Universitat Politècnica de Catalunya (UPC)-BarcelonaTech

Barcelona School of Informatics (FIB)

Improving forecasting accuracy of hourly electricity consumption

by

Cristina Capdevila Choy

Supervisor: Tetiana Klymchuk
(Base Technology and Information Services S.L.U.)

Tutor: Pedro Delicado
(Dept. of Statistics and Operations Research)

Master's degree in Data Science

Barcelona, January 24th 2023

"Only those who will risk going too far can possibly find out how far they can go."

Thomas Stearns Eliot

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Tetiana Klymchuk, for her invaluable guidance, support, and encouragement throughout my time working on this project. I would also like to thank Prof. Pedro Delicado for accepting being my tutor. Finally, I would like to thank my friends and family for their support and encouragement during the writing process.

Cristina Capdevila Choy, January 2023

Abstract

Forecasting electricity consumption plays an important role in the planning and operation of the electricity market. The main problem is that prediction is a difficult and complex task, due to a large number of factors involved, including the non-linearity, non-seasonality, and volatility of the price over time. Electricity providers base their purchases and sales on forecasting and it is essential for them to have the best approximation of the real purchases of consumers since under-estimation will provoke a deficit and over-estimation will result in the penalty of the government, "Red Eléctrica". HolaLuz is one of the relevant electricity providers in Spain with almost 400,000 users dedicated to the commercialization of electrical energy of 100% renewable origin and gas. The company classifies its consumers into predefined segments and uses a forecasting algorithm for the electricity consumption prediction. The algorithms which is Holaluz is currently using are: an XGBoost model and an Heuristic approach. The current implementations have their weaknesses which have a significant effect on its precision. The model is making a prediction based on hourly data such as previous "total_consumptions", temporal features, weather data , and customer's information. The algorithm is executed every day to predict the clients' hourly consumption for the next 2 days. However, it takes about 7 to 10 days to get the total clients' consumption, "total_consumption", which means that at the "execution_time" only a fraction of the total information is available. An improvement of current HolaLuz's forecasting models of hourly electricity consumption is achieved by using advanced Deep Learning techniques. Several models are presented to improve the current models' performance. There are two Machine Learning models: Prophet and XGBoost and two Deep Learning models: DeepAR and Time Fusion Transformer. Temporal Fusion Transformer is the best model and outperforms current HolaLuz's models. It is using a dataset configuration which included features such as labor days, weather, CUPS metadata and time cyclic encodings. Finally the proposed model has been tested as well to predict on a forecast horizon of 14 days which still outperforms HolaLuz's models.

Contents

List of Figures

Chapter 1

Introduction

Electricity consumption forecasting is an important aspect of managing electricity demand and ensuring a stable and reliable electricity supply. It is a difficult and complex task, due to numerous factors involved, including the non-linearity, non-seasonality, and volatility of the price over time. Electricity providers base their purchases and sales on forecasting, and it is essential for them to have the best approximation of the real purchases of consumers since under-estimation will provoke a deficit and over-estimation will result in the penalty of the government, "Red Eléctrica" (REE). Thus, it is of immense value to be able to get a reliable prediction, which, as a result, will maximize the profit of the company.

There are several approaches to forecasting electricity consumption, including classical statistical methods, machine learning algorithms and deep learning algorithms (see [Klyuev et al. 2022] for more details). One of the most common statistical methods for the electricity consumption forecasting is the autoregressive integrated moving average (ARIMA) model, which uses historical data to make predictions about future consumption (see [Nichiforov et al. 2017], [Mahia et al. 2019], [Parzen 1976]).

Electricity consumption forecasting has also been tackled using various machine learning and deep learning techniques (see [Mosavi and Bahmani 2019], [García-Martín et al. 2019] and [Nugaliyadde et al. 2019]).

Those algorithms take into account a wide range of variables and can improve the forecasting accuracy and minimize the error, particularly in cases where the data is complex or non-linear. In the case of deep learning algorithms, some of them incorporate a "memory" component, allowing the model to understand and take into account the time dimension, which can result in more accurate predictions by placing higher importance on

recent data. Overall, the current state of electricity consumption forecasting is characterized by the use of a wide range of methods and technologies, with a focus on improving accuracy and timeliness.

Current thesis was conducted using the data from HolaLuz Company as a part of a master project agreement. HolaLuz is one of the relevant electricity providers in Spain with almost 400,000 users dedicated to the commercialization of electrical energy of 100 per cent renewable origin and gas. The company classifies its customers, they call them CUPS [1], into predefined segments and uses a forecasting algorithm for the electricity consumption prediction for each segment, in this thesis it is denoted "total_consumption", thus it is the target variable.

It is important to mention that HolaLuz has access to the hourly forecasting of electricity demand provided by REE. This prediction is the aggregated forecasted demand of the Spanish population residing in the Iberian Peninsula. The algorithms which is Holaluz is currently using are: an eXtreme Gradient BoostingXGBoost (XGBoost) model and an Heuristic approach. The XGBoost is trained with a dataset of features including weather, labor days and customers info and the "demand". The Heuristic approach is basically the "demand" provided by REE scaled according to the amount of CUPS per segment of Holaluz and the moving average on a fixed time span, current predictions can be found in Appendix A.

Current implementations have its weaknesses which have a significant effect on its precision. Firstly, XGBoost has been used profitably for forecasting time series. Still, being a tree-based model, it has no capability to extrapolate beyond the range of the training data. This is essential for non-stationary time-series. Hence, the model occasionally predicts unreliable "total_consumption" values. On the other hand, the Heuristic approach, which in average is always having good results, is almost only depending on the "demand" from REE, and it is constrained to the performance of REE model which is a black box for HolaLuz. Note that XGBoost is trained with "demand" as a feature as well. Finally and most important, the model is making a prediction based on hourly data such as previous "total_consumptions", temporal features (day of week, festivity day, month), weather data (heat index, max temperature, precipitation), and customer's information. The algorithm is executed every day to predict the clients' hourly consumption for the next 8 days to give time to HolaLuz's electricity buyers to purchase the required electricity

[1] "The Universal Supply Point Code (CUPS) is a unique code which identifies an energy supply point (whether electricity or piped gas) in Spain. To explain a bit more, it is like the Tax ID Code" definition provided by HolaLuz webpage.

per product in the following days (actually, only 2 days are needed, the current and the following day). However, it takes about 7 to 10 days to get the total clients' consumption, "total_consumption", which means that at the moment of prediction, also referred as "execution_time", only a fraction of the total information is available, Figure 1.1 illustrates it.

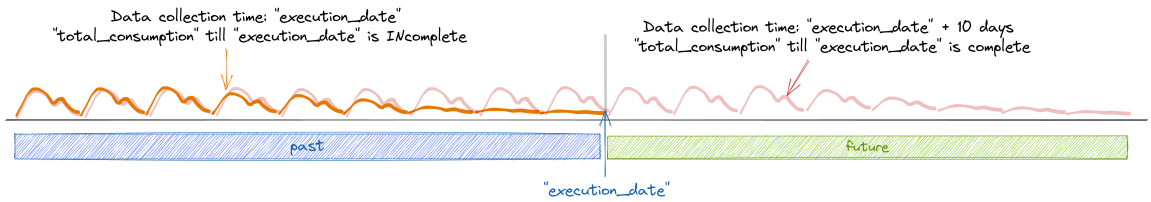


Figure 1.1: Input data at the "execution_date".

There are several approaches one can take to address the challenges described above:

1. Incorporate additional data sources: One way to improve the accuracy of the prediction algorithm is to incorporate additional data sources that may be relevant to electricity consumption. As mentioned in [Klyuev et al. 2022], this could include data on economic indicators, demographic factors, or other variables that may impact electricity consumption (see [Lehna et al. 2022] and [Jain 2019]).

