself and more about the temperature differential between polar and equatorial regions. This heightened temperature gradient fosters the development of more significant weather systems, culminating in increased wind generation.

5.2.6.4 Seasonal Plots

5.2.6.5 Daily and Weekly Seasonality Analysis for Actual Load Figure 23 delineates the daily and weekly seasonalities observed in January, July, and October, providing insights into the cyclical nature of the actual load data.

Figure 23

Daily and Weekly Seasonality Analysis for Actual load



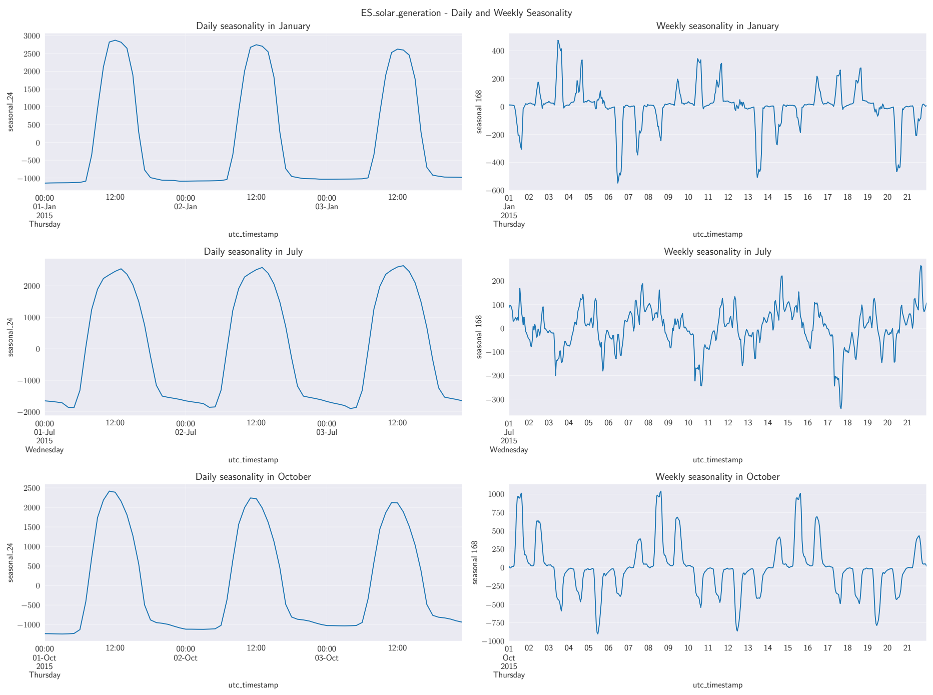
In January (specifically, January 1, 2015, which was a Thursday), the daily seasonality is characterized by a bimodal distribution, signifying heightened demand during specific hours. The weekly seasonality underscores the contrasts in electricity consumption across the weekdays.

July's seasonality, taken from July 1, 2015 (a Wednesday), varies from January's. The daily load demand manifests a steady escalation towards midday. In contrast, the weekly fluctuations echo those discerned in January.

October's pattern (October 1, 2015 - Thursday) diverges again, with daily seasonality indicating a modest midday surge followed by a pronounced afternoon peak. Such patterns encapsulate the variations in daily electricity consumption across diverse months. Analogous to the earlier months, October's weekly seasonality outlines the disparities in demand throughout the week.

Figure 24

Daily and Weekly Seasonality Analysis for Solar Generation



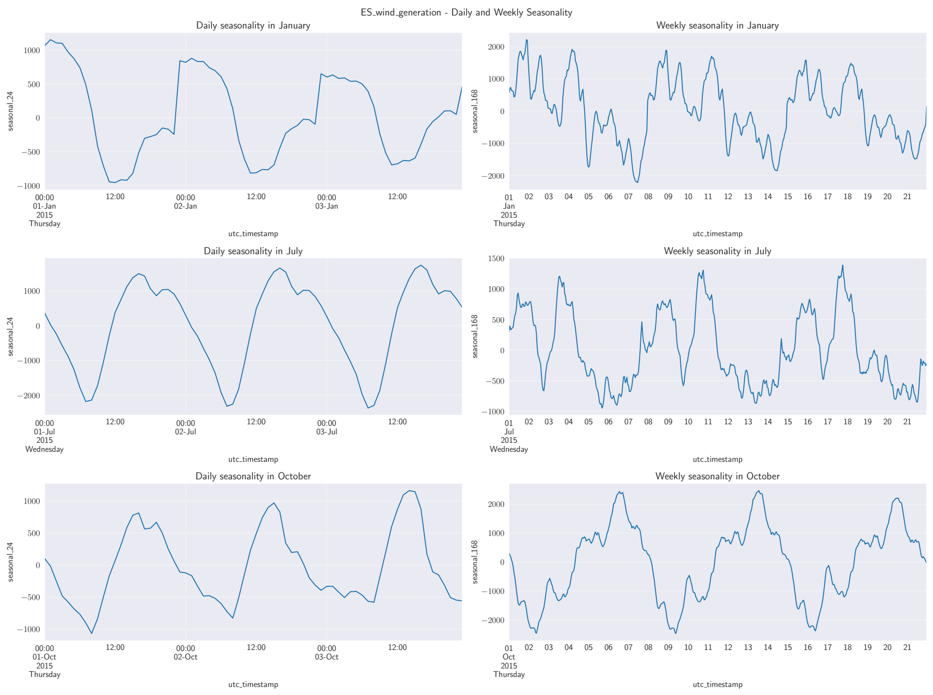
5.2.6.6 Daily and Weekly Seasonality Analysis for Solar Generation

Figure 24 depicts the daily and weekly seasonal trends associated with solar energy generation. The diurnal patterns are unmistakably evident, with solar generation reaching its zenith around noon and plummeting to nil during nocturnal hours.

On a weekly basis, a slight elevation in generation can be discerned during weekdays compared to weekends. This could be attributed to varying electricity demand and consumption behaviors during these intervals.

Figure 25

Daily and Weekly Seasonality Analysis for Wind Generation



5.2.6.7 Daily and Weekly Seasonality Analysis for Wind Generation

Figure 25 elucidates the daily and weekly seasonal patterns inherent to wind power generation.

For the daily cycles in both July and October, peaks are discernible around 13:00, signaling optimum wind generation during these timeframes. July experiences a more extensive range in variation, implying a pronounced presence of wind during the summer months.

Conversely, January's diurnal seasonality showcases troughs slightly earlier, around midnight, with zeniths occurring concurrently. The oscillation range, extending from -1000 to 1000, indicates reduced fluctuations in the winter season.

Regarding weekly patterns, January and July exhibit analogous trends, although a one-day lag is observable for July, likely stemming from the chosen days being Wednesday for July and Thursday for January. October's weekly trends, however, deviate from the prior months, showcasing a consistent 7-day cycle, possibly reflecting stable wind patterns characteristic of the autumn season.

Chapter 6
Model Development and Validation

6.1 Statistical Model Implementation

6.1.1 Introduction

In this chapter, we delve into the application of classical statistical models for forecasting electricity load. Our exploration utilizes the `statsforecast` [65] and Darts library [60] in Python. The statsforecast was used for non-retrain predictions while Darts was used for re-training forecasating. The primary aim is to assess the performance of these models and identify an optimal approach. This analysis not only provides insights into the efficacy of classical models but also establishes a benchmark for subsequent methods. We have considered several models, including ARIMA, MSTL, Dynamic Optimized Theta, AutoETS, and AutoCES, among others.

6.1.2 Data Preparation

Data is the cornerstone of any forecasting model. As such, the initial phase of our work centered on data preparation, predominantly facilitated by the `pandas` library [66]. We ensured the integrity of our time series data by addressing any missing values through techniques such as backward and forward filling, provided by the `bfill` and `ffill` methods of the `pandas` library.
For the purpose of model training and evaluation, we partitioned the dataset into distinct sets:

- Training set: 96%
- Test set: 4%

This partitioning aligns with the sets used in our machine learning and deep learning models. Notably, while machine learning and deep learning models necessitated a separate validation set, we incorporated this set into the training data for our classical methods. The test set spans a duration of three months, equivalent to 2,106 hours, providing a substantial window for rigorous model assessment.

6.1.3 Model Configuration

To ensure the precision and reliability of our forecasts, a careful configuration of the models was paramount. The chosen models and their respective configurations are detailed below: Regarding statsforecast library:

- **MSTL (Multiple Seasonal Decomposition of Time Series)**: The model was set to recognize two seasonalities, with lengths of 11 and 24. The trend component of the time series was forecasted using the AutoARIMA model.
- **AutoARIMA**: Two variants were employed. The first had a season length of 11 , while the second was set with a season length of 1.
- **Dynamic Optimized Theta (DOT)**: Configured with a season length of 11.
- **AutoETS**: Set with a season length of 11.
- **AutoCES**: Operated with a season length of 11.
- **AutoTheta**: Utilized a season length of 11.
- **HoltWinters**: Configured for a season length of 11.

Regarding Darts library, the defaults values were used. The models were:

- AutoARIMA
- AutoCES

- AutoETS
- AutoTheta

6.1.4 Forecasting

Our tools were `historical_forecast` method from darts library, and `predict` from statsforecast library. The forecasts were conducted under two distinct scenarios: one where the model was retrained at each stride and another where it wasn't retrained.

6.1.5 Forecasting

For the forecasting phase, we utilized two primary tools: the `historical_forecast` method from the Darts library and the `predict` method from the Statsforecast library. These tools were instrumental in generating forecasts using our statistical models, with the retraining process being an essential component to ensure accuracy and reliability over time.

6.1.5.1 Forecast Methodology Initially, our statistical models were configured as described in earlier sections. We employed the `historical_forecast` method from the Darts library and the `predict` method from the Statsforecast library to conduct the forecasting. Given the nature of statistical models, retraining at each stride is integral to achieving precise and reliable forecasts, which is why darts library was used for retraining. Retraining the Auto models is only supported from darts library and not natively in statsforecast library.

6.1.5.2 Forecast Settings and Parameters To commence the forecasting, we determined the starting point, denoted as ‘start‘, using the following formula:

$$\text{start} = \text{split\_train} \times (2 - \text{split\_val})$$

Here, both ‘split_train‘ and ‘split_val‘ are assigned a value of 0.8, resulting in a ’start’ value of 0.96. We allocated 80% of the data for training, subdividing the remaining 20% into validation (16

The forecast settings were carefully chosen to facilitate a rigorous evaluation of the forecasting process, detailed as follows:

- **Forecast Horizon:** A forecast horizon of 2 units was designated to define the period covered by the forecasts.
- **Stride:** A stride of 2 units was selected to dictate the step size during the forecasting process.
- **Training Length:** The training length was fixed at 336 hours, equivalent to two weeks, establishing the timeframe for the training data utilization in the forecasts.

6.1.5.3 Retraining Scenarios Considering the statistical models, retraining at each stride is a prerequisite, allowing the models to adapt continually to new data, thereby potentially enhancing forecast accuracy.

6.1.5.4 Evaluation and Analysis Upon completion of the forecasting process, we conducted a detailed analysis of the results, assessing performance through various metrics and visualizations.

6.2 Machine Learning Model Implementation

6.2.1 Introduction

In this section, we explore the use of machine learning models to forecast electricity load utilizing the Darts library [60]. This library stands out for its user-friendly and efficient approach to time series analysis [67]. Our objective is to analyze different model configurations to identify the most effective one for precise forecasting.

6.2.2 Data Preparation

The first phase of our implementation involves preparing the data, primarily using the pandas library [66]. We loaded the dataset and filled in missing values using the `bfill` and `ffill` methods from pandas library, ensuring a consistent time series dataset. Next, we converted the data from a DataFrame to a TimeSeries object to comply with the Darts library's requirements. Additionally, we changed the data type to `np.float32` to speed up the computational processes.
To build and assess the models, we divided the dataset into training, validation, and test sets with the following proportions:

- Training set: 80%
- Validation set: 16%
- Test set: 4%

We defined the forecast horizon to coincide with the length of the test set, covering a period of 3 months or 2106 hours.

6.2.3 Feature Engineering and Scaling

Before incorporating the data into the models, we performed necessary feature engineering steps. When past covariates were included in the model, we scaled them using the `MinMaxScaler` from the `sklearn.preprocessing` module [68], preventing any past covariate from having an improper influence due to a larger magnitude.
For more details on the use of encoders in the models, please refer to section 4.3.

6.2.4 Model Configuration

In this study, we utilized three prominent machine learning models: XGBoost, LightGBM, and RandomForest. Each model was tested with four different configurations to evaluate the effect of encoders and covariates on forecast accuracy. The configurations were:

- With encoders and past covariates
- With encoders but without past covariates
- Without encoders but with past covariates
- Without encoders and without past covariates

Next, the configuration details for each model:

6.2.4.1 XGBoost The XGBoost model, from the XGBoost library [51], was set up with specific lag values ranging from 1 to 72 hours to account for daily seasonal patterns. Other settings include defining the objective as `reg:squarederror` and choosing `gbtree` as the booster type to employ tree-based models.

6.2.4.2 LightGBM The LightGBM model, sourced from the LightGBM library [52], was configured similarly to the XGBoost model with lag values between 1 and 72 hours. The objective was set to optimize regression tasks using the L2 loss function, and the boosting type was defined as `gbdt`, which utilizes gradient boosting decision trees.

6.2.4.3 RandomForest The RandomForest model, part of the scikit-learn library, was aligned with specific lag values ranging from 1 to 72 hours to capture daily patterns in the data. The remaining parameters were kept at their default values.

6.2.5 Forecasting

Our primary tool for this endeavor is the `historical_forecast` method available in the Darts library. This method facilitates the computation of historical forecasts, offering a detailed analysis of the model's performance. Here, we detail the procedures and settings employed in this stage:

6.2.5.1 Forecast Methodology We first fit the our model using its configration, mentioned above. Next, We employed the `historical_forecast` method from the Darts library to conduct the forecasting. The forecasts were conducted under two distinct scenarios: one where the model was retrained at each stride and another where it wasn't retrained.

6.2.5.2 Forecast Settings and Parameters To initiate the historical forecasting, we calculate the starting point, denoted as 'start', using the formula:

$$\text{start} = \text{split\_train} \times (2 - \text{split\_val})$$

Here, 'split_train' and 'split_val' represent the proportions of the data allocated for training and validation, respectively. Both are assigned a value of 0.8. Consequently, the 'start' is calculated as:

$$\text{start} = 0.8 \times (2 - 0.8) = 0.8 \times 1.2 = 0.96$$

Initially, 80% of the data is reserved for training, with the remaining 20% temporarily held for further splitting. This 20% is then divided into validation and test datasets, with 80% going to the validation set and 20% to the test set, making up 16% and 4% of the total dataset, respectively. The forecast settings were meticulously chosen to evaluate the potential advantages of retraining the model at each stride. These settings are as follows:

- **Forecast Horizon:** We set a forecast horizon of 2 units to specify the time period covered by the forecasts.
- **Stride:** We used a stride of 2 units to determine the step size as the model makes predictions step-by-step.
- **Training Length:** The training length was fixed at 336 hours, or two weeks, defining the time span for the training data used in the forecasts.

6.2.5.3 Retraining Scenarios We conducted the forecasting under two different scenarios to ascertain the potential benefits of retraining the model at each stride:

1. **With Retraining:** In this scenario, the models were retrained at each stride, allowing for continual adaptation to new data and potentially enhancing forecast accuracy.
2. **Without Retraining:** In contrast, this scenario maintained a static model throughout the forecasting process, relying solely on the initial training phase for predictions.

6.2.5.4 Evaluation and Analysis Following the forecasting process, we embarked on a detailed analysis of the results, evaluating the performance based on various metrics and visual representations. This step aims to discern the most effective configurations and approaches.

6.3 Deep Learning Model Implementation

6.3.1 Introduction

In this segment, we delve into the implementation of the NHiTS and NBEATS models for forecasting the electricity load. These models are recognized for their robustness and efficiency in time series forecasting. Our goal in this section is to describe various configurations of these models and the methodology used.

6.3.2 Data Preparation

Similar to the machine learning models, the initial phase revolves around data preparation, primarily utilizing the pandas library. The dataset is loaded and any missing values are addressed using appropriate methods from the pandas library to maintain a uniform time series dataset. Subsequently, the data is transformed into a compatible format to meet the requirements of the Darts library, enhancing computational efficiency in the process.
The dataset is partitioned into training, validation, and test sets in the following proportions:

- Training set: 80%
- Validation set: 16%
- Test set: 4%

The forecast horizon is aligned with the test set duration, spanning a time frame of 3 months or 2106 hours.

6.3.3 Feature Engineering and Scaling

Before feeding the data into the models, requisite feature engineering steps are undertaken. Whenever past covariates are incorporated in the model, they are scaled

using appropriate methods to prevent undue influence from larger magnitudes.
For a deeper understanding of encoder utilization in the models, please refer to section section 4.3.

6.3.4 Model Configuration

In this investigation, we focus on two primary models: NHiTS and NBEATS. Both models are experimented with different configurations to assess the influence of encoders and covariates on the accuracy of the forecasts. The configurations being tested are delineated as follows:

- Incorporating encoders and past covariates
- Utilizing encoders but omitting past covariates
- Excluding encoders but including past covariates
- Neither using encoders nor past covariates

Following are the specific configurations for each model:

6.3.4.1 NHiTS The NHiTS model is configured with the following optimized parameters to enhance its forecasting accuracy:

- **Input Chunk Length:** 168 units, which specifies the amount of input data fed into the model at each step.
- **Output Chunk Length:** 24 units, defining the duration covered by each forecast the model generates.
- **Number of Stacks:** 3, indicating the number of stacks in the model, a parameter that influences the model's complexity and depth.
- **Number of Blocks:** 1, representing the number of blocks in each stack

- **Number of Layers:** 2, determining the layers within each block
- **Layer Widths:** 512, which sets the width of the layers,
- **Dropout:** A rate of 0.1, utilized to prevent overfitting by intermittently disabling a portion of the neurons during the training process.
- **Activation Function:** 'ReLU', a function introduced to incorporate non-linearity into the model.
- **Max Pooling:** Enabled (True), a technique employed in the convolutional layers to simplify computation and reduce the training duration by decreasing the spatial dimensions.