2. Use more advanced machine learning algorithms: The XGBoost algorithm is a powerful machine learning method, but there may be other algorithms that could perform even better for your specific prediction task (see for example [González-Briones et al. 2019], [Salam and Hibaoui 2018], Alrasheedi and Almalaq (2022),[Zhao et al. 2021] and [Rueda et al. 2021]).

3. Consider the impact of missing data: If the algorithm is making predictions based on incomplete data, this could affect its accuracy, [Klyuev et al. 2022]. Try to impute missing values or use methods that are robust to missing data, such as multiple imputation or complete-case analysis, see [Sridevi et al. 2011] and [Baraldi and Enders 2010].

In the current thesis, an improvement of current HolaLuz's forecasting models of hourly electricity consumption is achieved by using advanced Deep Learning techniques.

Improving HolaLuz's forecasting algorithm is important for several reasons:

- Accurate demand forecasting is critical for HolaLuz to ensure a stable and reliable electricity supply.

- Improved demand forecasting leads to cost savings. If HolaLuz can accurately predict peak demand, it can more effectively deploy resources to meet that demand, potentially reducing the need for expensive emergency generation capacity.

- Better demand forecasting can help HolaLuz better serve their customers. HolaLuz can more effectively manage its pricing and billing, potentially leading to lower costs for customers.

- Improved demand forecasting also helps to reduce the environmental impact of electricity generation. For example, if HolaLuz can accurately predict demand, it can more effectively balance the use of renewable and non-renewable energy sources, potentially reducing greenhouse gas emissions.

The thesis is organized as follows:

- An informal introduction into the theory of time series forecasting methods is given in Chapter 2.
- Data Analysis and EDA Insights are given in Chapter 3.
- Model Enhancement process has explained in Chapter 4.
- Model Selection process and results are explained in Chapter 5
- Summary and Future work finalizes study in Chapter 6.
- Appendix contains additional figures including: inputs and feature analysis, predictions and results of all models and the prediction of current Holaluz models.

Chapter 2

A survey of known results about time series forecasting

There has been a significant amount of research on time series forecasting of electricity demand in the last decades. One of the earliest studies in time series forecasting of electricity demand was conducted by [Parzen 1976], it presents the time series modelling problem in terms of whitening filters. In the same year [Brubacher and Wilson 1976] presented, the least squares principle is applied in a dataset of to the problem of estimating missing points in a time series represented by a Box-Jenkins seasonal model. Later in 1996 [Connor 1996] demonstrates a class of recurrent neural networks, NARMA, which show advantages over feed-forward neural networks for time series with a moving average component.

In the last decade they were published some papers about forecasting time series electricity consumption using eXtreme Gradient Boosting (XGBoost) are [Wang et al. 2017], [Li et al. 2018] and [Gokce and Duman 2022]. They all presented XGBoost-based models outperforming classical SVM, ANN and Random Forest methods.

Recent studies have also focused on the use of big data and machine learning techniques for electricity demand forecasting. In 2019 the paper of [Fawaz et al. 2019] applied deep learning techniques for time series forecasting, including feedforward neural networks, recurrent neural networks and long short-term memory networks. In [Alotaibi 2022] authors proposed a deep learning approach for short-term load forecasting using a stacked denoising autoencoder (SDAE) network, a variation of stacked autoencoder (SAE). The authors demonstrated that the proposed approach is able to achieve better forecasting performance compared to traditional methods such as ANN and SVM. Similarly, [Fazlipour et al. 2022] proposed a SAE and highlighted the importance of feature selection

and preprocessing for deep learning-based load forecasting. During the same year, a comparison of hybrid models for forecasting electricity demand was published by [Ruan et al. 2022]. These hybrid models combined ANN with Non-linear Autoregressive models.

Moreover, some other publications forecasting time series electricity consumption using DeepAR the following. In [Rafayal et al. 2022] authors compared that deepAR was the best model for 24h horizon forecasting than ML models such as SARIMAX, RF and DL such as GRU, and LSTM.

Additionally, in [Hernández et al. 2021] and [Zhang et al. 2022], authors demonstrated that the proposed Temporal Fusion Transformer models which achieved better forecasting performance compared to traditional methods such as LSTM, GRU and XGBoost.

Overall, the literature on time series forecasting of electricity demand has been focused on using various methods, such as classical ARIMA, SARIMAX, Exponential Smoothing and also Deep Learning methods such as DeepAR, Temporal Fusion Transformer, etc and some hybrid models. In the following section, an introduction to multiple concepts presented in the literature review and used further in this thesis.

2.1 Time series

2.1.1 Basic concepts

Time series can be defined as a sequence of observations over time. According to [Hyndman and Athanasopoulos 2018] , time series data is characterized by a temporal ordering of observations, where values are recorded in regular time intervals. This natural temporal order is what differentiates time series data from other types of data, such as cross-sectional data.

The time intervals at which observations are recorded can vary, and they define the sequence frequency of the time series. For instance, a time series can be recorded at yearly, monthly, daily, hourly, or even finer intervals, depending on the specific application. For example, in the case of electricity demand forecasting, the time series would be recorded in hourly intervals.

In time series analysis, univariate and multivariate time series refer to the number of variables being observed over time.

- **Univariate**: A univariate time series is a sequence of observations of a single variable over time. For example, the daily closing price of a stock on the stock market would

be a univariate time series. According to [Shumway and Stoffer 2017] univariate time series analysis is focused on modelling and forecasting the behaviour of a single variable over time.

- **Multivariate**: A multivariate time series is a sequence of observations of multiple variables over time. For instance, in the case of electricity demand forecasting, the temperature, humidity and wind are other variables that could be included to improve the predictions. As stated by [Chatfield 1989] multivariate time series analysis deals with more than one variable and their relationships, which can be useful to identify patterns and dependencies in the data.

Time series data has several characteristics that are important to consider when analysing and forecasting the behaviour of the variable over time. According to [Box and Jenkins 1976b], the main characteristics of time series data are:

- **Seasonality**: the presence of regular patterns that repeat over specific intervals of time, such as daily, weekly or yearly patterns.
- **Trend**: a long-term increase or decrease in the data.
- **Level**: the average value of the data over time.
- **Noise**: random variations in the data that do not follow any systematic pattern.

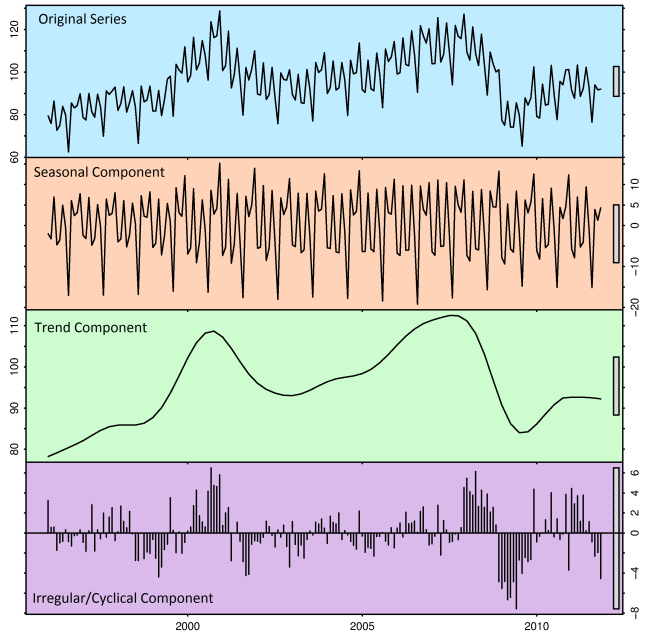


Figure 2.1: Time series characteristics decomposition. Source: Quantdare.

These characteristics can be observed and studied separately, or in combination, see

Figure 2.1. Seasonality and trend are considered to be deterministic components of the time series, while level and noise are considered to be random components.

It is important to note that these characteristics may not always be present in a time series, and the presence of one or more characteristics can affect the forecasting performance. For instance, if a time series has a trend, it would be more difficult to forecast than if it doesn't. Additionally, seasonal patterns can affect the level and the noise of the time series.

Another classification of time series is using the *stationarity* and *non-stationarity* are terms to describe the characteristics of a time series in relation to its mean, variance and autocorrelation. According to [Hamilton 1994], a time series is considered to be stationary if its statistical properties, such as mean and variance, are constant over time. In contrast, a non-stationary time series is one where the statistical properties change over time.

A time series is considered to be stationary if the following conditions are met:

- The *mean* of the time series is constant over time.
- The *variance* of the time series is constant over time.
- The *covariance* between any two observations is not a function of time.

A non-stationary time series, on the other hand, does not meet these conditions and can exhibit trends, seasonality or other patterns that change over time.

It is worth mentioning that a time series can be stationary or non-stationary for different time intervals, for example, a series can be non-stationary at a daily interval but stationary at a weekly interval. Additionally, a non-stationary time series can be transformed into a stationary time series by taking the difference between consecutive observations (first difference) or by applying a log transformation.

2.2 Time series forecasting models

In time series analysis, *parametric* and *non-parametric* methods are two broad categories of techniques used to model and forecast time series data.

Parametric methods assume a specific probability distribution for the data, such as normal, exponential or uniform distribution. These methods typically require fewer data points than non-parametric methods. Examples of parametric methods include the *Autoregressive Integrated Moving Average* (ARIMA) and *Exponential Smoothing* (ETS). Some machine learning algorithms further implemented in this study are also considered as para-

metric, they are: *Prophet*, eXtreme Gradient Boostin (XGBoost), *DeepAR* and *Temporal Fusion Transformer* (TFT). According to [Box and Jenkins 1976b], parametric methods are usually easier to implement and interpret than non-parametric methods, but they may not be appropriate when the assumptions about the underlying distribution of the data are not met.

On the other hand, non-parametric methods make fewer assumptions about the underlying distribution of the data, and they can be used with a wide range of data types. Examples of non-parametric methods include the *k-nearest neighbor* (KNN) method and the *kernel density estimation* (KDE) method. These methods are not implemented in this study. According to [Hyndman and Khandakar 2008], non-parametric methods are more flexible than parametric methods, but they typically require more data points to produce accurate forecasts.

2.2.1 Classical parametric algorithms

There are several classical parametric time series forecasting methods that have been widely used in practice and have been extensively studied in the literature. Some of the most commonly used are explained in the following sections.

2.2.1.1 Autoregressive

Autoregressive (AR) model is based on the idea that the current value of a time series is a linear combination of its past values and a white noise error term. The original paper that proposed the AR model is [Box and Jenkins 1976a]. The general form of an AR(p) model is given by:

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \epsilon_t$$

where y_t is the current value of the time series, c is a constant, ϕ_i are the autoregression coefficients, and ϵ_t is white noise.

2.2.1.2 Moving average

Moving Average (MA) model is based on the idea that the current value of a time series is a linear combination of past errors or white noise. The original paper that proposed the MA model is [Box and Jenkins 1976a].

The general form of an MA(q) model is given by:

$$y_t = \mu + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t$$

where μ is the mean of the time series, ϵ_t is white noise, and θ_i are the moving average coefficients.

2.2.1.3 Autoregressive Integrated Moving Average

Autoregressive Integrated Moving Average (ARIMA) model is a combination of the AR and MA models, and it is used to model and forecast time series data with both trend and seasonality. The original paper that proposed the ARIMA model is "An algorithm for forecasting time series" by [Box and Jenkins 1976b].

The general equation for an ARIMA(p,d,q) model can be represented as:

$$(1 - \phi_1 B - ... - \phi_p B^p)(1 - B)^d y_t = (1 + \theta_1 B + ... + \theta_q B^q)$$

where:

y_t is the observed value at time t,
B is the backward shift operator,
$\phi_1, \phi_2, ..., \phi_p$ are the autoregressive coefficients,
d is the degree of differencing,
$\theta_1, \theta_2, ..., \theta_q$ are the moving average coefficients,
ϵ_t is the white noise error term.

This equation can be broken down into two parts:

- the first part $(1-\phi_1 B-...-\phi_p B^p)(1-B)^d y_t$ represents the autoregressive component
- and the second part $(1+\theta_1 B+...+\theta_q B^q)\epsilon_t$ represents the moving average component.

It is worth to mention that the values of p, d, and q are determined by analysing the properties of the time series data, and it is determined by analysing the Autocorrelation Function (ACF) and the Partial Autocorrelation Function (PACF). They can be described as:

- The ACF plot shows the correlation between a time series and a lagged version of itself. It helps to identify the number of autoregressive (AR) terms (p) in an ARIMA model by showing the correlation between the time series and its lagged values. If there is a significant correlation between the time series and its lags, then it is likely that the time series is autoregressive and an AR term should be included in the model.

- The PACF plot is similar to the ACF plot, but it measures the correlation between a time series and its lags after controlling for the correlation between the time series and all other intermediate lags. It helps to identify the number of moving average (MA) terms (q) in an ARIMA model by showing the correlation between the time series and its lags after controlling for the intermediate lags. If there is a significant correlation between the time series and its lags after controlling for the intermediate lags, then it is likely that the time series is a moving average process and an MA term should be included in the model.

2.2.1.4 Holt-Winters Method Smothing

Holt-Winters exponential smoothing is a method for forecasting time series data with trends. This method is based on the idea of using a weighted average of past observations to forecast future values. Holt-Winters method is a variation of the exponential smoothing method that is specifically designed to handle time series data with both trend and seasonality. It is also known as Triple Exponential Smoothing (TES). Proposed in [Holt 1957] and [Winters 1960], the Holt-Winters method consists of three equations: one for the level, one for the trend, and one for the seasonal component. The general equations for the Holt-Winters method can be represented as:

Level equation: $l_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1})$
Trend equation: $b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$
Seasonal equation: $s_t = \gamma(y_t - l_t) + (1-\gamma)s_{t-m}$

Where:

y_t is the observed value at time t
y_t is the observed value at time t
l_t is the level component at time t
b_t is the trend component at time t
s_t is the seasonal component at time t
α, β, and γ are the smoothing parameters
m is the number of seasons

It is important to notice that the values of the smoothing parameters (α, β, and γ) are determined by analysing the properties of the time series data and the residuals errors of the model.

2.2.2 Machine Learning algorithms

2.2.2.1 Prophet

Prophet is a time series forecasting library for Python and R developed by Facebook's Core Data Science team, presented in [Taylor and Letham 2017]. It is based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is designed for analysing time series with daily observations that display patterns on different time scales such as hourly, daily, and weekly.

The model is composed of two main parts: a trend component and a seasonal component. The trend component is modelled using a piecewise linear or logistic growth curve. The seasonal component is modelled using Fourier series. The model also includes extra regressors, called holidays, which can be added to the model to model the effects of events like Christmas or the Super Bowl.

The general form of the Prophet model is as follows:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

Where:

$y(t)$ is the forecasted value at time t,
$g(t)$ is the piecewise linear or logistic growth curve modelling the trend component,
$s(t)$ is the seasonal component modelled using Fourier series,
$h(t)$ is the holiday component representing the effects of events,
ϵ_t is the error term assumed to be normally distributed with mean 0 and some constant variance.