6.3.4.2 NBEATS Similarly, the NBEATS model is setup with a 168-unit input chunk length and a 24-unit output chunk length. It employs a generic architecture and other values are kept at their default settings.

6.3.5 Forecasting

The primary tool leveraged in this stage is the `historical_forecast` method available in the Darts library. This method enables the calculation of historical forecasts, providing an extensive analysis of the model's performance over time. Here, we outline the methodologies and settings utilized in this phase:

6.3.5.1 Forecast Methodology Initially, the models are fitted using the above-mentioned configurations. Following this, the `historical_forecast` method is employed to carry out the forecasting, where different scenarios involving retraining at each stride are explored to ascertain the potential advantages.

6.3.5.2 Forecast Settings and Parameters The initiation of historical forecasting involves the calculation of the 'start' variable, which is computed using the equation:

$$\text{start} = \text{split\_train} \times (2 - \text{split\_val})$$

Further details regarding the computation and subdivision of the dataset into training, validation, and test sets are analogous to the machine learning model section.
The forecast settings adopted here are formulated to critically assess the benefits of retraining the model at each stride, comprising of the following elements:

- **Forecast Horizon:** A horizon of 2 units is established to delineate the forecast period.
- **Stride:** A 2-unit stride is chosen to dictate the step size during the forecasting progression.
- **Training Length:** The training duration is pegged at 336 hours, equivalent to a fortnight, which sets the time window for training data usage in the forecasts.

6.3.5.3 Non Retraining and Retraining The forecasting is conducted under different scenarios to gauge the potential benefits of retraining the model at successive strides, detailed as follows:

1. **With Retraining:** This scenario entails retraining the models at every stride, facilitating ongoing adaptation to new data, and possibly enhancing the accuracy of the forecasts.
2. **Without Retraining:** Conversely, this scenario maintains a stationary model throughout the forecasting phase, relying exclusively on the initial training data for predictions.

6.3.5.4 Evaluation and Analysis Upon the completion of the forecasting process, an in-depth analysis of the results is undertaken, evaluating the performance through various metrics and graphical illustrations.

6.4 Model Validation and Performance Metrics

6.4.0.1 Selection of Performance Metrics The validation of predictive models in forecasting research necessitates the implementation of precise performance metrics. These metrics are instrumental in assessing the predictive accuracy of the different forecasting models deployed in this study.

Among various metrics, the Root Mean Squared Error (RMSE) has been chosen as the primary metric for several substantial reasons, which are detailed below:

- **Mathematical Foundation:** RMSE is calculated using the formula

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

  where y_i represents the actual values, $\hat{y}_i$ denotes the predicted values, and n is the number of observations. This formula essentially computes the square root of the average of the squared differences between the actual and predicted values, thus offering a robust measure of model accuracy.

- **Penalization of Larger Errors:** RMSE inherently penalizes larger errors more severely, as the errors are squared before being averaged. This, however, prioritizes minimizing significant deviations from actual values.

- **Interpretability:** The RMSE value is expressed in the same units as the target variable, thereby offering an intuitive understanding of the model's performance and facilitating the comprehension of the real-world implications of forecast errors.

- **Comprehensive Evaluation:** Using RMSE alongside other metrics presents an entire view of the models' performance and averts any potential bias towards specific performance facets.

- **Widespread Acceptance:** RMSE is widely used in the research community as a metric.

To complement the RMSE, other metrics are also employed. The mathematical expressions for these metrics are given as:

- **MAE:**

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

  MAE calculates the average of the absolute errors between the actual and predicted values, offering a linear error penalty.

- **MAPE:**

  $$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

  MAPE computes the average of the percentage errors, thus providing an error measurement in terms of percentage, which is particularly useful in understanding the relative magnitude of the error.

- **MSE:**

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

  MSE represents the average of the squared differences between the actual and predicted values, giving a raw measure of the error magnitude.

- **RMSLE:**

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

  RMSLE is similar to RMSE but operates on the logarithm of the predictions and actual values plus one, thereby being sensitive to relative errors instead of absolute errors.

- **R-Squared (Coefficient of Determination):**

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

  where $\bar{y}$ is the mean of the actual values. R^2 quantifies the proportion of the variance in the dependent variable that is predictable from the independent variable(s), offering a measure of the model's explanatory power.

In conclusion, the choice of RMSE as the primary metric is rooted in its mathematical properties and its alignment with the goals of this research.

Chapter 7
Results and Discussion

7.1 Comparative Analysis of Models

In this section, we conduct an in-depth analysis to compare the performance of various models in forecasting electricity loads. This analysis is crucial in identifying the most effective model for predicting electricity demand with high accuracy. The metric that will be used to compare across metrics is RMSE.

7.1.1 Statistical Forecasing Models

In this subsection, we evaluate the performance metrics of different configurations used in forecasting electricity loads. We specifically focus on the ARIMA, MSTL, Dynamic Optimized Theta, AutoETS, and AutoCES models. A detailed analysis of these models is presented next, however, graphical representations can also be be found in the appendices: For statsforecast library:

- ARIMA Plots: Appendix subsection .10.1
- MSTL Plots: Appendix subsection .10.2
- DOT Plots: Appendix subsection .10.3
- AutoETS Plots: Appendix subsection .10.4
- AutoCES Plots: Appendix subsection .10.5
- AutoTheta Plots: Appendix subsection .10.6
- HoltWinters Plots: Appendix subsection .10.7

For darts library:

- AutoARIMA Plots: Appendix subsection .11.1
- AutoCES Plots: Appendix subsection .11.2
- AutoETS Plots: Appendix subsection .11.3
- AutoTheta Plots: Appendix subsection .11.4

7.1.1.1 Analysis of Statistical Model Forecasts This section is dedicated to analyzing the forecast performance of various statistical models. Two types of forecasts are presented: retrain and non-retrain. A detailed visual depiction of the forecast outcomes is available in appendix section .10.

7.1.1.1.1 Predictions without Retraining The subsequent table showcases the performance indicators of predictions generated using the statsforecast python library. The models under consideration include MSTL, AutoARIMA, Dynamic Optimized Theta, AutoETS, AutoCES, AutoTheta, and HoltWinters. Evaluation metrics encompass MAPE, MAE, MSE, RMSLE, R, and RMSE. The highest-performing model for each metric is distinctly highlighted.

7.1.1.1.2 Overall Performance Analysis Upon analyzing both the graphical representations and the statistical metrics, it is evident that several models are underperforming, particularly highlighted by their negative R^2 values. These negative values signify that the models are far from accurately predicting the variations in the data, in fact, performing worse than a model that would predict the mean value at every point.
Notwithstanding, the Dynamic Optimized Theta together with MSTL model seems to be a leading contender, securing the lowest Mean Squared Error (MSE), indicating a lower average error per prediction. Unfortunately, its negative R^2 value indicates a lack of effectiveness in capturing the underlying trends in the data.

Conversely, the MSTL model performs similary to DOT. Similar to the Dynamic Optimized Theta model, its negative R^2 value highlights a limitation in understanding the intrinsic patterns in the data effectively.

Regrettably, the CES model seems to be the worst perfomant of all, evident from its high error metrics and substantially negative R^2 value.

In summary, while the Dynamic Optimized Theta and MSTL models exhibit strengths in certain areas, their negative R^2 values underline a central limitation in their predictive accuracy. This can be also seen in the figures in the Appendix section .10.

Table 8

Performance Metrics for the different classical models we have used

Model	**MAPE**	**MAE**	**MSE**	**RMSLE**	R^2	**RMSE**
MSTL	13.24	3447.61	1.86×10^7	0.15	-0.01	4309.76
AutoARIMA_11	14.13	3694.36	1.95×10^7	0.16	-0.06	4415.07
AutoARIMA_1	14.74	3792.85	2.10×10^7	0.17	-0.15	4586.00
Dynamic Optimized Theta	13.37	3667.26	1.84×10^7	0.15	-0.01	4293.13
AutoETS	13.32	3692.42	1.87×10^7	0.15	-0.02	4325.11
CES	83.47	23454.50	6.02×10^8	3.38	-31.84	24537.18
AutoTheta	13.36	3667.20	1.84×10^7	0.15	-0.01	4292.28
HoltWinters	13.32	3692.42	1.87×10^7	0.15	-0.02	4325.11

7.1.1.1.3 Predictions with Retraining It's worth noting that the initial performance of the models above seemed unsatisfactory, largely due to the inherent limitations of statistical models in forecasting over extended horizons. Fortunately, the incorporation of retraining methodologies significantly minimized this performance gap. This improvement underscores the utility of the darts library, which facilitates retraining capabilities, thereby enhancing the effectiveness of the statsforecast library in long-term forecasting.

As we can see from the table above, the retraining procedure has a significant impact

Table 9

Model Performance Metrics (Retrain)

Model	MAPE	MAE	MSE	RMSLE	R^2	RMSE
Auto ARIMA	4.164	1149.901	2.283×10^6	0.0548	0.87	1511.005
Auto CES	7.018	1930.168	5.464×10^6	0.086	0.699	2337.590
Auto ETS	6.338	1738.864	4.424×10^6	0.078	0.756	2103.274
Auto Theta	6.447	1769.403	4.566×10^6	0.079	0.749	2136.824

on the performance of the models. Models actually are able to forecast close to the actual values, as seen in the figures in the Appendix section .11. The AutoARIMA model seems to be the best performing model, while AutoCES model is the worst performing model.

7.1.2 Machine Learning Models

In this subsection, we evaluate the performance metrics of different configurations used in forecasting electricity loads. We specifically focus on the LightGBM, RandomForest, and XGBoost models. A detailed analysis of these models is presented next, however, graphical representations can also be be found in the appendices:

- RandomForest Predictions: Appendix section .5
- XGBoost Predictions: Appendix section .6
- LightGBM Predictions: Appendix section .7

7.1.2.1 Random Forest Model Predictions In this section of the analysis, we focus on the Random Forest model. Different setups were examined, including variations with or without encoders, and with or without covariates. A summary

Table 10

Analyzing Random Forest Model Configurations: *The table examines metric variations across Random Forest configurations, considering the inclusion or exclusion of encoders (E) and covariates (C).* ***Color and Formatting Guide:*** *Cells in Melrose and Alto colors highlight the best performers for non-retrain and retrain setups, respectively. Bolded values pinpoint the optimal performance in each metric category.*

		MAPE	MAE	MSE	RMSLE	R2	RMSE
RF NE NC	R	2.902	814.6	1086148.633	**0.038**	0.942	1042.185
	NR	1.236	355.527	219497.089	0.017	0.988	468.505
RF NE WC	R	3.058	849.178	1172335.736	0.04	0.937	1082.745
	NR	**1.216**	**350.216**	**202597.83**	**0.016**	**0.989**	**450.109**
RF WE NC	R	**2.864**	803.97	**1065965.75**	**0.038**	0.943	**1032.456**
	NR	1.228	354.068	216223.193	**0.016**	0.988	464.998
RF WE WC	R	2.866	**792.81**	1081110.558	0.039	0.942	1039.765
	NR	1.232	354.168	241999.316	0.017	0.987	491.934

table highlights key performance metrics such as Mean Absolute Percentage Error (MAPE), Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and the coefficient of determination (R^2).

7.1.2.1.1 No-retrain model The data from the table shows that the Random Forest model without retraining performs best when encoders are excluded and covariates are included (RF NE WC), yielding the lowest values for MAPE, MAE, MSE, and RMSE. It is notable that the R^2 value is the same in two setups (RF NE NC and RF WE NC), indicating a stable predictive power in these cases. Not using encoders suggests that the initial features are strongly predictive, and adding encoders might bring unnecessary complexity or noise, possibly lowering the performance. The worst performance is seen when both encoders and covariates are used, which might make the model too complex and unable to generalize well.

7.1.2.1.2 Retrain model In contrast, the retrained Random Forest model performs optimally when encoders are included and covariates are excluded. This setup might help the model to identify complex patterns in the data that were not captured initially, enhancing performance. Leaving out covariates during the retraining might help the model to focus more on the primary features, avoiding potential noise and the risk of overfitting.

7.1.2.1.3 Conclusion In conclusion, it appears that the Random Forest model does not benefit greatly from using both encoders and covariates at the same time. The best outcomes are observed when either covariates are included and encoders are excluded, or the other way around, depending on whether the model is being retrained or not.

7.1.2.2 XGBoost Model Predictions This section shifts the focus to the XGBoost model, evaluated under similar conditions as those applied to the Random Forest model, concerning the use or non-use of encoders and the inclusion or exclusion of past covariates. In addition to numerical data, the appendix section .6 contains visual representations that ease a comprehensive understanding of the model's performance over different periods, including a detailed view of a one-month segment. The forthcoming table defines the key metrics indicative of the model's performance.

7.1.2.2.1 Analysis of Results Analyzing the table, we can observe various performance levels across different configurations of the XGBoost model. The metrics such as MAPE, MAE, MSE, RMSLE, R^2, and RMSE provide a clear insight into the effectiveness of each configuration.

Table 11

Analyzing XGB Model Configurations: *This table examines metric variations across XGB configurations, considering the inclusion or exclusion of encoders (E) and covariates (C).* ***Color and Formatting Guide:*** *Cells in Melrose and Alto spotlight the best performers for non-retrain and retrain setups, respectively. Bolded values denote the best performance in each metric category.*

		MAPE	**MAE**	**MSE**	**RMSLE**	**R2**	**RMSE**
XGB NE NC	R	2.973	836.783	1246142.9	0.041	0.933	1116.308
	NR	1.438	406.844	288498.66	0.019	0.985	537.121
XGB NE WC	R	2.943	826.011	1165365.0	0.039	0.938	1079.521
	NR	1.338	378.53	247748.16	0.018	0.987	497.743
XGB WE NC	R	2.973	836.783	1246142.9	0.041	0.933	1116.308
	NR	1.438	406.844	288498.66	0.019	0.985	537.121
XGB WE WC	R	**2.813**	**783.709**	**1071577.2**	**0.038**	**0.943**	**1035.17**
	NR	**1.207**	**344.812**	**203175.66**	**0.016**	**0.989**	**450.75**

7.1.2.2.2 No-retrain Model

For the no-retrain model, the configuration XGB WE WC NR exhibits the best performance, yielding the lowest values across most metrics including MAPE, MAE, MSE, and RMSE, indicating a more accurate and reliable model. This suggests that including both encoders and covariates without retraining the model seems to capture the underlying patterns in the data more effectively.

7.1.2.2.3 Retrain Model

On the other hand, the retrained model shines in the XGB WE WC R configuration, where it again achieves the lowest values in several metrics, signaling the potential optimal setup when considering a retraining strategy. This pattern suggests a consistency in the performance of the configuration that includes both encoders and covariates, even when the model is retrained.

7.1.2.2.4 Conclusion To conclude, the analysis indicates that the XGBoost model generally performs better with the inclusion of both encoders and covariates, irrespective of the retraining state. This might be due to the model's proficiency in mitigating overfitting and adeptly recognizing underlying patterns in the data. However, it is notable that the retraining approach seems to slightly reduce the model's performance. Additionally, the identical performance of the XGB WE NC and XGB NE NC configurations is because the model disregards future encoders during the evaluation, maintaining a consistent output.

7.1.2.3 LightGBM Model Predictions In this section, we turn our focus to the analysis of the LightGBM model. Just like the previous models, the LightGBM is analyzed based on different setups: with or without encoders and including or excluding covariates. To enhance the numerical analysis, detailed graphical representations are also provided, presenting a broader view as well as a one-month detailed snapshot to closely examine the model's predictions in comparison to the actual outcomes. Please refer to appendix section .7 for the visual insights.
The ensuing table delineates the crucial metrics that serve as benchmarks to evaluate the model's performance.

7.1.2.3.1 No-retrain Model In the scenario without retraining, the LGBM WE WC configuration emerges as the most promising, demonstrating the lowest values in several metrics including MAPE, MAE, and MSE, and exhibiting the highest R^2 value. This hints at a better adaptability of this configuration in recognizing and adapting to the underlying patterns in the data without the necessity of retraining.

7.1.2.3.2 Retrain Model Conversely, for the retrained models, the LGBM WE WC configuration maintains its superior performance, registering the

Table 12

Analyzing LGBM Model Configurations: *The table examines metric variations across LGBM configurations, taking into account the presence or absence of encoders (E) and covariates (C).* ***Color and Formatting Guide:*** *Cells in Melrose and Alto denote the best performance for non-retrain and retrain setups, respectively. Bolded values mark the optimal performance in each metric category.*

		MAPE	**MAE**	**MSE**	**RMSLE**	**R2**	**RMSE**
LGBM NE NC	R	2.539	712.001	898639.425	0.035	0.952	947.966
	NR	1.436	405.194	277965.596	0.019	0.985	527.224
LGBM NE WC	R	2.646	733.76	944908.22	**0.036**	0.949	972.064
	NR	1.319	371.454	224922.193	0.017	0.988	474.26
LGBM WE NC	R	2.539	712.001	898639.425	0.035	0.952	947.966
	NR	1.436	405.194	277965.596	0.019	0.985	527.224
LGBM WE WC	R	**2.494**	**691.271**	**893631.363**	**0.036**	0.952	**945.321**
	NR	**1.192**	**338.852**	**188418.129**	**0.015**	**0.99**	**434.072**

lowest values in numerous metrics once again. This indicates a consistent robustness in this configuration, which seems to hold even when the model undergoes retraining, possibly suggesting a generally more reliable setup.

7.1.2.3.3 Conclusion In conclusion, the data strongly suggests that the LightGBM model showcases optimal performance when both encoders and covariates are utilized, unaffected by the retraining process. Moreover, the identical results observed in the LGBM WE NC and LGBM NE NC configurations indicate that the model does not differentiate based on the use of future encoders in these cases, leading to uniform outputs.

7.1.2.4 Summary of Machine Learning results It's worth noting that the retraining process didn't benefit any machine learning model. That could be because the retraining was done very frequently (every 2 hours) or because the training length was relatively small (2 weeks).

7.1.3 Deep Learning Models

In this subsection, we extend our analysis to encompass deep learning models, which offer a different approach to forecasting compared to traditional machine learning models. The detailed analysis of these models is presented next, however, graphical representations can be found in the appendices:

- N-BEATS Predictions: Appendix section .9
- NHITS Predictions: Appendix section .8

7.1.3.1 Analysis of NBEATS Model Configurations This segment elucidates the performance of the NBEATS model across various configurations. Similar to previous evaluations, the model has been assessed based on different setups, which include or exclude encoders and covariates. The subsequent analysis, derived from the data presented in the table, seeks to offer a concise interpretation of the model's performance across different metrics such as MAPE, MAE, MSE, RMSLE, R^2 , and RMSE. To facilitate a visual analysis, refer to the appendix section .9 for graphical illustrations of the model's performance across, including an in-depth analysis of a one-month segment.