The trend component is modelled using either a piecewise linear function or a logistic function. The seasonal component is modelled using Fourier series with a fixed number of terms. The holiday component is modelled using a categorical variable for each holiday and additional variables for the days before and after each holiday.

Prophet's forecast is generated by fitting the model to historical data and then generating predictions for the future using Monte Carlo simulations. Prophet uses a Bayesian inference framework to infer the unknown parameters in the model, with a default prior of weakly informative log-normal priors for the trend changepoints and the holiday effects.

2.2.2.2 XGBoost

XGBoost (eXtreme Gradient Boosting) is an open-source library for gradient boosting that was developed by [Chen and Guestrin 2016]. It is widely used in various machine learning tasks, such as classification and regression, and has been proven to be very effective in many applications, including Kaggle competitions.

The basic idea behind gradient boosting is to train a sequence of weak models, and combine their predictions to form a stronger model. XGBoost uses gradient boosting decision trees (GBDTs) as the weak models. A decision tree is a tree-based model that partitions the input space into different regions, and makes predictions based on the region to which a given input belongs. GBDTs are a specific type of decision tree where the final prediction is made by combining the predictions of multiple decision trees.

The training process of XGBoost consists of two main steps. First, fit a base learner to the negative gradient of the loss function. Then, add the base learner to the ensemble and update the ensemble prediction.

The key innovation of XGBoost is the use of a more regularized model formalization to control over-fitting, which gives it better performance. The regularization is achieved by adding a penalty term to the loss function that is minimized during the training process. The objective function for XGBoost is typically represented in the form of a gradient boosting algorithm. The general form of the objective function is as follows:

$$\mathcal{L} = \sum_{i=1}^{n} \ell(y_i, \hat{y}i) + \sum k = 1^K \omega_k \mathcal{T}_k(\omega)$$

Where:

- $\ell(y_i, \hat{y}_i)$ is the loss function, such as logistic loss for binary,
- classification or mean squared error for regression,
- K is the number of weak learners (trees) in the ensemble,
- ω_k is the weight of the k^{th} tree,
- $\mathcal{T}_k(\omega)$ is the regularization term for the k^{th} tree.

The default regularization term in XGBoost is "gammaloss" which is defined as:

$$\mathcal{T}_k(\omega) = \frac{1}{2}\lambda \sum i = 1^T \omega_i^2$$

Where λ is a positive scalar value called the regularization parameter and T is the number of leaves in the k^{th} tree.

So the XGBoost's objective function with gammaloss regularization would be:

$$\mathcal{L} = \sum_{i=1}^{n} \ell(y_i, \hat{y}i) + \frac{1}{2}\lambda \sum k = 1^K \sum_{i=1}^{T} \omega_i^2$$

XGBoost has several advantages over traditional gradient boosting, including faster training speed, better performance, and more flexibility in dealing with missing values and categorical features. Additionally, XGBoost has a number of built-in features that can be used to speed up the training process, such as parallel and distributed computing, and early stopping.

2.2.3 Deep Learning algorithms

2.2.3.1 DeepAR

DeepAR was first introduced in a 2017 paper by AWS researchers published "DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks" [Salinas et al. 2017]. The paper describes the algorithm in detail and presents experimental results on several publicly available datasets that demonstrate its superior performance compared to traditional time series forecasting methods.

The model learns seasonal behaviours and dependencies on given covariates across time series, hence providing covariates is recommended in order to be able to capture complex related behaviours.

DeepAR makes probabilistic forecasts in the form of Monte Carlo samples that can be used to compute consistent quantile estimates for all sub-ranges in the prediction horizon. The approach does not assume Gaussian noise, but can incorporate a wide range of likelihood functions, which allows choosing one that is appropriate for the statistical properties of the data.

Architecture

The architecture of the proposed model for time series forecasting is depicted in Figure 2.2. The input at each time-step comprises the data point preceding the current time-step's data, as well as the previous network's output. Additional input, such as covariates, are not represented in this diagram for simplicity.

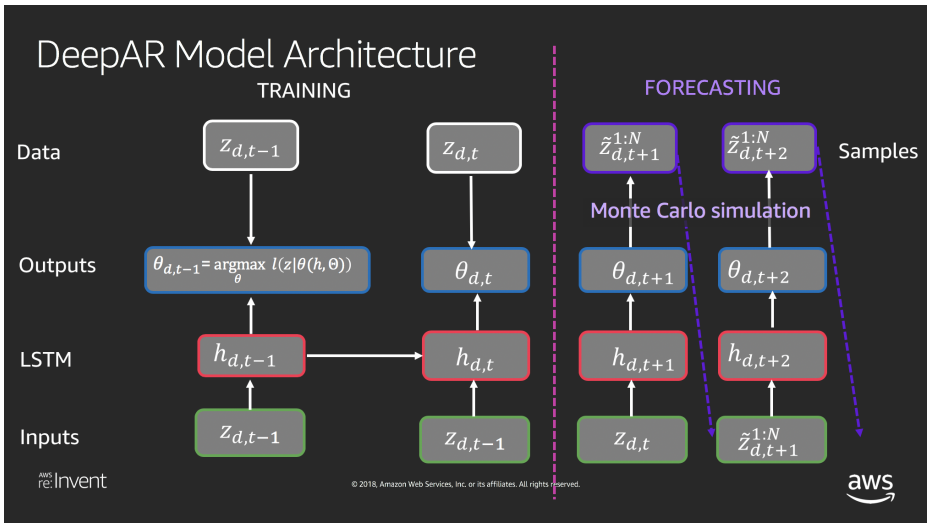


Figure 2.2: DeepAR Architecture. Source: aim457.

The model utilizes a Recurrent neural network (RNN) with long short-term memory (LSTM) layers, which are represented in red, and the final hidden layer produces the $h_{d,t}$ value. This value is then passed through an activation function for each parameter of the specified likelihood. The parameters of the activation function are learned through the use of Stochastic Gradient Descent (SGD) on the $h_{d,t}$ value at time t and the data up until time t, in order to maximize the likelihood at that time.

The output layer, represented in blue, utilizes the SGD-optimized activation functions to output the maximum likelihood parameters. This process is how the model, referred as DeepAR, trains itself to the input data. The model is capable of providing probabilistic forecasts for the next time-step, once it is trained.

The output layer, shown in blue, uses the optimized activation functions to output the maximum likelihood parameters. This process is how the DeepAR model is trained on the input data.

To generate probabilistic forecasts for the next time step, the model utilizes the optimized activation functions to produce the maximum likelihood parameters for time $t + 1$. The model then simulates Monte Carlo (MC) samples from this likelihood, resulting in an empirical distribution for the predicted data point, as shown in purple. The MC samples produced at time $t + 1$ are used as input for time $t + 2$, and so on, until the end of the prediction horizon. This allows the model to provide a confidence interval around the point estimate.

Formulation

The RNN with LSTM units to model the dependencies between time steps of a time series. Given a time series y_t of length T and a set of external covariates x_t, the model learns to predict the probability distribution of future values $y_t + h$, for $h > 0$, given the past observations y_t and x_t.

The model can be formulated as follows:

$$p(y_{t+1:t+h}|y_{1:t}, \mathbf{c}t) = \prod i = t + 1^{t+h} p(y_i|y_{1:t}, \mathbf{c}_i)$$

Where:

- $y_{t+1:t+h}$ is the vector of future values to be predicted at times $t + 1$ to $t + h$,
- $y_{1:t}$ is the vector of historical values up to time t,
- $\mathbf{c}_t$ is a vector of categorical variables that may be used to condition the prediction,
- $p(y_i|y_{1:t}, \mathbf{c}_i)$ is the likelihood function for predicting,
- y_i at time step i given the historical values $y1 : t$ and the categorical variables $\mathbf{c}_i$.

2.2.3.2 Temporal Fusion Transformer

The Temporal Fusion Transformer (TFT) is a deep learning-based model for forecasting time series data. It is built on top of the *Transformer* architecture, which is known for its ability to handle sequential data and has been successful in many natural language processing tasks.

Transformer

Transformer is a neural network architecture for natural language processing tasks such as language translation, text summarization and language model that was introduced in the 2017 paper by Google researchers in [Vaswani et al. 2017]. The architecture is based on the concept of self-attention, which allows the model to weigh the importance of different parts of the input when making predictions.

As shown in Figure 2.3, the Transformer model consists of an encoder and a decoder, each made up of a stack of identical layers. The encoder takes the input sequence and produces a fixed-length context vector, which is then used by the decoder to generate the output sequence.

The key component of the Transformer architecture is the attention mechanism. The attention mechanism is implemented as a dot-product attention function, which is defined

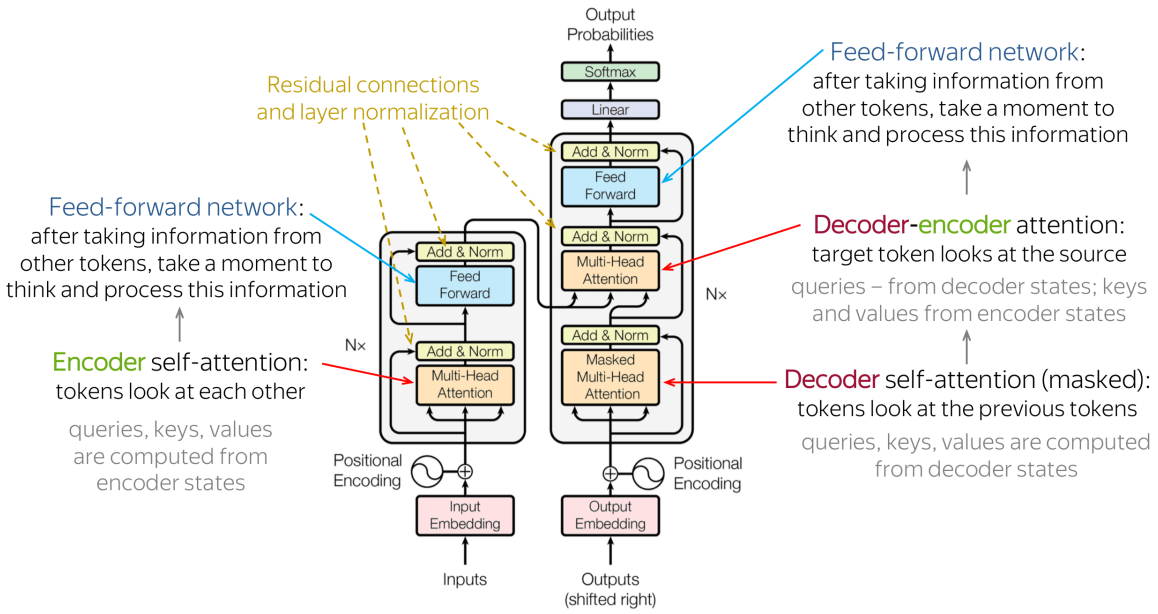


Figure 2.3: Transformer Architecture. Source: Adaptation of Lena. Original source: [Vaswani et al. 2017].

mathematically as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where Q, K, and V are matrices representing the query, key, and value respectively and d_k is the dimension of the key. The dot product of Q and K is scaled by $\frac{1}{\sqrt{d_k}}$ to prevent the dot product from becoming too large. *Softmax* is applied to the dot product to obtain the *attention weights*.

The attention function calculates the dot product between the queries and keys, and then applies a softmax function to obtain a probability distribution over the keys. This distribution is then used to weigh the values and obtain a weighted sum, which represents the output of the attention mechanism.

The Transformer model architecture is the base of many other models such as BERT, GPT-2, and GPT-3. It has been used in a variety of natural language processing tasks, and has been shown to achieve state-of-the-art results on many benchmarks. The transformer is an important step for the natural language processing field and has a great impact in the field.

Time Fusion Transformer Architecture

Temporal Fusion Transformer is a neural network architecture for time series forecasting that combines the transformer architecture with temporal convolutional networks

(TCNs). This architecture was first introduced in a 2020 paper by researchers from Google Brain, [Lim et al. 2019].

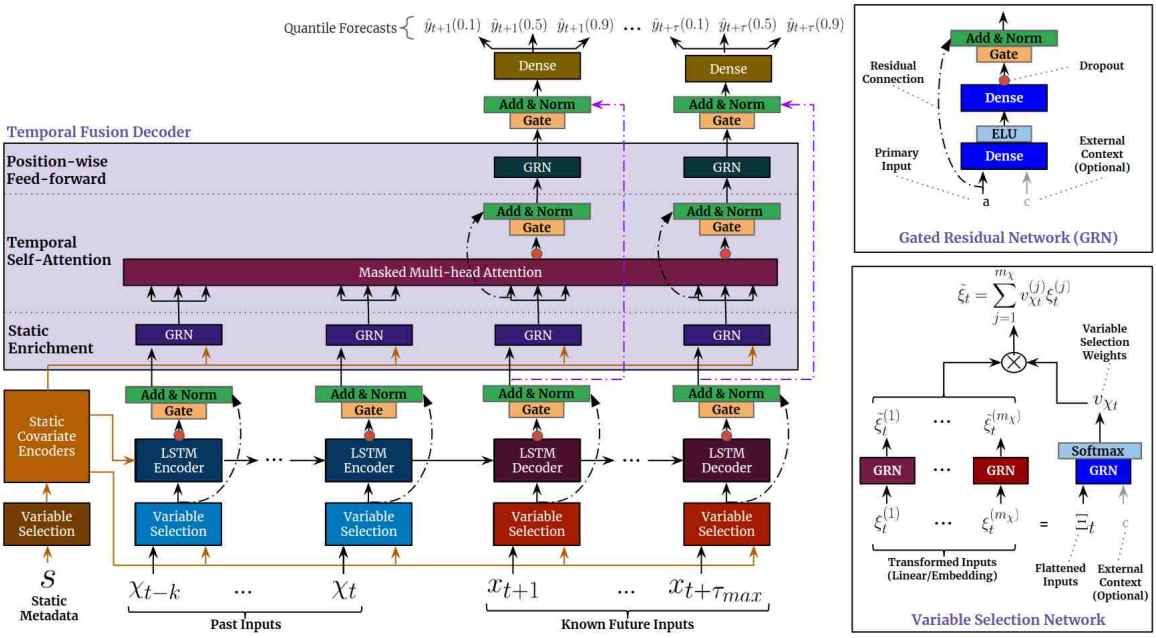


Figure 2.4: TFT Architecture. Source: [Lim et al. 2019].

The Temporal Fusion Transformer architecture, Figure 2.4, consists of two main components: a transformer encoder that captures long-term dependencies in the time series data and a temporal convolutional network (TCN) decoder that captures short-term dependencies.

The transformer encoder takes the input time series and produces a fixed-length context vector, which is then used by the TCN decoder to generate the output time series. The TCN decoder is a stack of dilated causal convolutional layers, which allows the model to increase the receptive field exponentially with the depth of the network.

The key component of the Temporal Fusion Transformer architecture is the fusion mechanism, which combines the information from the transformer encoder and the TCN decoder. The fusion mechanism is implemented as a residual connection, which is defined mathematically as follows:

$$y = \text{TCN}(x) + \text{Transformer Encoder}(x)$$

Where x is the input to the TFT, y is the final output of the TFT, $\text{TCN}(x)$ is the output of the TCN decoder and $\text{Transformer Encoder}(x)$ is the output of the Transformer encoder.