7.1.3.1.1 Analysis of Results The table delineates the variations in metric values across various NBEATS configurations, paving the way for a nuanced understanding of the model's efficacy under different circumstances. The Melrose and Alto color codings pinpoint the configurations that have outshone in the non-retrain and retrain setups, respectively. Moreover, the bolded figures emphasize the configurations exhibiting supreme performance in each metric category.

Table 13

Analyzing NBEATS Model Configurations: *The table scrutinizes metric variations across NBEATS configurations, considering the inclusion or exclusion of encoders (E) and covariates (C).* ***Color and Formatting Guide:*** *Cells in Melrose and Alto spotlight the best performers for non-retrain and retrain setups, respectively. Bolded values indicate the superior performance in each metric category.*

		MAPE	**MAE**	**MSE**	**RMSLE**	**R2**	**RMSE**
NBEATS NE NC	R	**3.692**	**1039.86**	**1969413.6**	**0.049**	0.894	**1403.358**
	NR	1.715	501.24	336508.8	0.02	**0.982**	580.094
NBEATS NE WC	R	4.292	1209.311	2555396.0	0.056	0.863	1598.561
	NR	**0.893**	**256.742**	**107181.22**	**0.011**	0.994	**327.385**
NBEATS WE NC	R	4.586	1296.343	2987447.2	0.06	0.84	1728.423
	NR	1.657	472.278	342157.1	0.02	**0.982**	584.942
NBEATS WE WC	R	4.495	1270.429	2872229.0	0.059	0.846	1694.765
	NR	1.287	374.223	219082.95	0.016	0.988	468.063

7.1.3.1.2 Non-retrain Model Analysis In the non-retrain scenario, the NBEATS NE WC NR configuration emerges as the frontrunner, showcasing the best values across all metrics. These values, which indicate higher accuracy and reliability, suggest that the setup - which incorporates both encoders and covariates without retraining - is adept at identifying the intrinsic patterns in the data.

7.1.3.1.3 Retrain Model Analysis Contrastingly, when the focus shifts to the retrained models, the NBEATS NE NC R configuration performs the best. As with the non-retrain scenario, this setup not only reports the best values across the metrics,for this strategy.

7.1.3.1.4 Conclusion To wrap up, the NBEATS model tends to deliver a superior performance when both encoders and covariates are integrated, regardless of whether retraining is applied or not. This trend might be a reflection of the model's capability to curb overfitting effectively while proficiently detecting the underlying patterns in the data set. It's worth noting that the retraining approach seems to have a marginal impact on diminishing the model's performance. Furthermore, it performs better than all machine learning algorithms, which is a testament to its efficacy in forecasting electricity loads.

7.1.3.2 Detailed Analysis of NHITS Model Configurations

This section delineates a thorough analysis of the NHITS model configurations, emphasizing the influence of encoders (E) and covariates (C) on the model's performance. The marked cells in Melrose and Alto colors spotlight optimal performance in non-retrain and retrain setups, respectively, while bolded figures pinpoint peak performance in each metric category. Further visual representations can be found in appendix section .8.

Table 14

Analyzing NHITS Model Configurations: *This table scrutinizes metric variations across NHITS configurations, considering the inclusion or exclusion of encoders (E) and covariates (C).* ***Color and Formatting Guide:*** *Cells in Melrose and Alto highlight the best performers for non-retrain and retrain setups, respectively. Bolded values pinpoint the optimal performance in each metric category.*

		MAPE	**MAE**	**MSE**	**RMSLE**	**R2**	**RMSE**
NHITS NE NC	R	4.086	1166.386	2314193.2	0.053	0.876	1521.247
	NR	0.759	220.120	84015.6	**0.010**	0.995	289.854
NHITS NE WC	R	4.061	1154.943	2282733.8	0.053	0.878	1510.872
	NR	1.098	306.576	148839.7	0.014	0.992	385.798
NHITS WE NC	R	**3.947**	1121.042	**2079310.6**	**0.051**	**0.889**	**1441.981**
	NR	**0.750**	**214.853**	**81541.3**	0.010	**0.996**	**285.554**
NHITS WE WC	R	3.966	**1114.452**	2252650.8	0.053	0.879	1500.883
	NR	1.603	447.193	289075.3	0.019	0.985	537.657

7.1.3.2.1 Metric Analysis

The table illustrates marked variations in performance metrics across different NHITS model configurations. The color-coded cells guide the reader to the configurations with superior performance in non-retrain and retrain categories, and bold text highlights the top performance in individual metrics, simplifying the evaluation process.

7.1.3.2.2 Non-retrain Configurations Analysis In the non-retrain configurations, the NHITS WE NC NR setup emerges as highly efficient, exhibiting the lowest values in critical metrics such as MAPE, MAE, and MSE. This configuration also demonstrates a really high R^2 value, indicating a strong correlation between the predicted and actual values, and a minimized RMSLE score, reflecting its proficiency in reducing the logarithmic error between predicted and observed values.

7.1.3.2.3 Retrain Configurations Analysis For retrain configurations, the NHITS WE NC R setup stands out, showcasing minimal values across most metrics, implying high predictive accuracy. This setup illustrates a notable performance, particularly in the reduced MSE values and higher R^2 coefficients, indicators of substantial predictive accuracy and a close correlation between the observed and predicted values.

7.1.3.2.4 Conclusion In summary, the analysis reveals that configurations integrating solely the encoders encoders and yields better performance, a trend evident in both retrain and non-retrain setups. The difference between the two setups is perceptible, with the non-retrain setup exhibiting a better performance.

Chapter 8
Conclusion

8.1 Summary of Findings

In this section, we delve into an analysis of the performance of the various forecasting models we analyzed in the previous sections. We compare the performance of each model based on their respective best-performing variants, considering the influence of retraining on the RMSE metric.

Table 15

Comparison of Best Model Variants Based on RMSE

Model	Best Variant (Retraining Status)	RMSE
NHITS	NHITS WE NC (NR)	285.554
N-BEATS	NBEATS NE WC (NR)	327.385
LightGBM	LGBM WE WC (NR)	434.072
Random Forest	RF NE WC (NR)	450.109
XGBoost	XGB WE WC (NR)	450.75
Classical Series	Auto ARIMA (R)	1511.005

As depicted in Table 15, the NHITS model with the "WE NC" variant and not retrained (NR) showcases the superior performance with the lowest RMSE of 285.554, indicating its high predictive accuracy. Here, "WE" stands for "With Encoders", signifying that the model incorporates encoder mechanisms to enhance the representation of the input data, and "NC" denotes "Not past covariates", indicating the model does not utilize past covariate information in the forecasting process. The high accuracy of this model hints at the effectiveness of utilizing encoders without relying on past covariates, possibly facilitating the learning of more generalized patterns in the data.

This is closely followed by the N-BEATS model, another deep learning model, with the ”NE WC” variant (NR), which exhibits an RMSE of 327.385. The notable performance of this variant suggests that the inclusion of past covariate information can compensate for the absence of encoders, still enabling the model to achieve a high level of accuracy in predictions.

On the machine learning front, the LightGBM model, with the ”WE WC” variant (NR), outperforms both the Random Forest and XGBoost models, posting an RMSE of 434.072. This variant, which incorporates both encoders and past covariate data, demonstrates the synergistic benefits of combining these two features, potentially offering a rich representation of the input data and thus facilitating more accurate predictions.

In contrast, the Classical Series model employing the Auto ARIMA variant and retrained (R) demonstrates a significantly higher RMSE value of 1511.005, indicative of a larger deviation from the actual values in its predictions. This suggests that statistical methods, although foundational in time series analysis, may not hold the same predictive prowess as modern machine learning and deep learning approaches in certain contexts.

In summation, deep learning models emerge as the front-runners in predictive accuracy, followed by machine learning models, and then statistical models. The analysis also sheds light on the nuanced influences of encoder mechanisms and past covariate data on model performance, with different combinations of these features yielding varying levels of accuracy. It is evident that the LightGBM models hold a slight advantage over the Random Forest and XGBoost models, with the latter two showcasing very close performance metrics.

Interestingly, a noteworthy observation from the analysis is the generally superior performance exhibited by models that were not retrained, as opposed to their retrained counterparts. This counterintuitive phenomenon warrants a deeper exploration to uncover potential underlying factors.

One plausible explanation could be the introduction of noise during the retraining phase. If the additional data incorporated during retraining contains anomalies or patterns not present in the initial dataset, it might inadvertently increase the noise level in the model, thereby hampering its ability to generalize well to unseen data

Secondly, it is conceivable that the initial training phase already encapsulated a representative subset of the data, capturing the essential underlying patterns and trends proficiently. In such scenarios, further retraining may not necessarily enhance the model's predictive capacity, and might even degrade performance by fitting to the noise or inconsistencies present in the new data.

Lastly, it is possible that the retraining process may not have been optimized to its full potential, resulting in suboptimal performance. Possibly increasing the training length could have help the model to learn more effectively, thereby improving its predictive accuracy. Also, from the plots, it's noticeable that the retrained model predicts the data with a lag, overestimating the values.

References

[1] R. Weron, "Electricity price forecasting: A review of the state-of-the-art with a look into the future," *International Journal of Forecasting*, vol. 30, no. 4, pp. 1030–1081, Oct. 1, 2014, ISSN: 0169-2070. DOI: 10.1016/j.ijforecast.2014.08.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207014001083 (visited on 08/24/2023).

[2] K. Metaxiotis, A. Kagiannas, D. Askounis, and J. Psarras, "Artificial intelligence in short term electric load forecasting: A state-of-the-art survey for the researcher," *Energy Conversion and Management*, vol. 44, no. 9, pp. 1525–1534, Jun. 1, 2003, ISSN: 0196-8904. DOI: 10.1016/S0196-8904(02)00148-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0196890402001486 (visited on 08/24/2023).

[3] T. Hong and S. Fan, "Probabilistic electric load forecasting: A tutorial review," *International Journal of Forecasting*, vol. 32, no. 3, pp. 914–938, Jul. 1, 2016, ISSN: 0169-2070. DOI: 10.1016/j.ijforecast.2015.11.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207015001508 (visited on 08/24/2023).

[4] R. H. Inman, H. T. Pedro, and C. F. Coimbra, "Solar forecasting methods for renewable energy integration," *Progress in Energy and Combustion Science*, vol. 39, no. 6, pp. 535–576, 2013, ISSN: 0360-1285. DOI: 10.1016/j.pecs.2013.06.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360128513000294.

[5] T. P. Hughes, *Networks of Power*. Johns Hopkins University Press, 1983, ISBN: 978-0-8018-4614-4 978-0-8018-2873-7. DOI: 10.56021/9780801828737. [Online]. Available: https://www.press.jhu.edu/books/title/2031/networks-power (visited on 08/24/2023).

[6] J. Jonnes, *Empires of Light: Edison, Tesla, Westinghouse, and the Race to Electrify the World*. New York: Random House, 2003, 416 pp., ISBN: 978-0-375-50739-7.

[7] G. S. Smith, "Joseph Henry's role in the discovery of electromagnetic induction," *European Journal of Physics*, vol. 38, no. 1, p. 015 207, Dec. 2016, ISSN: 0143-0807. DOI: 10.1088/0143-0807/38/1/015207. [Online]. Available: https://dx.doi.org/10.1088/0143-0807/38/1/015207 (visited on 08/24/2023).

[8] M. Pollitt, "Electricity Reform in Chile. Lessons for Developing Countries," *Journal of Network Industries*, vol. os-5, no. 3-4, pp. 221–262, Sep. 1, 2004,

ISSN: 1389-9597. DOI: 10.1177/178359170400500301. [Online]. Available: https://doi.org/10.1177/178359170400500301 (visited on 08/18/2023).

[9] R. J. Green and D. M. Newbery, “Competition in the British Electricity Spot Market,” *Journal of Political Economy*, vol. 100, no. 5, pp. 929–953, 1992, ISSN: 0022-3808. JSTOR: 2138629. [Online]. Available: https://www.jstor.org/stable/2138629 (visited on 08/18/2023).

[10] O. Gjerde, “The deregulated Nordic electricity market-10 years of experience,” in *IEEE/PES Transmission and Distribution Conference and Exhibition*, vol. 2, Oct. 2002, 1473–1478 vol.2. DOI: 10.1109/TDC.2002.1177699.

[11] Y. Chang, “The New Electricity Market of Singapore: Regulatory framework, market power and competition,” *Energy Policy*, vol. 35, no. 1, pp. 403–412, Jan. 1, 2007, ISSN: 0301-4215. DOI: 10.1016/j.enpol.2005.11.036. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0301421505003344 (visited on 08/18/2023).

[12] H. Verdejo-Fredes *et al.*, “Retail Electricity Market Liberalization: An Overview of International Experience and Effects on the Chilean Regulated Tariff,” *Sustainability*, vol. 14, no. 21, p. 13 996, Oct. 27, 2022, ISSN: 2071-1050. DOI: 10.3390/su142113996. [Online]. Available: https://www.mdpi.com/2071-1050/14/21/13996 (visited on 08/18/2023).

[13] A. Ghezelbash, V. Khaligh, S. H. Fahimifard, and J. J. Liu, “A Comparative Perspective of the Effects of CO2 and Non-CO2 Greenhouse Gas Emissions on Global Solar, Wind, and Geothermal Energy Investment,” *Energies*, vol. 16, no. 7, p. 3025, Mar. 26, 2023, ISSN: 1996-1073. DOI: 10.3390/en16073025. [Online]. Available: https://www.mdpi.com/1996-1073/16/7/3025 (visited on 08/18/2023).

[14] P. Ponce and S. A. R. Khan, “A causal link between renewable energy, energy efficiency, property rights, and CO2 emissions in developed countries: A road map for environmental sustainability,” *Environmental Science and Pollution Research*, vol. 28, no. 28, pp. 37 804–37 817, Jul. 2021, ISSN: 0944-1344, 1614-7499. DOI: 10.1007/s11356-021-12465-0. [Online]. Available: https://link.springer.com/10.1007/s11356-021-12465-0 (visited on 08/18/2023).

[15] U. Uzar, “Political economy of renewable energy: Does institutional quality make a difference in renewable energy consumption?” *Renewable Energy*, vol. 155, pp. 591–603, Aug. 2020, ISSN: 09601481. DOI: 10.1016/j.renene.2020.03.172. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S096014812030505X (visited on 08/18/2023).

[16] A. Omri and D. K. Nguyen, “On the determinants of renewable energy consumption: International evidence,” *Energy*, vol. 72, pp. 554–560, Aug. 2014, ISSN: 03605442. DOI: 10 . 1016 / j . energy . 2014 . 05 . 081. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0360544214006483 (visited on 08/18/2023).

[17] P. Menanteau, D. Finon, and M.-L. Lamy, “Prices versus quantities: Choosing policies for promoting the development of renewable energy,” *Energy Policy*, vol. 31, no. 8, pp. 799–812, Jun. 1, 2003, ISSN: 0301-4215. DOI: 10.1016/S0301-4215(02)00133-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0301421502001337 (visited on 08/24/2023).

[18] P. E. Bett and H. E. Thornton, “The climatological relationships between wind and solar energy supply in Britain,” *Renewable Energy*, vol. 87, pp. 96–110, Mar. 2016, ISSN: 09601481. DOI: 10.1016/j.renene.2015.10.006. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0960148115303591 (visited on 08/18/2023).

[19] A. Vaderobli, D. Parikh, and U. Diwekar, “Optimization under Uncertainty to Reduce the Cost of Energy for Parabolic Trough Solar Power Plants for Different Weather Conditions,” *Energies*, vol. 13, no. 12, p. 3131, Jun. 17, 2020, ISSN: 1996-1073. DOI: 10.3390/en13123131. [Online]. Available: https://www.mdpi.com/1996-1073/13/12/3131 (visited on 08/18/2023).

[20] R. J. Hyndman and G. Athanasopoulos, “Forecasting: Principles and Practice,” 2018. [Online]. Available: https://otexts.com/fpp3/ (visited on 08/24/2023).

[21] E. S. Gardner Jr., “Exponential smoothing: The state of the art,” *Journal of Forecasting*, vol. 4, no. 1, pp. 1–28, 1985, ISSN: 1099-131X. DOI: 10.1002/for.3980040103. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/for.3980040103 (visited on 08/24/2023).

[22] G. Box, G. Jenkins, G. Reinsel, and G. Ljung, *Time Series Analysis: Forecasting and Control*, 4th ed. Wiley, 2011.

[23] N. I. Sapankevych and R. Sankar, “Time Series Prediction Using Support Vector Machines: A Survey,” *IEEE Computational Intelligence Magazine*, vol. 4, no. 2, pp. 24–38, May 2009, ISSN: 1556-6048. DOI: 10.1109/MCI.2009.932254.

[24] P.-H. Kuo and C.-J. Huang, “A High Precision Artificial Neural Networks Model for Short-Term Energy Load Forecasting,” *Energies*, vol. 11, no. 1, p. 213, Jan. 16, 2018, ISSN: 1996-1073. DOI: 10.3390/en11010213. [Online]. Available: http://www.mdpi.com/1996-1073/11/1/213 (visited on 08/24/2023).

[25] K. Amarasinghe, D. L. Marino, and M. Manic, “Deep neural networks for energy load forecasting,” *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, pp. 1483–1488, Jun. 2017. DOI: 10.1109/ISIE.2017.8001465. [Online]. Available: http://ieeexplore.ieee.org/document/8001465/ (visited on 08/24/2023).

[26] D. L. Marino, K. Amarasinghe, and M. Manic, “Building energy load forecasting using Deep Neural Networks,” *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, pp. 7046–7051, Oct. 2016. DOI: 10.1109/IECON.2016.7793413. [Online]. Available: http://ieeexplore.ieee.org/document/7793413/ (visited on 08/24/2023).