To finish, the Temporal Fusion Transformer model is trained by minimizing the mean squared error (MSE) between the predicted and the actual values, using the Adam optimizer.

2.3 Model performance

In the field of time series forecasting, various techniques are employed to improve the performance of the models and their generalization ability. One of the main concerns is to reduce the bias of the models, which refers to the tendency of the model to systematically over- or under-estimate the true values.

dataset split and cross-validation are commonly used techniques to evaluate the performance of the models. dataset split involves dividing the data into a training set and a test set, where the training set is used to train the model, and the test set is used to evaluate the model's performance. Cross-validation, on the other hand, is a technique that involves training the model on different subsets of the data and evaluating it on the remaining subset. This allows to test the model's ability to generalize to unseen data and to identify overfitting.

Regularization is another technique that is used to prevent overfitting. This is achieved by adding a penalty term to the cost function to discourage large weights. This technique helps to reduce the variance of the model and increase its generalization ability.

Hyperparameter tuning is the process of optimizing the values of the model's hyperparameters, which are the parameters that control the model's capacity and the trade-off between bias and variance. This technique allows finding the best set of hyperparameters that minimize the error and increase the model's generalization ability [Hyndman and Khandakar 2008].

2.3.1 Bias in time series forecasting models

Machine learning model bias refers to a systematic error or deviation of the model's predictions from the true values. Bias occurs when the model makes assumptions about the data that are not accurate, leading to an over- or under-estimation of the true values, [Makridakis et al. 2018]. Bias can be caused by various factors such as the choice of the model, the data used to train the model, and the model's parameters. In time series forecasting bias can appear in different forms, including model bias and data bias [Hyndman and Athanasopoulos 2018].

Model bias, also known as *underfitting*, occurs when a model is too simple to capture the underlying patterns in the data. This can lead to a poor fit and low forecasting accuracy. To overcome this issue, more complex models or a combination of models can be used.

Data bias, also known as *overfitting*, occurs when a model is too complex and fits the noise in the data rather than the underlying patterns. This can lead to a high forecasting accuracy on the training data but poor performance on unseen data. To overcome this issue, techniques such as cross-validation, regularization, and ensemble methods can be used.

Another type of bias that can appear in time series forecasting is temporal bias, which occurs when the model is not able to capture the temporal dependencies in the data. This can lead to poor forecasting performance, particularly for long-term.

In addition, another bias that can appear in time series forecasting is the stationarity bias. This occurs when the model assumes that the data is stationary, while it is not.

2.3.2 Dataset split: train, validation and test

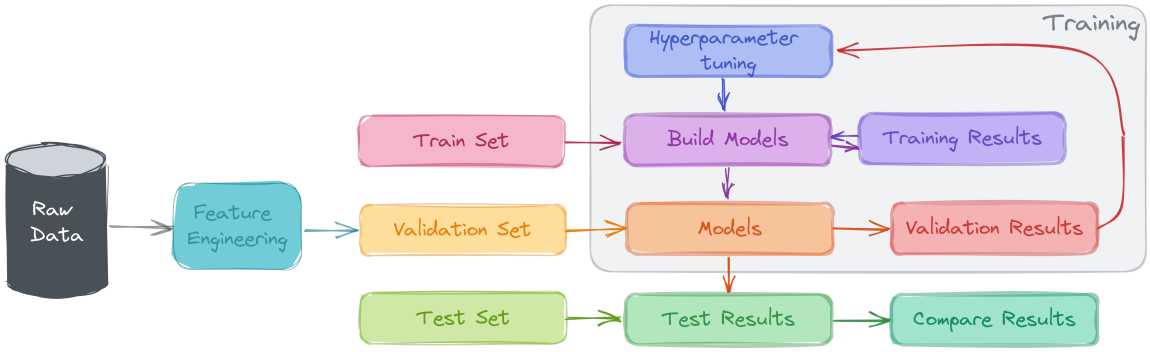


Figure 2.5: Machine Learing Flow.

Splitting a dataset into train, validation, and test sets is an important step in the machine learning process because it helps to prevent overfitting and evaluate the model's performance on unseen data.

Overfitting occurs when a model is trained too well on the training data and performs poorly on new, unseen data. This is because the model has learned the noise and random variations present in the training data, and is not able to generalize to new data. To prevent overfitting, it is necessary to evaluate the model's performance on a separate dataset, which is called the validation set. The validation set is used to tune the model's hyperparameters in order to optimize the model's performance.

The test set is a separate dataset that is used to evaluate the model's performance on unseen data. This is important because it provides an estimate of how well the model will perform on new data in the real world. It is a good practice to keep the test set separate and untouched until the final evaluation of the model, this way it ensures that the model is not trained or tuned on the test set.

2.3.3 Cross-validation

Cross-validation is a technique used to evaluate the performance of a machine learning model by training it on a subset of the data and evaluating it on a held-out subset of the data. However, it can be challenging to perform cross-validation on time series forecasting problems for several reasons.

- **Temporal dependencies**: Time series data typically has temporal dependencies, meaning that the observations are not independent and identically distributed (i.i.d). This can make it difficult to use traditional cross-validation techniques such as k-fold cross-validation, which assume that the observations are i.i.d.

- **Small datasets**: Time series forecasting problems often have small datasets, which can make it difficult to divide the data into training and validation sets. When the dataset is small, the validation set may not be representative of the data distribution, leading to poor performance estimates.

- **Stationarity**: Time series data may not be stationary, meaning that the mean and variance of the data may change over time. This can make it difficult to use the same validation set across different time periods.

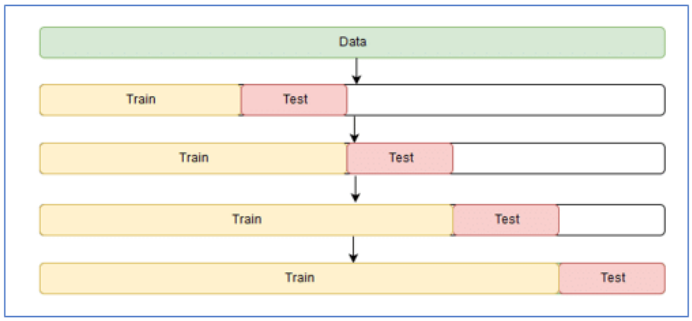


Figure 2.6: Time-based cross validation schema. Source: [Vineeth et al. 2020].

To overcome these challenges, alternative cross-validation techniques have been proposed such as rolling window cross-validation, expanding window cross-validation, and walk-forward validation.

- **Time-based cross-validation**: This method involves splitting the time series data into chunks based on time rather than randomly, Figure 2.6. The most common approaches are the rolling window method and the expanding window method. The rolling window method involves moving a fixed-size window along the time series and using each window as the test set in turn, while the expanding window method involves starting with a small test set and gradually increasing it with each iteration [Jong 1988].

- **Walk-forward validation**: This method involves dividing the time series data into a sequence of overlapping windows, where each window is used to forecast the next one. This method is particularly useful for long-term predictions, as it allows the model to generalize well to unseen future data [Kirkpatrick and Dahlquist 2010].

- **Leave-one-out cross-validation**: This method involves training the model on all but one observation, and using the left-out observation as the test set. This process is repeated for each observation in the time series, resulting in numerous training and test sets. This method can be computationally expensive, but it can provide a more robust evaluation of the model's performance.

It is worth mentioning that the choice of the cross-validation technique depends on the characteristics of the time series and the goal of the evaluation, and it is important to use the appropriate evaluation metric.

2.3.4 Regularization

Regularization is a technique used in machine learning to prevent overfitting. It works by adding a term to the model's cost function that penalizes certain model parameters if they take on large values. The purpose of this is to constrain the model, making it simpler and less prone to overfitting.

In simple terms, regularization is a way of keeping the model from becoming too complex. It does this by adding a penalty term to the model's loss function, which is a measure of how well the model fits the data. This penalty term encourages the model to have smaller parameter values, which in turn makes the model less complex.

There are two main types of regularization commonly used in machine learning: *L1 regularization* and *L2 regularization*. *L1 regularization*, also known as *Lasso regularization*, adds a penalty term to the cost function that is proportional to the absolute value of the

parameters. *L2 regularization*, also known as *Ridge regularization*, adds a penalty term to the cost function that is proportional to the square of the parameters.

2.3.5 Hyperparameter tuning

Hyperparameter tuning is the process of systematically searching for the best combination of hyperparameters for a machine learning model. There are several techniques that can be used for hyperparameter tuning:

- **Grid search**: This technique involves specifying a set of possible values for each hyperparameter, and then training the model for every possible combination of hyperparameter values. Grid search can be computationally expensive, but it can be useful for models with few hyperparameters. The performance of the model is then evaluated using a validation set, and the combination of hyperparameters that results in the best performance is chosen [Bergstra and Bengio 2012].

- **Random search**: This technique involves randomly sampling from the possible hyperparameter values, and then training the model for each set of sampled values. Random search is less computationally expensive than grid search, but it may not find the optimal set of hyperparameters. This can be more efficient than grid search when the number of hyperparameters is large [Bergstra and Bengio 2012].

- **Bayesian optimization**: This technique uses a probabilistic model to guide the search for the best hyperparameters. It can be more efficient than grid search and random search, as it tries to focus the search on promising regions of the hyperparameter space. It starts with a prior distribution over the possible hyperparameters, and updates the distribution after each iteration based on the performance of the model on the validation set [Snoek et al. 2012].

- **Gradient-based optimization**: This technique uses gradient information to directly optimize the hyperparameters. This method is often used in deep learning, and it can be efficient, but it can be sensitive to initialization. It starts with an initial set of hyperparameters, and then iteratively updates the hyperparameters in the direction of the gradient [Maclaurin et al. 2015].

It is important to note that the best technique for hyperparameter tuning depends on the specific task and the model architecture, and it is often a good idea to try several different techniques and compare their performance.

2.3.6 Regression evaluation metrics

Regression evaluation metrics are used to evaluate the performance of a regression model. Some commonly used regression evaluation metrics include: **Mean Absolute Error** (MAE): It measures the average absolute difference between the predicted values and the true values. It is calculated using the following formula:

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

Mean Squared Error (MSE): It measures the average squared difference between the predicted values and the true values. It is calculated using the following formula:

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

Root Mean Squared Error (RMSE): It is the square root of the MSE. It is calculated using the following formula:

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

R-Squared (R^2): Also known as the coefficient of determination, it measures the proportion of the variance in the dependent variable that is predictable from the independent variable(s). It is calculated using the following formula:

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

In these equations n is the number of observations, y_i is the true value of the $i-th$ observation, $\hat{y}_i$ is the predicted value of the $i-th$ observation.

Chapter 3

Data Analysis

This chapter serves as a foundation for the entire study and sets the stage for the modeling and evaluation that follows. We start by providing an overview of the data that is be used such as the source of the data, the number of observations, and other relevant characteristics or features of the data.

Next, we provide a description of the data preprocessing and cleaning steps that were performed including information about handling missing values, outliers, and some other issues that were encountered with the data. Once the data has been preprocessed, some exploratory data analysis (EDA) is provided to give an insight into the data and its structure.

Finally, the chapter is concluded by summarizing the key findings of the data analysis, highlighting important patterns, relationships, and insights that were discovered, and discussed how the data will be used in the subsequent chapters of the study.

3.1 Dataset Overview

The data for this study was collected from multiple sources. The primary source of data was a HolaLuz company that provided access to their historical electricity consumption data. Supplementary data was collected from open sources, including

- weather data from GFS: The Global Forecast System which is the cornerstone of NCEP's suite of numerical guidance that provides global atmospheric and wave predictions at 13 km resolution from NOAA.
- consumption data from ESIOS: which is a public website where the non confidential

information, result of the Electric System Operator (they denote it as SO) market operations or other information of public interest, related with the electricity markets is published by "Red Eléctrica Española", REE.

- demographic data from PAGe: which is the acronym of "Punto de Acceso General electrónico (PAGe). It is a web-page of "Administración General del Estado". It constitutes a unique access for the citizens to all public administrations: "Estatal", "Autonómica", "Local" and from the European Union.

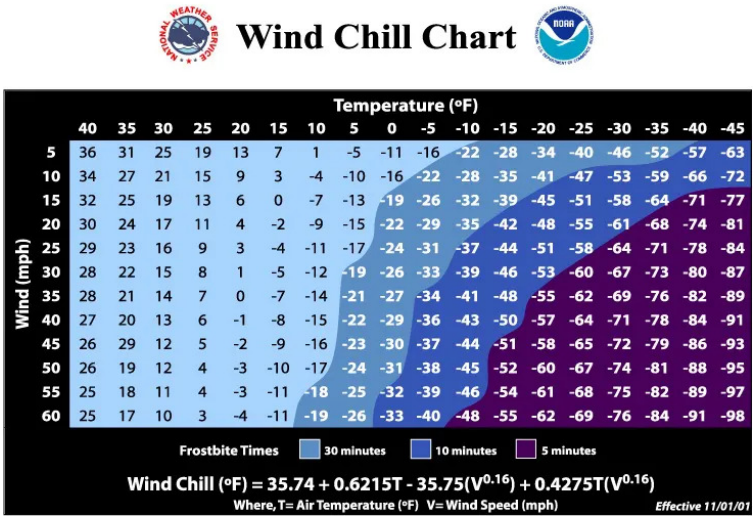


Figure 3.1: Wind Chill Table. Source: NOAA.

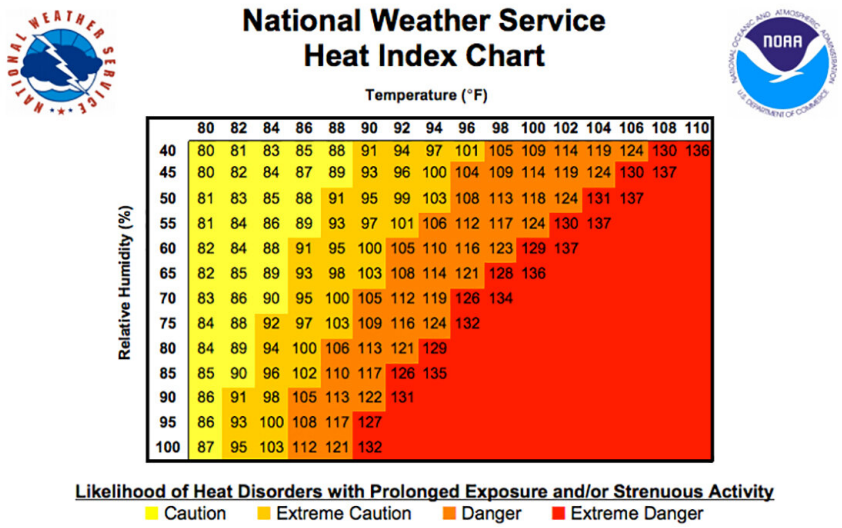


Figure 3.2: Heat Index Table. Source: NOAA.

The data covers a period of 16 months, from July 2021 to December 2022 both included. This date window was chosen because it covers the period of time after significant changes in fares were implemented that resulted in changes in the consumption patterns and it was necessary to use more recent data for the analysis. Using older data would not have been suitable for the study as it would not have reflected the impact of these changes.