[27] S. Fan and L. Chen, “Short-Term Load Forecasting Based on an Adaptive Hybrid Method,” *IEEE Transactions on Power Systems*, vol. 21, no. 1, pp. 392–401, Feb. 2006, ISSN: 0885-8950. DOI: 10.1109/TPWRS.2005.860944. [Online]. Available: http://ieeexplore.ieee.org/document/1583738/ (visited on 08/24/2023).

[28] M. A. Hammad, B. Jereb, B. Rosi, and D. Dragan, “Methods and Models for Electric Load Forecasting: A Comprehensive Review,” *Logistics & Sustainable Transport*, vol. 11, no. 1, pp. 51–76, Feb. 1, 2020, ISSN: 2232-4968. DOI: 10.2478/jlst-2020-0004. [Online]. Available: https://www.sciendo.com/article/10.2478/jlst-2020-0004 (visited on 08/24/2023).

[29] J.-M. Wang and L.-P. Wang, “A new method for short-term electricity load forecasting,” *Transactions of the Institute of Measurement and Control*, vol. 30, no. 3-4, pp. 331–344, Aug. 2008, ISSN: 0142-3312, 1477-0369. DOI: 10.1177/0142331208090626. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0142331208090626 (visited on 08/24/2023).

[30] M. Alhussein, K. Aurangzeb, and S. I. Haider, “Hybrid CNN-LSTM Model for Short-Term Individual Household Load Forecasting,” *IEEE Access*, vol. 8, pp. 180 544–180 557, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3028281. [Online]. Available: https://ieeexplore.ieee.org/document/9210478/ (visited on 08/24/2023).

[31] Wei, Wang, Ni, and Tang, “Research and Application of a Novel Hybrid Model Based on a Deep Neural Network Combined with Fuzzy Time Series for Energy Forecasting,” *Energies*, vol. 12, no. 18, p. 3588, Sep. 19, 2019, ISSN: 1996-1073. DOI: 10.3390/en12183588. [Online]. Available: https://www.mdpi.com/1996-1073/12/18/3588 (visited on 08/24/2023).

[32] G. Zhang, “Time series forecasting using a hybrid ARIMA and neural network model,” *Neurocomputing*, vol. 50, pp. 159–175, Jan. 2003, ISSN: 09252312. DOI:

10.1016/S0925-2312(01)00702-0. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0925231201007020 (visited on 08/24/2023).

[33] M. Khashei and M. Bijari, "A novel hybridization of artificial neural networks and ARIMA models for time series forecasting," *Applied Soft Computing*, vol. 11, no. 2, pp. 2664–2675, Mar. 2011, ISSN: 15684946. DOI: 10.1016/j.asoc.2010.10.015. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1568494610002759 (visited on 08/24/2023).

[34] "Time Series Analysis: Forecasting and Control, 5th Edition — Wiley," Wiley.com. (), [Online]. Available: https://www.wiley.com/en-us/Time+Series+Analysis%3A+Forecasting+and+Control%2C+5th+Edition-p-9781118675021 (visited on 08/24/2023).

[35] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, Dec. 1974, ISSN: 1558-2523. DOI: 10.1109/TAC.1974.1100705.

[36] "Forecasting Sales by Exponentially Weighted Moving Averages — Management Science." (), [Online]. Available: https://pubsonline.informs.org/doi/10.1287/mnsc.6.3.324 (visited on 08/24/2023).

[37] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," *International Journal of Forecasting*, vol. 20, no. 1, pp. 5–10, Jan. 1, 2004, ISSN: 0169-2070. DOI: 10.1016/j.ijforecast.2003.09.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207003001134 (visited on 08/24/2023).

[38] J. W. Taylor, "Exponential smoothing with a damped multiplicative trend," *International Journal of Forecasting*, vol. 19, no. 4, pp. 715–725, Oct. 2003, ISSN: 01692070. DOI: 10.1016/S0169-2070(03)00003-7. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0169207003000037 (visited on 08/24/2023).

[39] "Rob J Hyndman - Forecasting with Exponential Smoothing: The State Space Approach," Rob J Hyndman. (), [Online]. Available: https://robjhyndman.com/expsmooth/ (visited on 08/24/2023).

[40] C. Chatfield, A. B. Koehler, J. K. Ord, and R. D. Snyder, "A New Look at Models For Exponential Smoothing," *Journal of the Royal Statistical Society: Series D (The Statistician)*, vol. 50, no. 2, pp. 147–159, 2001, ISSN: 1467-9884. DOI: 10.1111/1467-9884.00267. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-9884.00267 (visited on 08/24/2023).

[41] E. S. Gardner Jr., "Exponential smoothing: The state of the art," *Journal of Forecasting*, vol. 4, no. 1, pp. 1–28, 1985, ISSN: 1099-131X. DOI: 10.1002/for.3980040103. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/for.3980040103 (visited on 08/24/2023).

[42] I. Svetunkov, N. Kourentzes, and J. K. Ord, "Complex exponential smoothing," *Naval Research Logistics (NRL)*, vol. 69, no. 8, pp. 1108–1123, 2022, ISSN: 1520-6750. DOI: 10.1002/nav.22074. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.22074 (visited on 08/24/2023).

[43] V. Assimakopoulos and K. Nikolopoulos, "The theta model: A decomposition approach to forecasting," *International Journal of Forecasting*, The M3- Competition, vol. 16, no. 4, pp. 521–530, Oct. 1, 2000, ISSN: 0169-2070. DOI: 10.1016/S0169-2070(00)00066-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207000000662 (visited on 08/24/2023).

[44] J. A. Fiorucci, T. R. Pellegrini, F. Louzada, F. Petropoulos, and A. B. Koehler, "Models for optimising the theta method and their relationship to state space models," *International Journal of Forecasting*, vol. 32, no. 4, pp. 1151–1161, Oct. 1, 2016, ISSN: 0169-2070. DOI: 10.1016/j.ijforecast.2016.02.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207016300243 (visited on 08/24/2023).

[45] A. E. Hoerl and R. W. Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970, ISSN: 0040-1706. DOI: 10.2307/1267351. JSTOR: 1267351. [Online]. Available: https://www.jstor.org/stable/1267351 (visited on 08/25/2023).

[46] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*, 2nd edition. New York, NY: Springer, Jan. 1, 2016, 767 pp., ISBN: 978-0-387-84857-0.

[47] J. Brownlee. "Bagging and Random Forest Ensemble Algorithms for Machine Learning," MachineLearningMastery.com. (Apr. 21, 2016), [Online]. Available: https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/ (visited on 08/25/2023).

[48] G. S. E. Marzban, *38 Random Forests — All Models Are Wrong: Concepts of Statistical Learning.* [Online]. Available: https://allmodelsarewrong.github.io (visited on 08/25/2023).

[49] Jeremybeauchamp, *English: A visual comparison between the complexity of decision trees and random forests.* Dec. 13, 2020. [Online]. Available: https://commons.wikimedia.org/wiki/File:Decision_Tree_vs._Random_Forest.png#metadata (visited on 08/25/2023).

[50] J. Riebesell. "Random Forest." (Apr. 9, 2022), [Online]. Available: https://tikz.net/random-forest/ (visited on 08/25/2023).

[51] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 13, 2016, pp. 785–794. DOI: 10.1145/2939672.2939785. arXiv: 1603.02754 [cs]. [Online]. Available: http://arxiv.org/abs/1603.02754 (visited on 08/25/2023).

[52] G. Ke *et al.*, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://papers.nips.cc/paper_files/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html (visited on 08/25/2023).

[53] Y. Shi, G. Ke, Z. Chen, S. Zheng, and T.-Y. Liu, "Quantized Training of Gradient Boosting Decision Trees," in *Advances in Neural Information Processing Systems*, vol. 35, Curran Associates, Inc., 2022, pp. 18 822–18 833. [Online]. Available: https://papers.nips.cc/paper_files/paper/2022/hash/77911ed9e6e864ca1a3d165b2c3cb258-Abstract.html (visited on 08/25/2023).

[54] "Learning long-term dependencies with gradient descent is difficult — IEEE Journals & Magazine — IEEE Xplore." (), [Online]. Available: https://ieeexplore.ieee.org/abstract/document/279181 (visited on 08/25/2023).

[55] A. Graves, "Long Short-Term Memory," in *Supervised Sequence Labelling with Recurrent Neural Networks*, ser. Studies in Computational Intelligence, A. Graves, Ed., Berlin, Heidelberg: Springer, 2012, pp. 37–45, ISBN: 978-3-642-24797-2. DOI: 10.1007/978-3-642-24797-2_4. [Online]. Available: https://doi.org/10.1007/978-3-642-24797-2_4 (visited on 08/25/2023).

[56] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 15, 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735 (visited on 08/25/2023).

[57] "Understanding LSTM Networks – colah's blog." (), [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 08/25/2023).

[58] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling." arXiv: 1412.3555 [cs]. (Dec. 11, 2014), [Online]. Available: http://arxiv.org/abs/1412.3555 (visited on 08/25/2023), preprint.

[59] A. dprogrammer. "RNN, LSTM & GRU," dProgrammer lopez. (Apr. 6, 2019), [Online]. Available: http://dprogrammer.org/rnn-lstm-gru (visited on 08/25/2023).

[60] J. Herzen *et al.*, "Darts: User-Friendly Modern Machine Learning for Time Series," *Journal of Machine Learning Research*, vol. 23, no. 124, pp. 1–6, 2022, ISSN: 1533-7928. [Online]. Available: http://jmlr.org/papers/v23/21-1177.html (visited on 08/25/2023).

[61] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio. "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting." arXiv: 1905.10437 `[cs, stat]`. (Feb. 20, 2020), [Online]. Available: http://arxiv.org/abs/1905.10437 (visited on 09/05/2023), preprint.

[62] C. Challu, K. G. Olivares, B. N. Oreshkin, F. Garza, M. Mergenthaler-Canseco, and A. Dubrawski. "N-HiTS: Neural Hierarchical Interpolation for Time Series Forecasting." arXiv: 2201.12886 `[cs]`. (Nov. 29, 2022), [Online]. Available: http://arxiv.org/abs/2201.12886 (visited on 09/05/2023), preprint.

[63] "Data Platform – Open Power System Data." (), [Online]. Available: https://data.open-power-system-data.org/time_series/2020-10-06 (visited on 08/27/2023).

[64] *Python-holidays*, vacanza, Aug. 21, 2023. [Online]. Available: https://github.com/vacanza/python-holidays (visited on 08/27/2023).

[65] Cristian Challú, Kin G. Garza, Max Canseco, *StatsForecast: Lightning fast forecasting with statistical and econometric models*, PyCon Salt Lake City, Utah, US 2022, 2022. [Online]. Available: https://github.com/Nixtla/statsforecast.

[66] T. pandas development team, *Pandas-dev/pandas: Pandas*, Zenodo, Aug. 30, 2023. DOI: 10.5281/zenodo.8301632. [Online]. Available: https://zenodo.org/record/8301632 (visited on 09/05/2023).

[67] J. Herzen *et al.*, "Darts: User-Friendly Modern Machine Learning for Time Series," *Journal of Machine Learning Research*, vol. 23, no. 124, pp. 1–6, 2022, ISSN: 1533-7928. [Online]. Available: http://jmlr.org/papers/v23/21-1177.html (visited on 09/04/2023).

[68] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011, ISSN: 1533-7928. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html (visited on 09/05/2023).

Appendices

.1 Descriptive Statistics

.1.0.1 Central Tendency

Central tendency measures provide insights into the central or typical value of a distribution. They summarize the data by identifying a single value that represents the entire dataset. The main measures of central tendency are:

1. **Mean**: The arithmetic mean, or simply the mean, is the sum of all values divided by the number of values. Mathematically:

   $$\mu = \frac{\sum_{i=1}^{n} x_i}{n}$$

   where x_i are the individual observations, and n is the total number of observations.

2. **Median**: The median is the middle value of a sorted dataset. If the dataset has an odd number of values, the median is the middle value. For an even number of values, it's the average of the two middle values.

3. **Mode**: The mode is the value that appears most frequently in the dataset. A distribution can have more than one mode, known as bimodal or multimodal, depending on the number of values that appear most frequently. The mode can be found using:

   $$\text{Mode} = \max_{x_i} (\text{frequency}(x_i))$$

   where frequency(x_i) is the count of occurrences of the value x_i in the dataset.

Understanding the central tendency helps in identifying the typical behavior in time series data such as electricity load patterns. For instance, the mean might represent the average daily electricity consumption, while the mode might indicate the most common hourly demand level.

.1.0.2 Dispersion Dispersion measures provide insights into how spread out the values are in a dataset. They reveal the variability and give a sense of how much the data deviates from the central value.

1. **Range**: The range is calculated as the difference between the maximum and minimum values.
$$\text{Range} = \max(x) - \min(x)$$

2. **Variance**: The variance measures the average of the squared differences from the mean.
$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}$$

3. **Standard Deviation**: The standard deviation gives a measure of how much individual data points deviate from the mean.
$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}}$$

4. **Interquartile Range (IQR)**: The IQR measures the statistical spread between the first quartile (Q1) and the third quartile (Q3).
$$\text{IQR} = Q_3 - Q_1$$

Dispersion measures are vital for the variability data. Understanding dispersion is beneficial for modeling, since it could help in tuning model parameters.

.1.0.3 Shape Shape measures help describe the distribution's form, including its symmetry and the nature of its tails.

1. **Skewness**: Skewness measures the asymmetry of a distribution. The mathe-

matical formula for sample skewness is:

$$\text{Skewness} = \frac{n\sqrt{n-1}}{(n-2)} \cdot \frac{\sum_{i=1}^{n}(x_i-\mu)^3}{\sigma^3}$$

2. **Kurtosis**: Kurtosis measures the "tailedness" of the distribution. The mathematical formula for sample kurtosis is:

$$\text{Kurtosis} = \frac{\sum_{i=1}^{n}(x_i-\mu)^4}{n\sigma^4}$$

Analyzing the skewness and kurtosis can guide the transformation or preprocessing steps required to make the data more suitable for modeling.

.1.0.4 Moving Average Plots Moving average plots are an essential tool for smoothing time series data and revealing underlying trends, patterns and even possible baseline predictions.

Simple Moving Average (SMA): The SMA calculates the average of the last k observations.

$$\text{SMA}(t) = \frac{\sum_{i=t-k+1}^{t} x_i}{k}$$

.1.0.5 Rolling Statistics Rolling statistics apply statistical measures over a rolling window of a fixed size across a time series. They're used to reveal local trends and to analyze volatility of a time series.

1. **Rolling Mean**: The rolling mean is the average of the observations within the window.

$$\text{Rolling Mean}(t) = \frac{\sum_{i=t-k+1}^{t} x_i}{k}$$

2. **Rolling Standard Deviation**: The rolling standard deviation measures the

local variability.

$$\text{Rolling Std Dev}(t) = \sqrt{\frac{\sum_{i=t-k+1}^{t}(x_i - \mu_t)^2}{k}}$$

Rolling statistics are crucial in electricity load forecasting for revealing local trends, analyzing volatility, and possibly enhancing modeling.

.1.1 Statistical Tests

Statistical tests play a vital role in validating assumptions about the data and in model selection.

.1.1.1 Durbin-Watson Statistic The Durbin-Watson statistic serves as a diagnostic tool used to detect the presence of autocorrelation in the residuals of a regression model. Autocorrelation in residuals can violate the OLS assumptions, potentially leading to biased or misleading results.

Mathematical Formula:

$$\text{Durbin-Watson} = \frac{\sum_{t=2}^{n}(e_t - e_{t-1})^2}{\sum_{t=1}^{n} e_t^2}$$

In this formula, e_t represents the residual at time t, and n is the number of observations.

Interpretation:

- **Value close to 2**: Indicates that there is no autocorrelation in the residuals, meaning that the model satisfies one of the key OLS assumptions.
- **Value close to 0**: Suggests positive autocorrelation, which could mean that the model is missing some explanatory variables related to time or other factors.

- **Value close to 4**: Implies negative autocorrelation, which is relatively rare but could indicate an overly complicated model.

The Durbin-Watson statistic is used to detect linear autocorrelation and only can work in equally spaced observations. For multiple lags or non linear relations, other tests like Ljung-Box test should be used.

.1.1.2 Augmented Dickey-Fuller Test The Augmented Dickey-Fuller (ADF) test is used to determine if a time series is stationary. It's application could guide transformations to achieve stationarity. Stationary data can be likened to a car traveling on a straight, level highway. While the car may occasionally need to pull over for brief stops or detours, it always returns to the main road and continues its straight journey.

Nonstationary data, on the other hand, is similar to a car trying to navigate a winding, unpredictable path. The car might occasionally veer off course due to various obstructions or distractions, making its path appear erratic and without a consistent direction. To an observer, it might seem as though the car is moving without a clear, linear trend.

In time series analysis, ensuring data stationarity is crucial as most forecasting models rely on the assumption that the statistical properties (like mean and variance) of the series are constant over time. Nonstationary data can lead to unreliable and spurious results. Thus, transforming nonstationary data into a stationary form often becomes a necessary preprocessing step in time series forecasting.

Mathematical Formula:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \delta_1 \Delta y_{t-1} + \ldots + \delta_p \Delta y_{t-p} + \varepsilon_t$$

Interpretation: In the unit root test, we aim to evaluate two scenarios: the null hypothesis $\gamma = 0$ and the alternative hypothesis $\gamma < 0$. To make this assessment, a test statistic, denoted as DF_τ, is calculated using the formula:

$$\mathrm{DF}_\tau = \frac{\hat{\gamma}}{\mathrm{SE}(\hat{\gamma})}$$

Here, $\hat{\gamma}$ is the estimated value of γ, and $\mathrm{SE}(\hat{\gamma})$ is the standard error associated with $\hat{\gamma}$.

Next, DF_τ is compared to a Dickey-Fuller specific critical value. Importantly, this test is one-sided and only negative values of DF_τ are considered. If DF_τ is more negative than the critical value, the null hypothesis ($\gamma = 0$) is rejected, leading to the conclusion that the data series does not contain a unit root.

- Fail to Reject Null Hypothesis ($\gamma = 0$): Non-stationary.
- Reject Null Hypothesis ($\gamma \neq 0$): Stationary.

.1.1.3 Johansen Test

The Johansen test serves as a statistical tool for determining the number of cointegrating relationships in a multivariate time series system. Essentially, cointegration refers to a long-term equilibrium relationship between multiple time series variables, even if the individual series themselves are non-stationary.

Mathematical Formula:

$$\Delta\mathbf{Y}_t = \mathbf{\Pi}\mathbf{Y}_{t-1} + \sum_{i=1}^{p-1}\mathbf{\Gamma}_i\Delta\mathbf{Y}_{t-i} + \mathbf{u}_t$$

Here, $\Delta\mathbf{Y}_t$ represents the change in the vector of observed variables at time t. $\mathbf{\Pi}$ is the long-run impact matrix, while $\mathbf{\Gamma}_i$ captures the short-run dynamics. $\mathbf{u}_t$ is the error term.

Interpretation:

- **Rank = 0**: This implies that there is no cointegration among the variables, meaning that any long-term equilibrium relationship is absent.

- **Rank = 1**: Indicates the presence of a single cointegrating relationship among the variables. In this case, all the variables are linked by one long-term equilibrium condition.
- **Rank** > 1: Suggests that multiple cointegrating relationships exist. This scenario is more complex, as it implies that there are several long-term equilibrium conditions that tie the variables together.

The rank of cointegration helps to clarify the underlying structure of the system, providing insights into the long-term adjustments that the variables make towards equilibrium. However, it's important to note that multiple cointegrating relatioship may require addiotional analysis to decipher any implications.

.2 Data Normalization

Data normalization, at its core, is an essential preprocessing step in various machine learning and deep learning applications. Its primary purpose is to adjust the scales of different features to a unified range or distribution, ensuring the algorithm's stable and efficient convergence. Mathematically, the normalization process can be represented as:

$$X_{\text{normalized}} = \frac{X - \mu}{\sigma}$$

where X is the original feature, μ is the mean, and σ is the standard deviation.

.2.0.1 Why is Normalization Important?

1. **Convergence Rate:** Most machine learning algorithms, especially those dependent on gradient-based optimization methods, are sensitive to the scale of input features. When features are on disparate scales, the gradients can either

vanish (become too small) or explode (become too large), leading to unstable training. Mathematically, this can be seen in the update rule of gradient descent:

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta)$$

Here, $\nabla J(\theta)$ is the gradient of the loss function with respect to the parameters θ. When features are not normalized, the magnitude and direction of this gradient can be largely influenced by the feature with the highest scale, leading to slow or unstable convergence.

2. **Avoiding Dominance:** In algorithms that compute distances or similarities, like k-means clustering or k-nearest neighbors, features with larger scales can dominate the objective function. This means the algorithm is more sensitive to changes in that particular feature, possibly leading to suboptimal models.

3. **Preserving Patterns:** In time series data, trends, cycles, and seasonality are crucial. Proper normalization ensures that these patterns are retained. For instance, a sudden spike in electricity demand due to an event will still be noticeable post-normalization, allowing models to capture such anomalies.

.3 Common Techniques

1. **Min-Max Scaling:** This technique scales the data based on the minimum and maximum values to a specific range, typically [0, 1]. Mathematically, this can be represented as:

$$X_{\text{scaled}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

2. **Z-Score or Standard Scaling:** Here, the data is scaled based on its mean and

standard deviation, resulting in a distribution with a mean of 0 and a standard deviation of 1. This is the formula we introduced at the beginning.

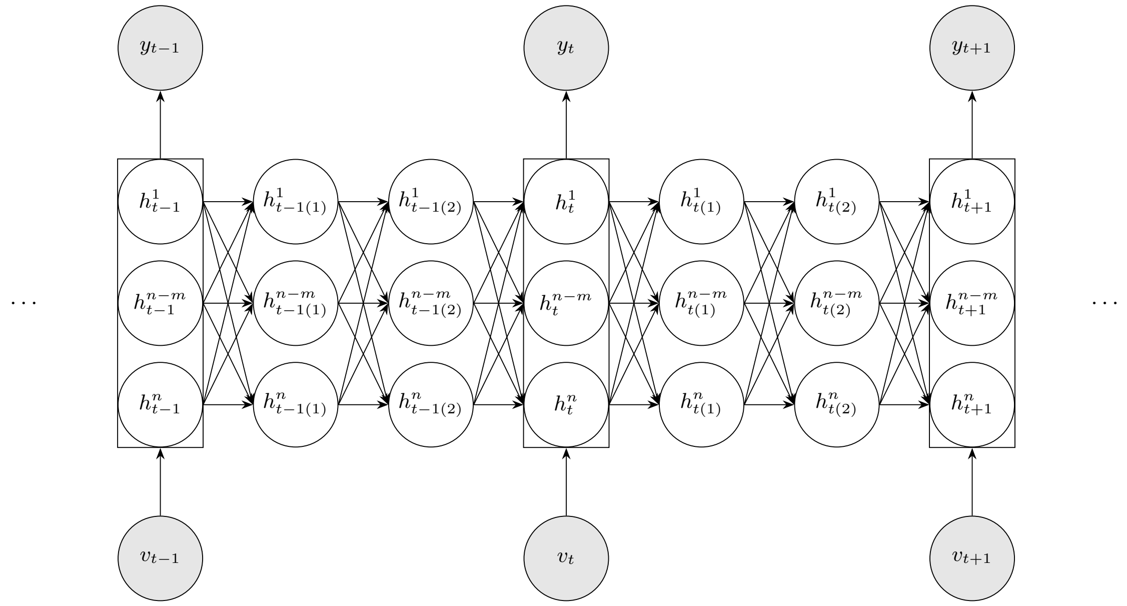
3. **Max Absolute Scaling:** Particularly useful for data that spans both positive and negative values, this scaler adjusts data such that the absolute maximum value corresponds to 1 (or another chosen value). Mathematically:

$$X_{\text{scaled}} = \frac{X}{\max(|X|)}$$

.4 Appendix B: Detailed RNN Architecture

Figure A1

Unfolded architecture of an RNN across various time steps.



The figure illustrates the architecture of an unfolded Recurrent Neural Network (RNN) across various time steps:

1. **Hidden States**: The nodes $h^{\text{index}}_{\text{time}}$ denote the hidden states at different time steps and layers.

2. **Inputs/Outputs**: The gray-filled nodes, y and v, represent outputs and inputs at specific time steps, respectively.

3. **Layers**: The RNN consists of multiple layers, as indicated by varying indices from 1 to n.

4. **Connections**:

   - Horizontal arrows signify memory transition between time steps.
   - Vertical arrows within a time step convey information flow between layers.

5. **Bounding Boxes**: These encapsulate hidden states at certain time steps, showcasing them as unified entities.

6. **Ellipses**: These on the edges suggest the sequence's continuity beyond the displayed time steps.

This visual representation underscores the recurrent nature of RNNs, emphasizing the sequential flow of information across layers and time steps.

.5 Random Forest plots

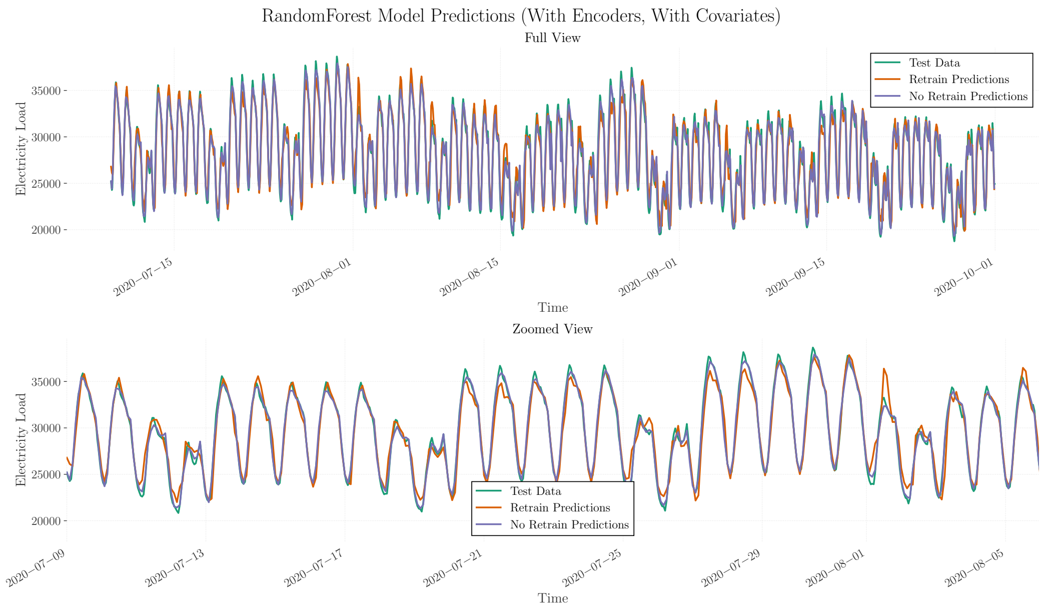


Figure A2

Random Forest model predictions with encoders and covariates - Full-scale and zoomed views

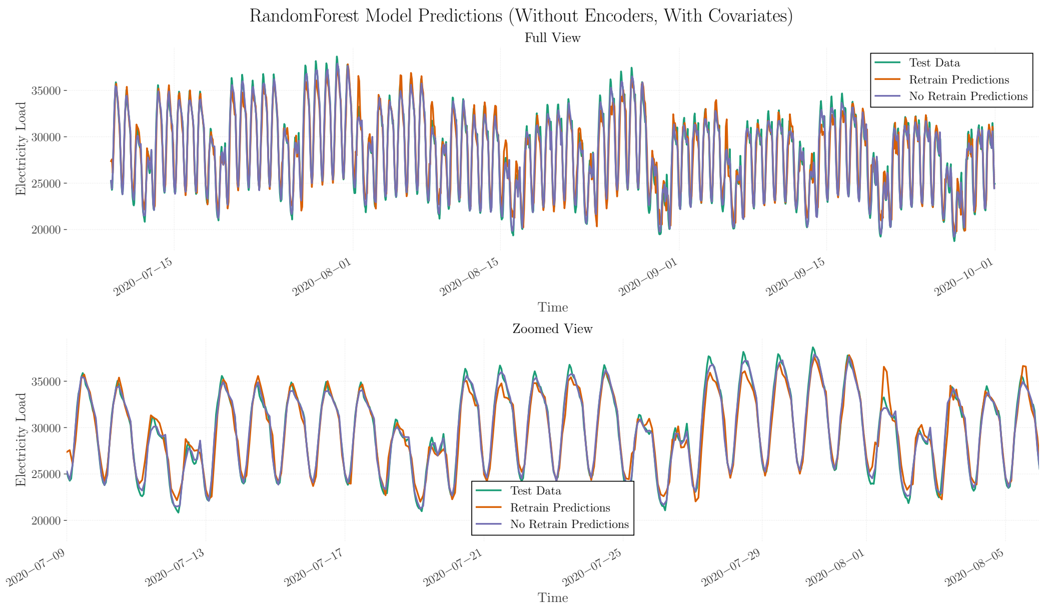


Figure A3

Random Forest model predictions without encoders but with covariates - Full-scale and zoomed views

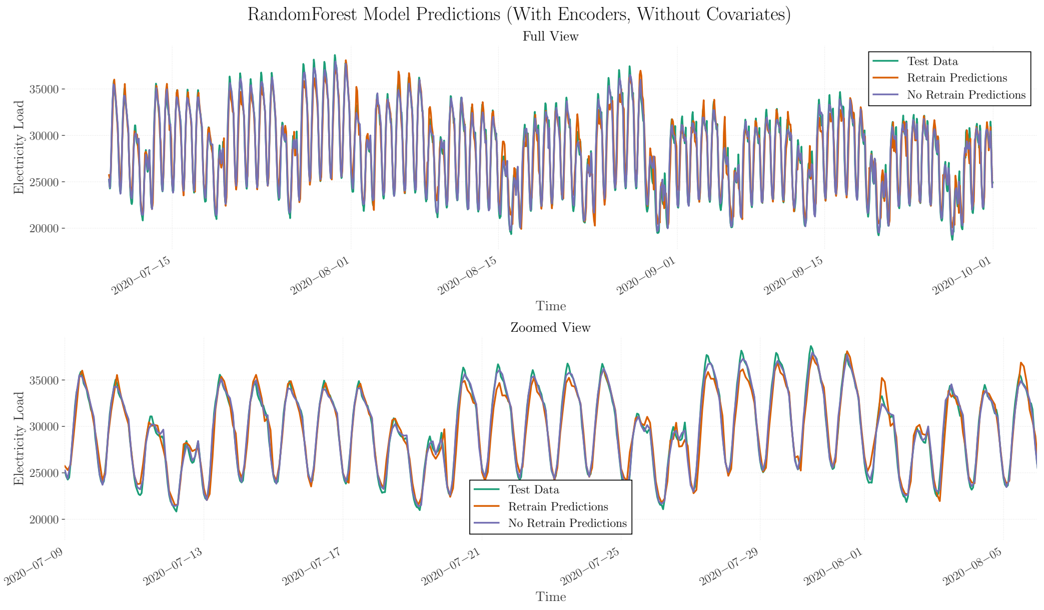


Figure A4

Random Forest model predictions with encoders but without covariates - Full-scale and zoomed views

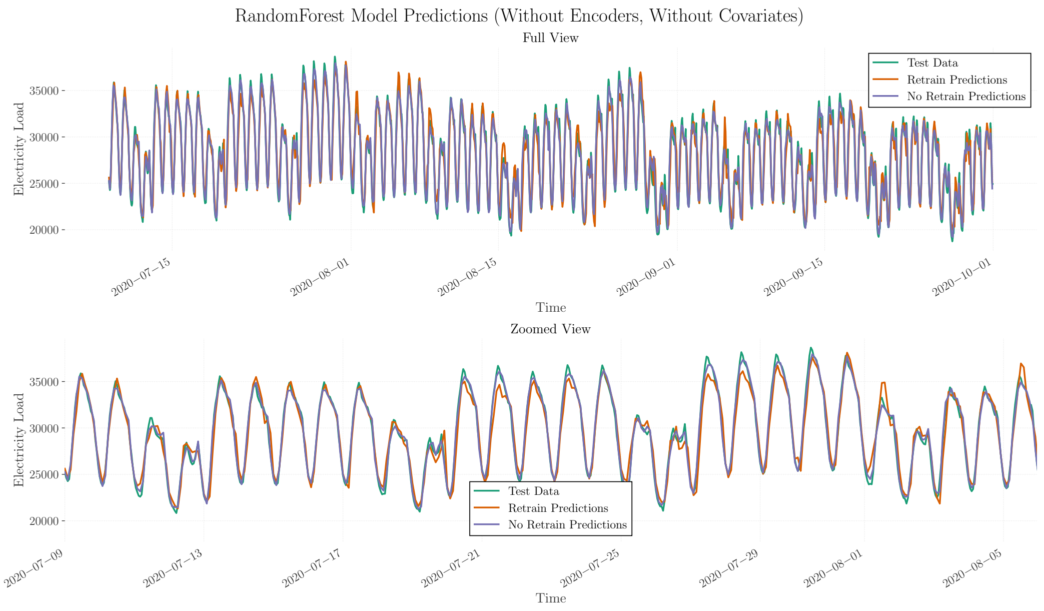


Figure A5

Random Forest model predictions without encoders and without covariates - Full-scale and zoomed views

.6 XGBoost plots

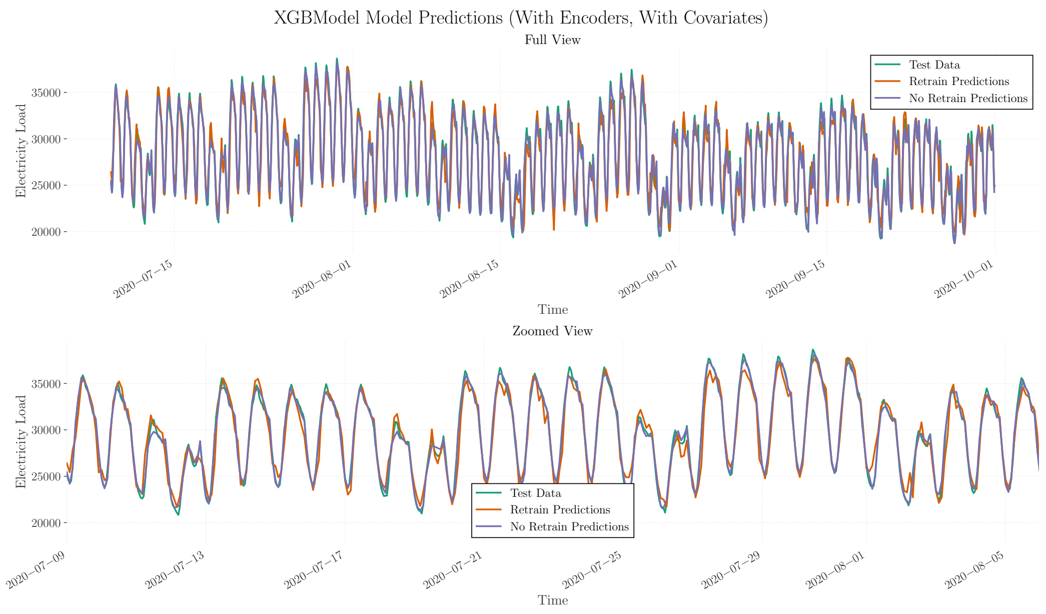


Figure A6

XGBoost model predictions with encoders and covariates - Full-scale and zoomed views

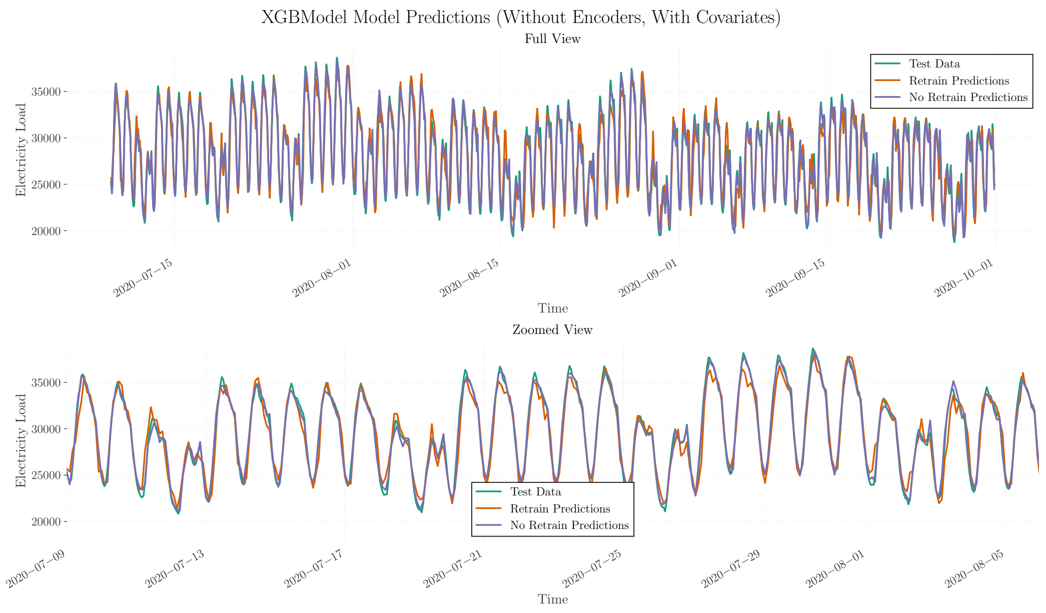


Figure A7

XGBoost model predictions without encoders but with covariates - Full-scale and zoomed views

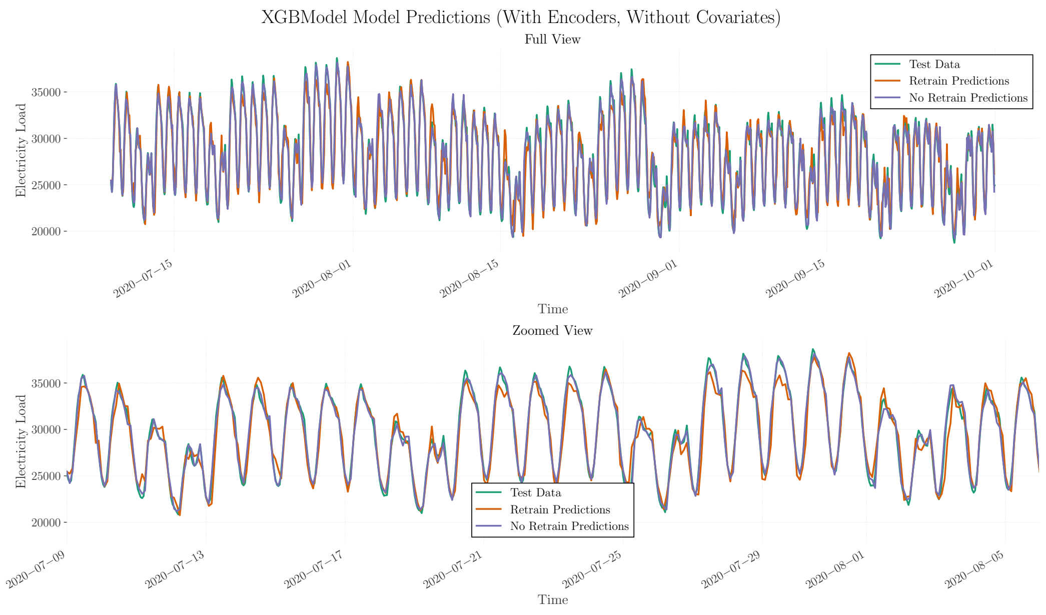


Figure A8

XGBoost model predictions with encoders but without covariates - Full-scale and zoomed views

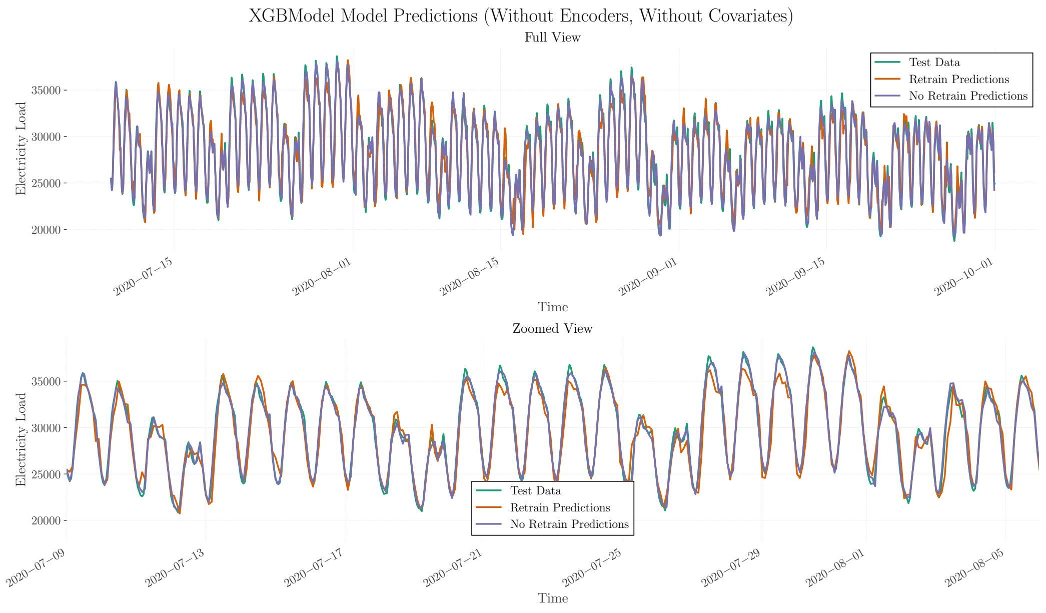


Figure A9

XGBoost model predictions without encoders and without covariates - Full-scale and zoomed views

.7 LightGBM plots

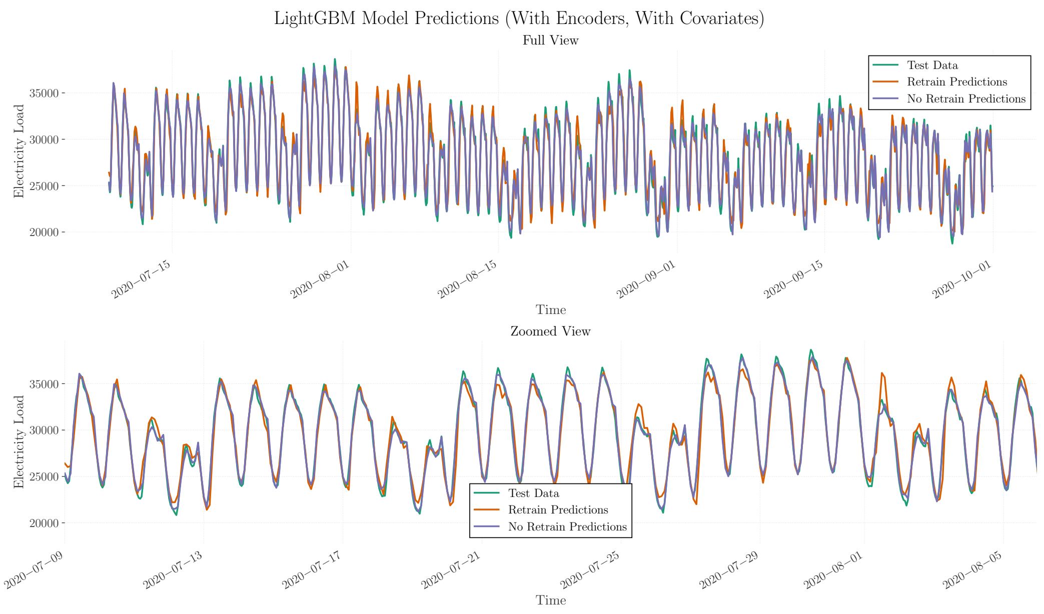


Figure A10

LightGBM model predictions with encoders and covariates - Full-scale and zoomed views

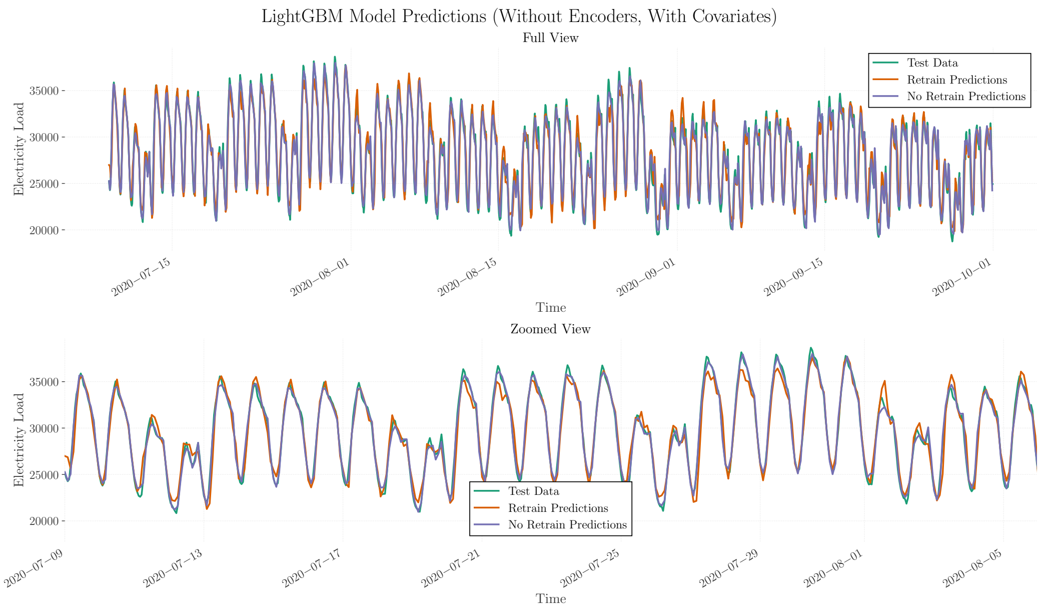


Figure A11

LightGBM model predictions without encoders but with covariates - Full-scale and zoomed views

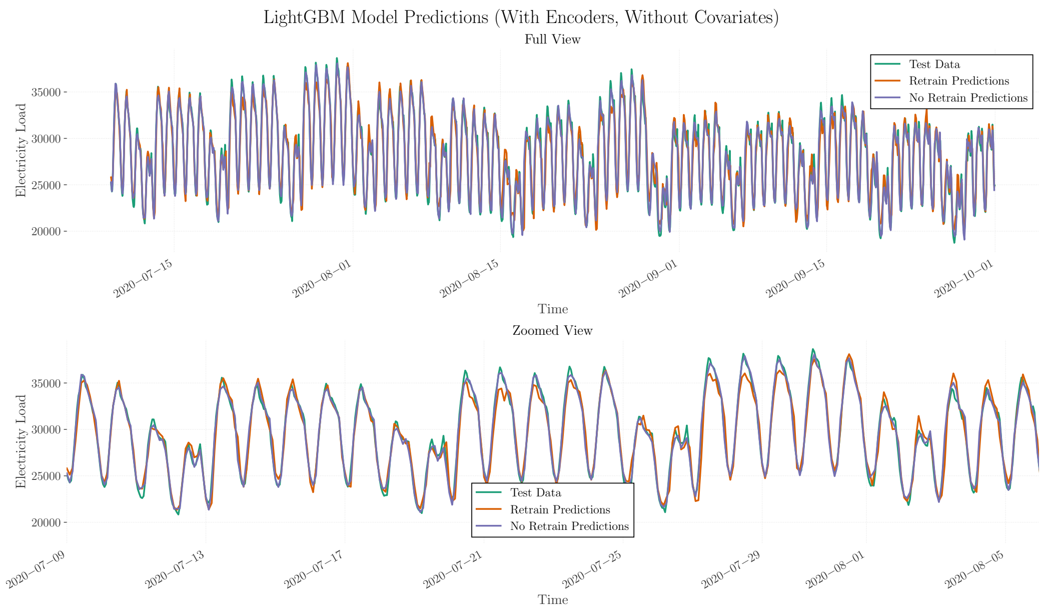


Figure A12

LightGBM model predictions with encoders but without covariates - Full-scale and zoomed views

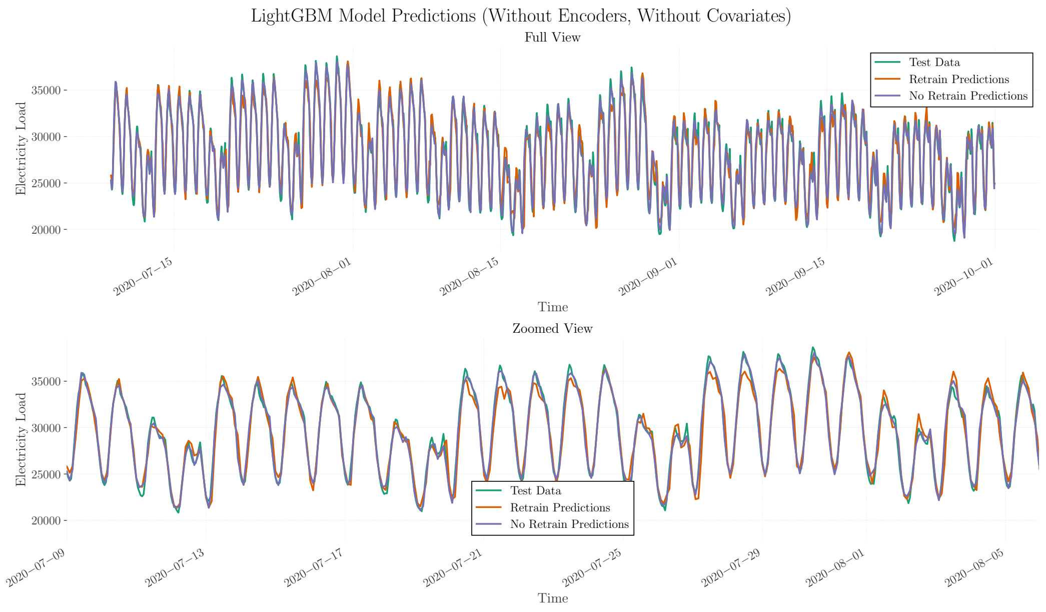


Figure A13

LightGBM model predictions without encoders and without covariates - Full-scale and zoomed views

.8 NHITS Model Predictions

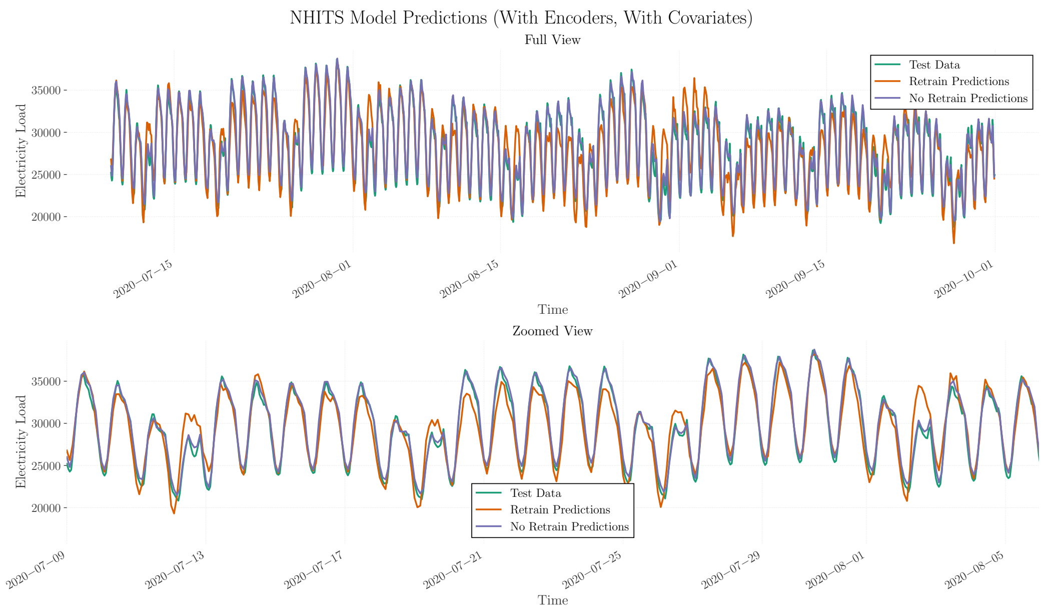


Figure A14

NHITS model predictions with encoders and covariates - Full-scale and zoomed views

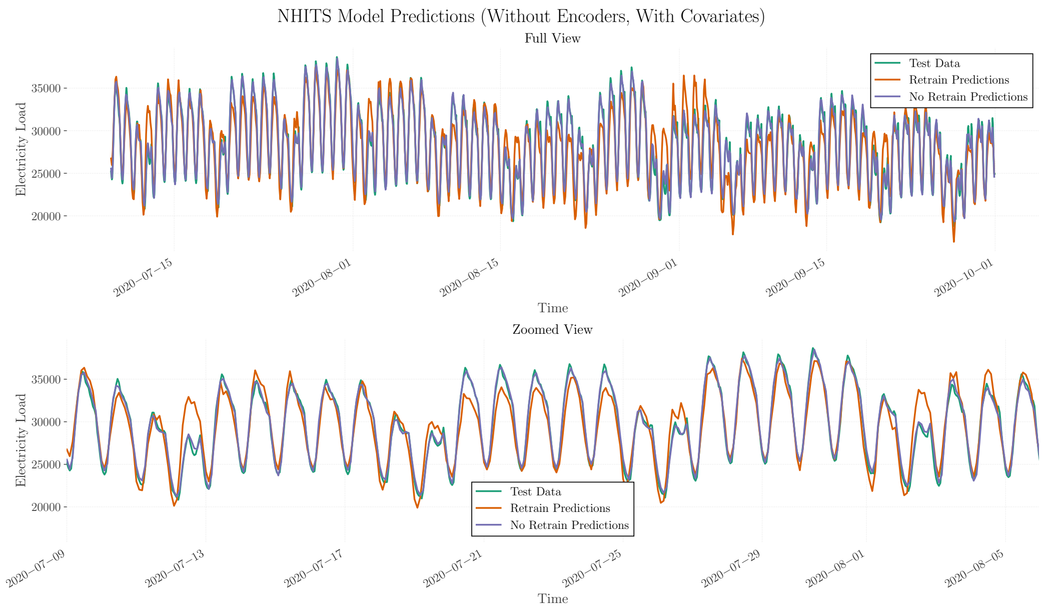


Figure A15

NHITS model predictions without encoders but with covariates - Full-scale and zoomed views

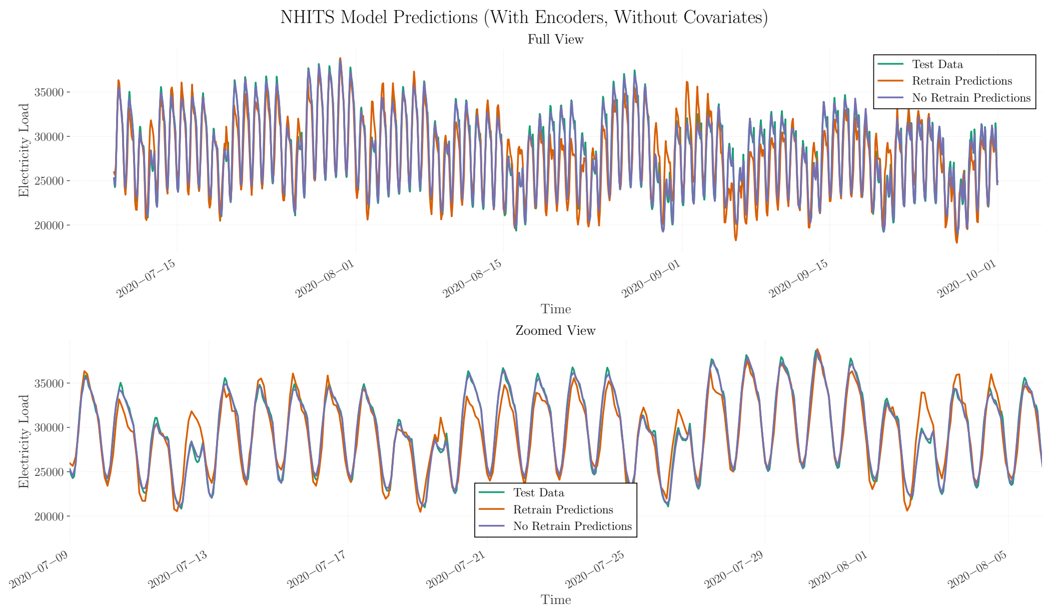


Figure A16

NHITS model predictions with encoders but without covariates - Full-scale and zoomed views

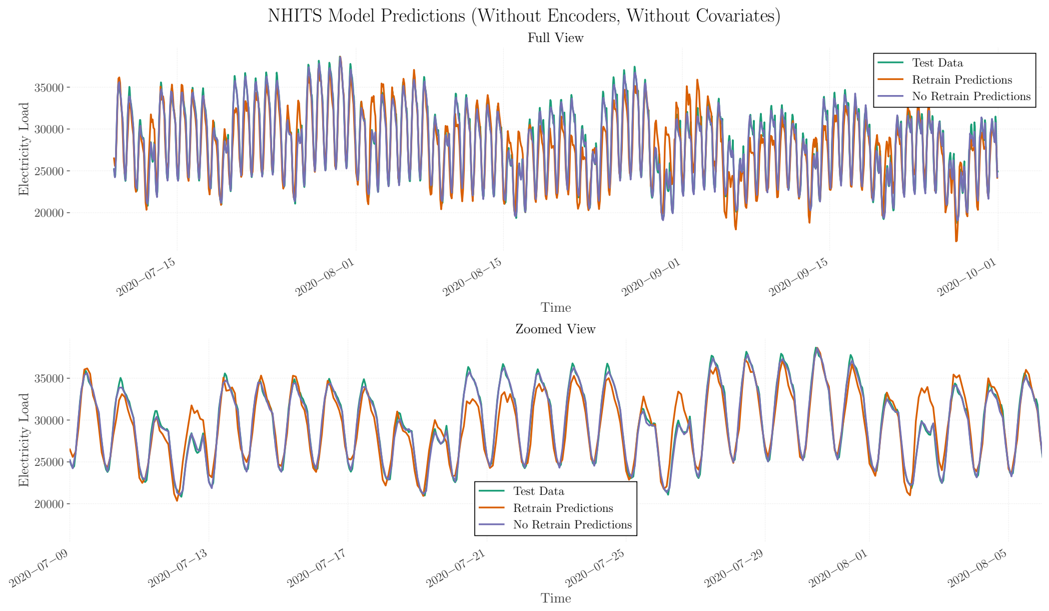


Figure A17

NHITS model predictions without encoders and without covariates - Full-scale and zoomed views

.9 NBEATS Model Predictions

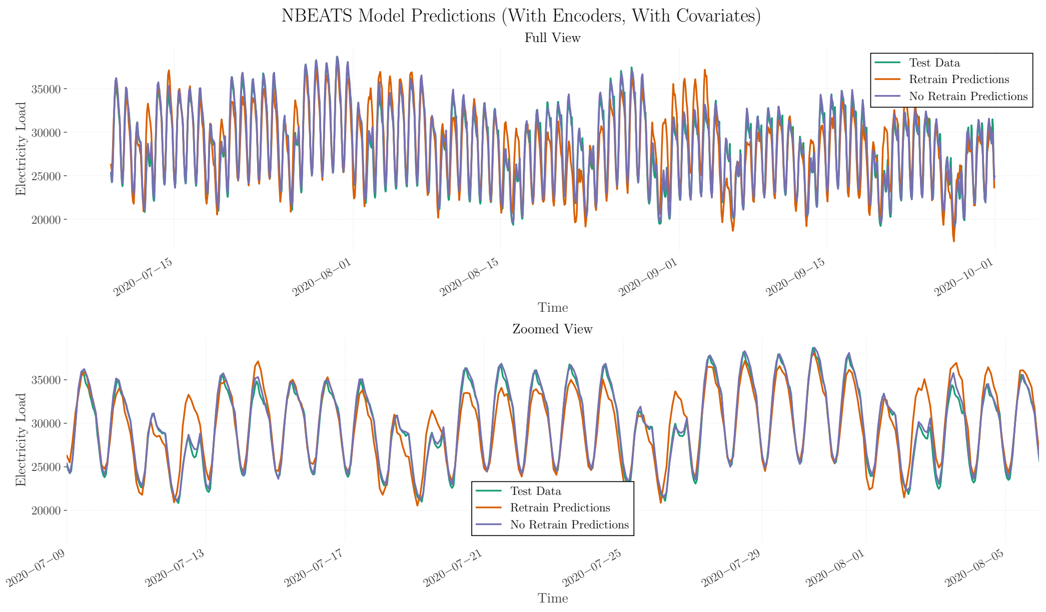


Figure A18

NBEATS model predictions with encoders and covariates - Full-scale and zoomed views

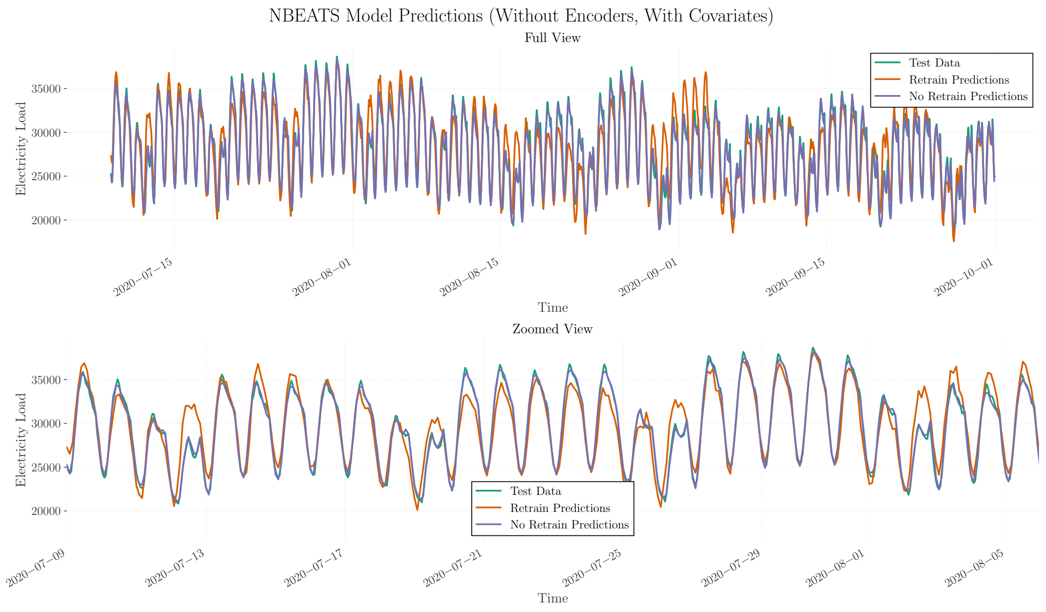


Figure A19

NBEATS model predictions without encoders but with covariates - Full-scale and zoomed views

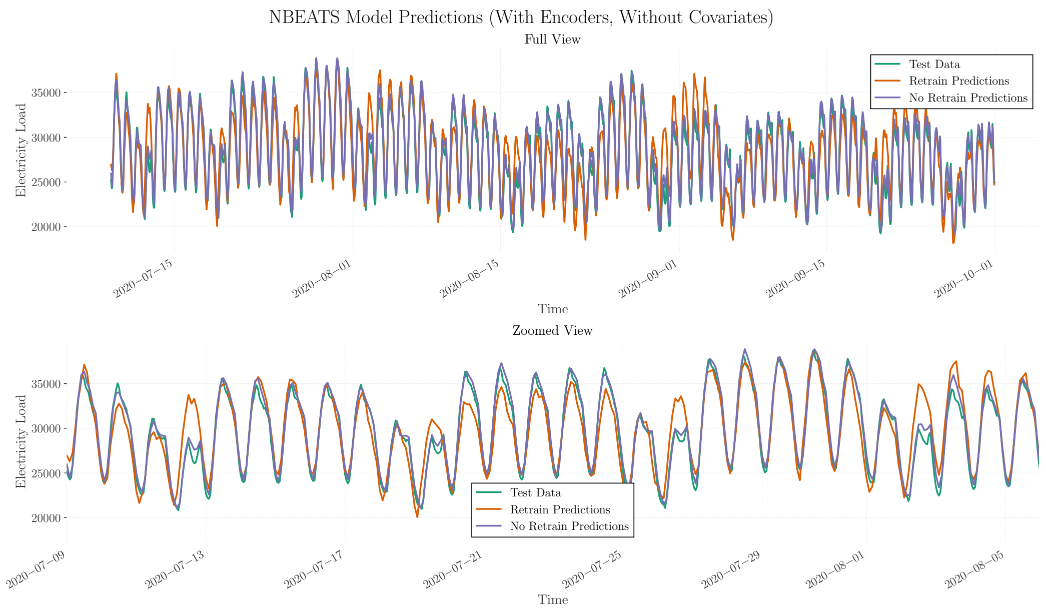


Figure A20

NBEATS model predictions with encoders but without covariates - Full-scale and zoomed views

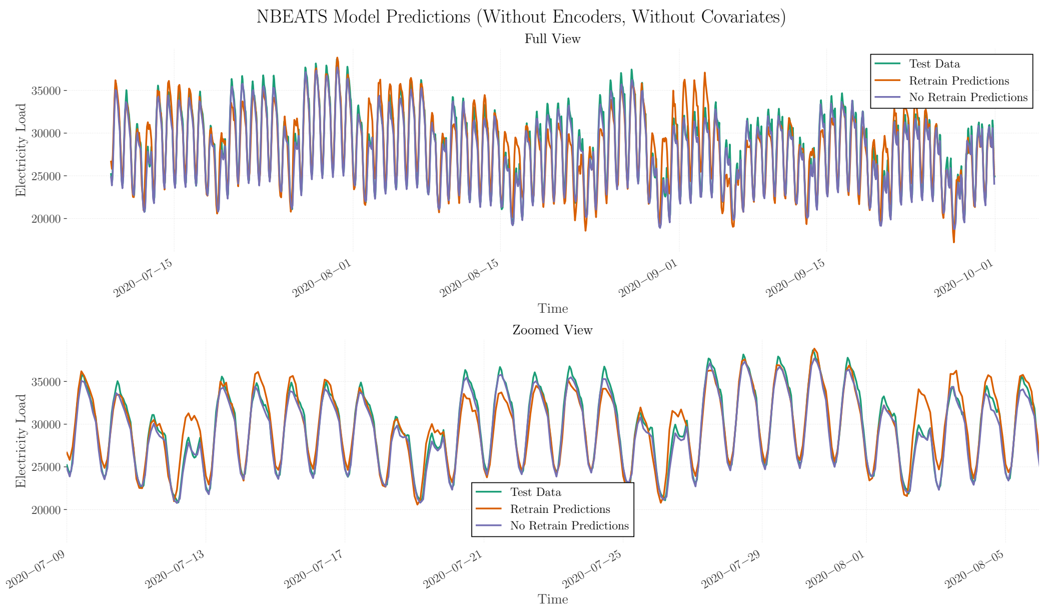


Figure A21

NBEATS model predictions without encoders and without covariates - Full-scale and zoomed views

.10 Non-retrain statistical models (statsforecast library)

.10.1 ARIMA Plots

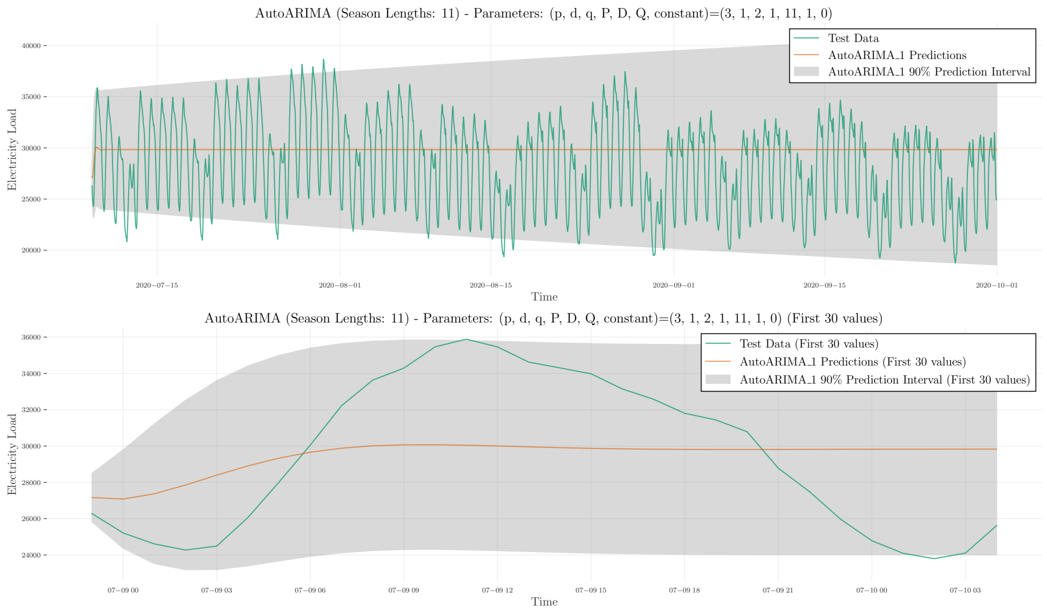


Figure A22

ARIMA model predictions with season length 11

.10.2 MSTL Plots

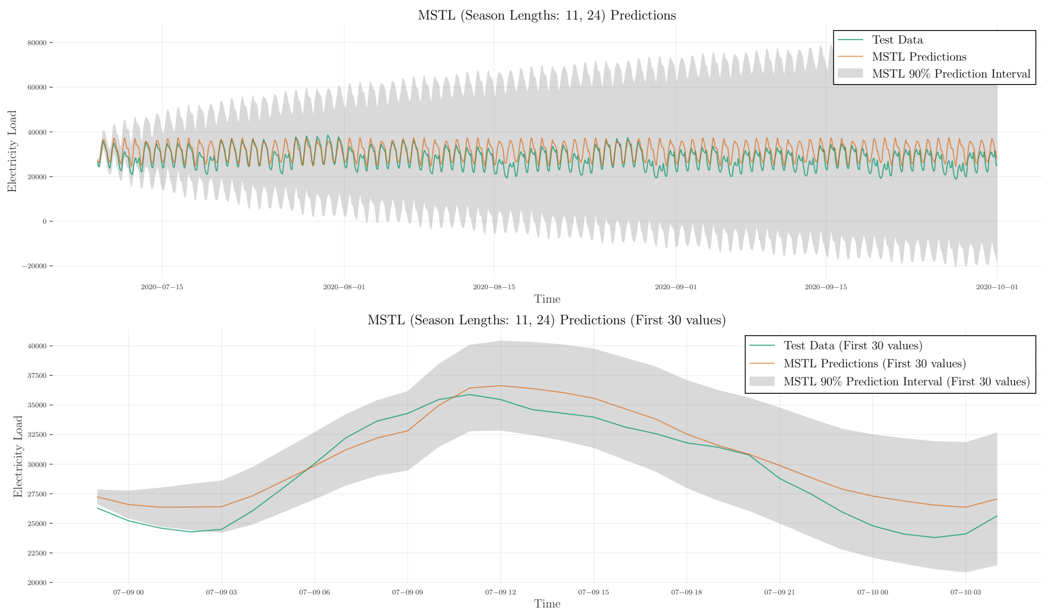


Figure A23

MSTL model predictions with season lengths 11 and 24

.10.3 DOT Plots

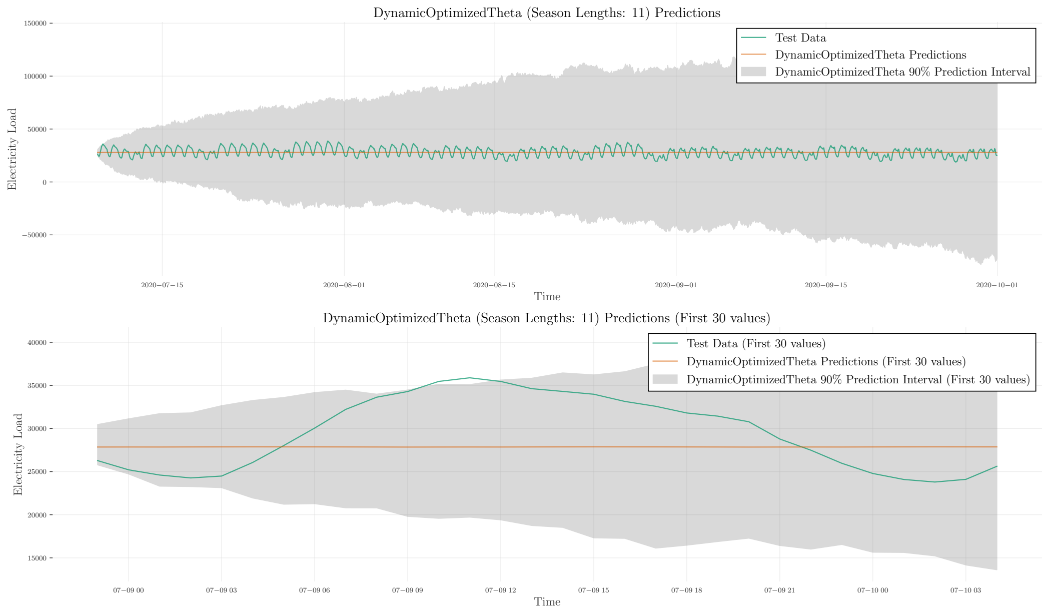


Figure A24

Dynamic Optimized Theta model predictions with season length 11

.10.4 AutoETS Plots

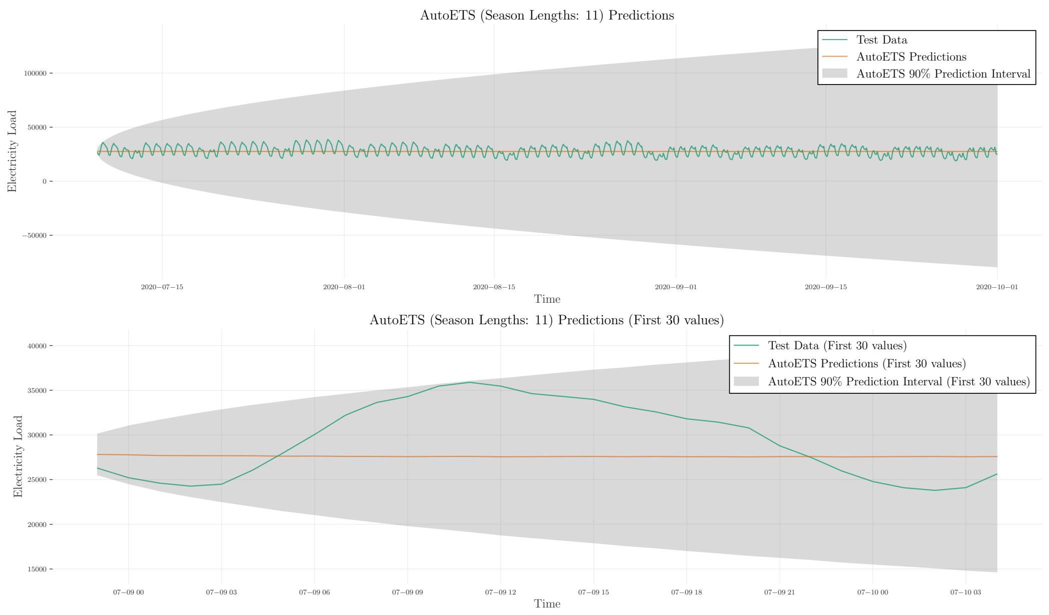


Figure A25

AutoETS model predictions with season length 11

.10.5 AutoCES Plots

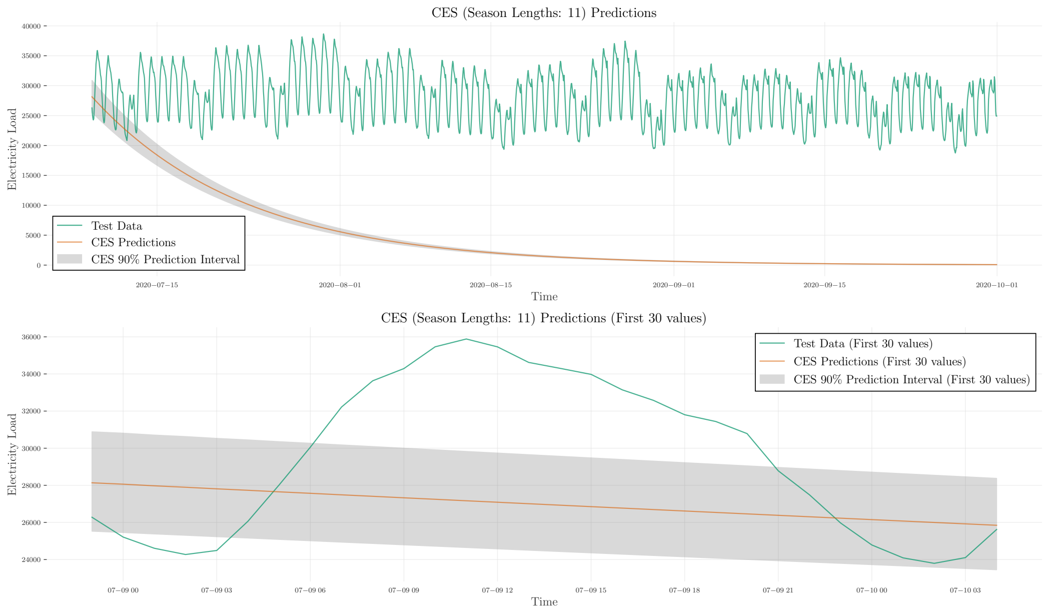


Figure A26

AutoCES model Plots

.10.6 AutoTheta Plots

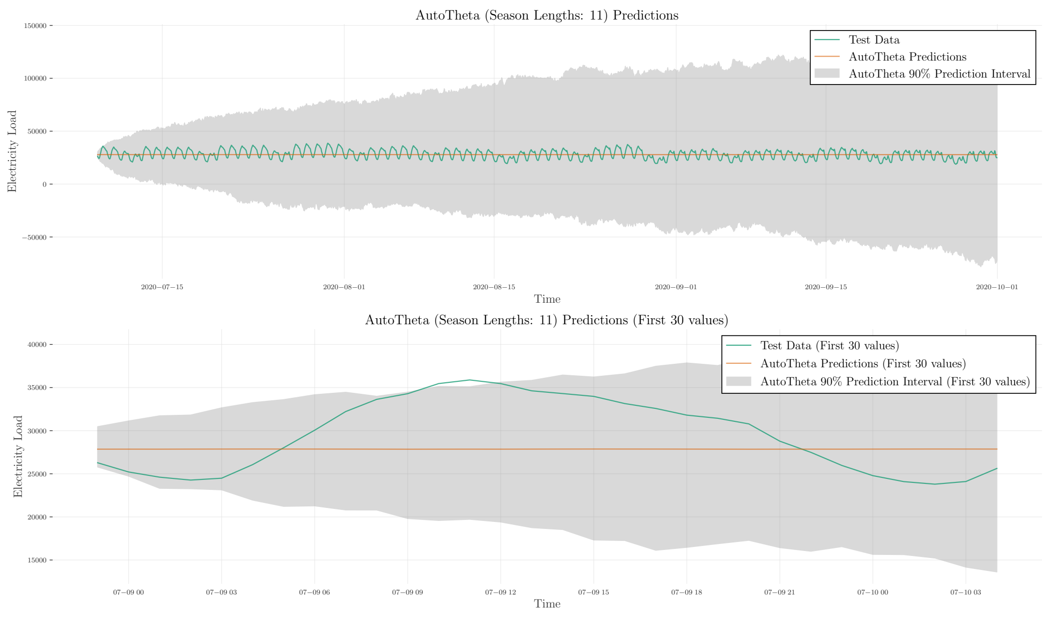


Figure A27

AutoTheta model predictions with season length 11

.10.7 HoltWinters Plots

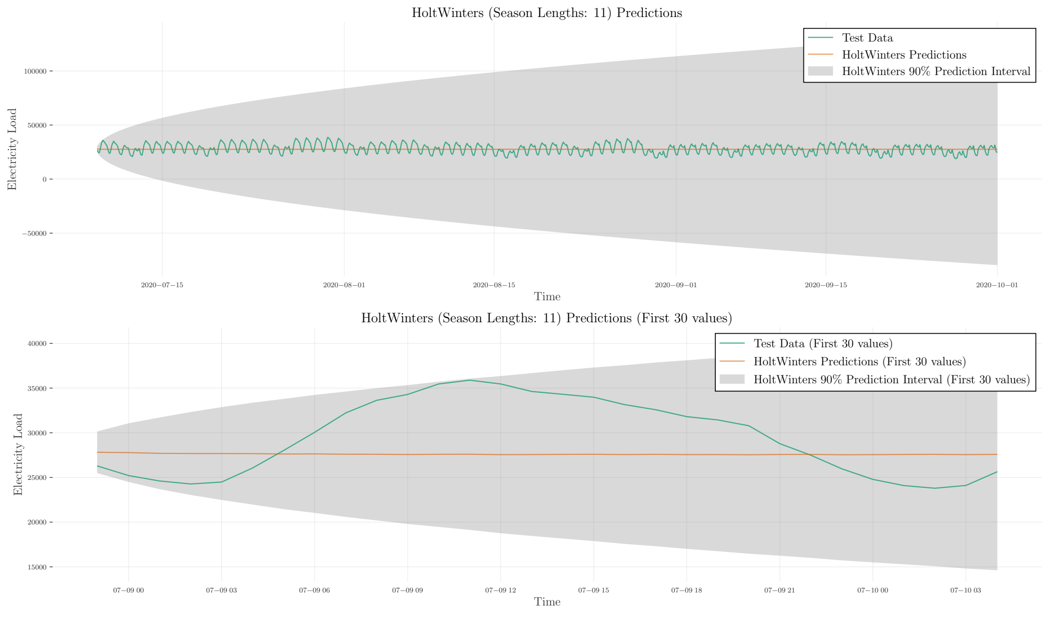


Figure A28

HoltWinters model predictions with season length 11

.11 Retrain statistical models (Darts library)

.11.1 *AutoARIMA Retrain Plots*

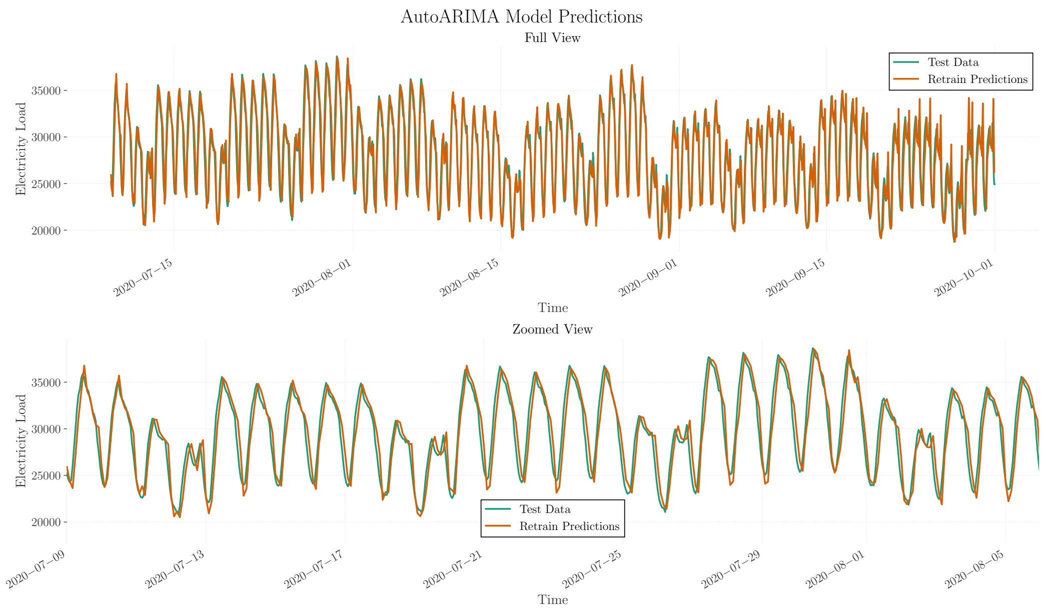


Figure A29

AutoARIMA Retrain Model Predictions

.11.2 AutoCES Retrain Plots

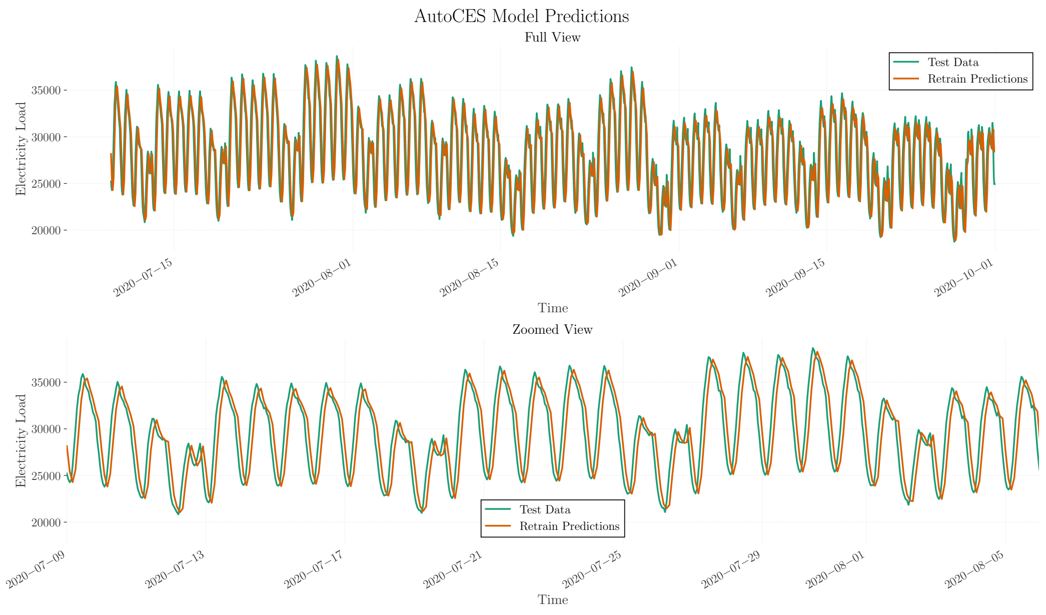


Figure A30

AutoCES Retrain Model Predictions

.11.3 AutoETS Retrain Plots

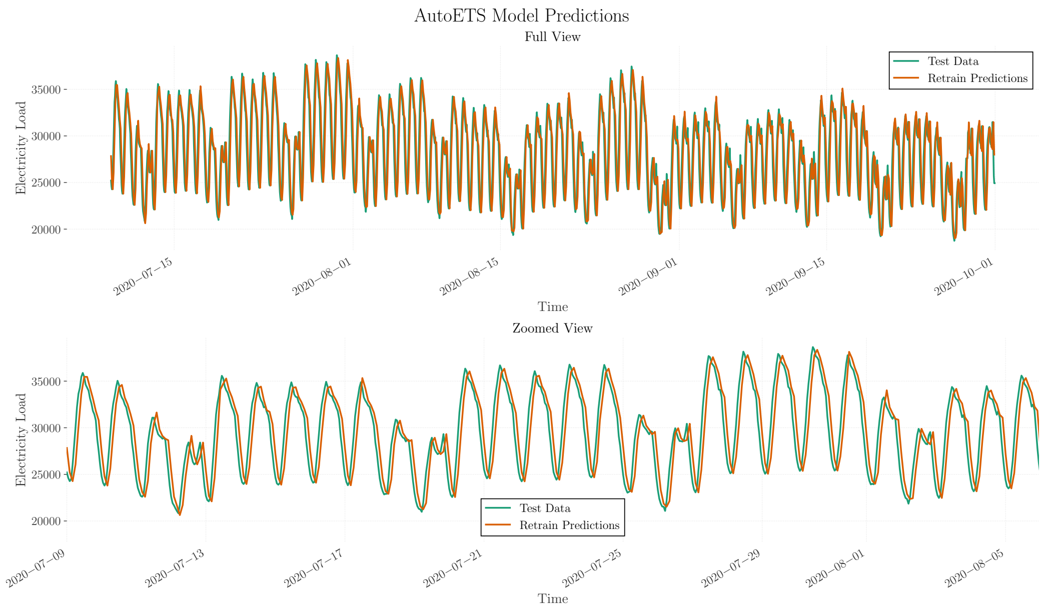


Figure A31

AutoETS Retrain Model Predictions

.11.4 AutoTheta Retrain Plots

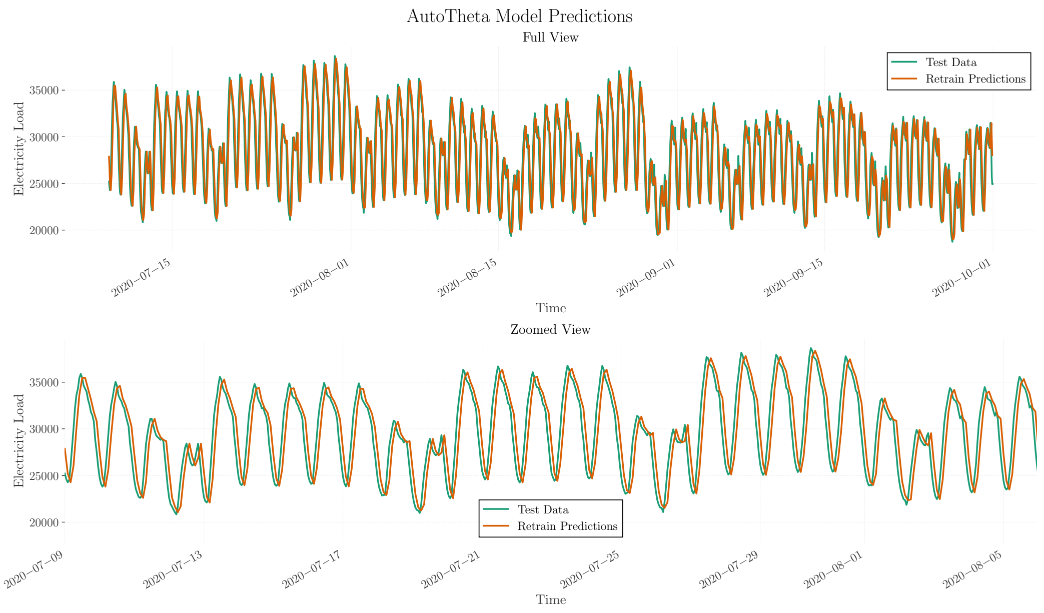


Figure A32

AutoTheta Retrain Model Predictions