Dataset contains 9,489 rows in total and has the following features:

- *datetime* (index): Date and time in UTC.
- *total_consumption* (target): Aggregated observed consumption, in MW, for all Holaluz's CUPS living in the Iberian Peninsula, belonging to a specific product called "3.0TD", mainly for industries. Source: ESIOS.
- *total_active_cups*: Total number of Holaluz's CUPS living in the Iberian Peninsula, belonging to a specific product called "3.0TD" in the current date. Source: ESIOS.
- *total_cups_with_consumptions*: Total number of CUPS living in the Iberian Peninsula, belonging to a specific product called "3.0TD" whose consumption is informed in the data system. Source: ESIOS.
- *demand*: It is the predicted electricity demand by all costumers living in the Iberian Peninsula by the algorithm of Red Eléctrica Española. This is actually the *total_consumption* prediction but for the hole Peninsula. Hence, it is a very special feature because it is highly related with the target value, indeed it is the same concept but in a higher aggregation level. It should be said that the incorporation of this variable to the forecasting model is quite dangerous since it is already a forecasting of the target. Furthermore, the forecasting model used to predict it is not known neither the source code. Source: ESIOS.
- *weighted_labor_days*: Considering that a labor day is set to 1 and non-labor day to 0. This value is the weighted average consider the autonomous communities festivity calendar weighted with the Holaluz's CUPS in each. Source: PAGe.
- *availability*: It is the ratio between *total_cups_with_consumptions* and *total_active_cups*. It represents the completeness of the *total_consumption*, the total number of CUPS whose information is already in the data system from the total set of active CUPS for the given product. Source: Derived from others.
- *snow_depth*: Amount of snow accumulation on the ground in meters. Source: GFS.

- *temperature*: Physical quantity that expresses quantitatively the perceptions of hotness and coldness. in Kelvin degrees.Source: GFS.
- *relative_humidity*: Percentage representing concentration of water vapor present in the air. Source: GFS.
- *temperature_max*: Maximum temperature of the day in Kelvin degrees. Source: GFS.
- *temperature_min*: Minimum temperature of the day in Kelvin degrees. Source: GFS.
- *precipitation_rate*: The rate of rainfall measured in millimeters per hour. Source: GFS.
- *short_wave_radiation*: Measure of the radiant energy emitted by the sun in the visible and near-ultraviolet wavelengths. Source: GFS.
- *wind_speed*: Wind flow velocity in meters per second. Source: GFS.
- *windchill*: It is an index used to provide an accurate, understandable, and useful formula for calculating the dangers from winter winds and freezing temperatures, see Figure 3.1. Source: GFS.
- *heat_index*: A measure of how hot it really feels when relative humidity is factored in with the actual air temperature. The National Weather Service proposes a table which related the relative humidity with the air temperature to obtain a heat index value, see Figure 3.2. Source: GFS.

	count	mean	std	min	25%	50%	75%	max
availability	9,480.0	1.0	0.0	0.9	1.0	1.0	1.0	1.0
demand	9,480.0	27,215.3	4,299.6	17,393.0	23,675.9	27,443.1	30,479.6	37,868.8
heat_index	9,356.0	30.6	6.8	20.3	25.3	28.4	33.9	59.2
precipitation_rate	9,356.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
relative_humidity	9,356.0	60.2	12.5	26.3	50.6	61.0	69.8	92.9
short_wave_radiation	9,356.0	187.8	253.6	0.0	0.0	27.2	333.5	967.7
snow_depth	9,356.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
temperature	9,356.0	290.0	7.1	276.4	283.8	289.1	295.9	307.2
temperature_max	9,356.0	291.0	7.4	276.6	284.7	290.1	297.3	307.8
temperature_min	9,356.0	288.9	6.9	276.2	283.0	288.0	294.5	306.4
total_active_cups	9,480.0	7,791.3	586.5	6,768.0	7,150.0	7,875.0	8,299.0	8,653.0
total_consumption	9,480.0	23,795.5	9,636.6	10,365.8	16,049.3	21,016.2	30,158.4	54,550.5
total_cups_with_consumptions	9,480.0	7,603.0	484.6	6,727.0	7,057.0	7,779.0	8,015.0	8,254.0
weighted_labor_days	9,480.0	0.7	0.5	0.0	0.0	1.0	1.0	1.0
wind_speed	9,356.0	3.1	1.3	0.9	2.2	2.8	3.9	9.1
windchill	9,356.0	16.8	8.6	-0.2	9.3	15.7	23.9	37.5

Table 3.1: Description of the statistics of each feature in the raw dataset.

A comprehensive overview of the raw dataset is presented in Table 3.1. The table includes various statistics that summarize the central tendency, dispersion, and shape of the data distribution. It is important to note that missing values (NaN values) have been excluded in order to ensure an accurate representation of the dataset. The described statistics include the mean, count, standard deviation, percentiles, minimum and maximum values of all the features. These statistics provide valuable insights into the dataset, including the average values, the range of values, and the distribution of the data. This is a typical step before starting to treat a dataset. For instance, column *count* has different values per feature due to the fact that counts the non-NaN values. Hence, one can observe missing data on the weather features since their *counts* are lower than the other features. Particularly, NaN values represent about 1.3% of the total size of the dataset, they are only present on weather data. Finally, *snow_depth* has been removed as a feature due to the fact that has no statistical significance, all its statistical measures are zero.

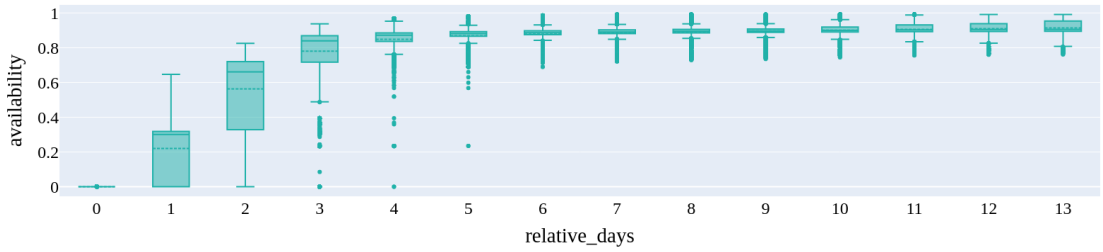


Figure 3.3: Availability vs days after the prediction time.

Figure 3.3 shows the evolution of the *availability* rate on the days after the prediction time, denoted as *relative_days*. At prediction time, when *relative_hours* is zero, consumption measures of CUPS are not stored in the system so that the aggregated value, the *total_consumption* is zero. Day by day the *availability* increases, and so the curve is also increasing with the *relative_days*. According to the plot it takes about a week to have almost the complete set of CUPS registered. Note that the 100% of *availability* is never achieved but 99.5% at most.

This fact influences a lot the possibility of building a predictive model to forecast *total_consumption* on the "prediction time", also called "execution time", due to the fact that is the time when the script with the predictor executes the forecasts. The problem is that when the prediction needs to be computed, the most recent information of the target, *total_consumption*, is not available or not valid because it is not completed. This implies that the prediction model should be able to understand the relation between *total_consumption*

vs. *availability* which it is not an easy task. Another possibility is to remove the most recent data, which is not complete, with low *availability*, and predict the future assuming this lag between last input time and first prediction time. Other strategies can be applied, this is discussed later. For the moment, to be able to continue with the data analysis and to simplify the time series analysis: rows with the *availability* lower than 94% are removed. Note that this filter is not affecting only on most recent data, the oldest one is also affected, i.e. after the filter the time series range from November 2021 till November 2022.

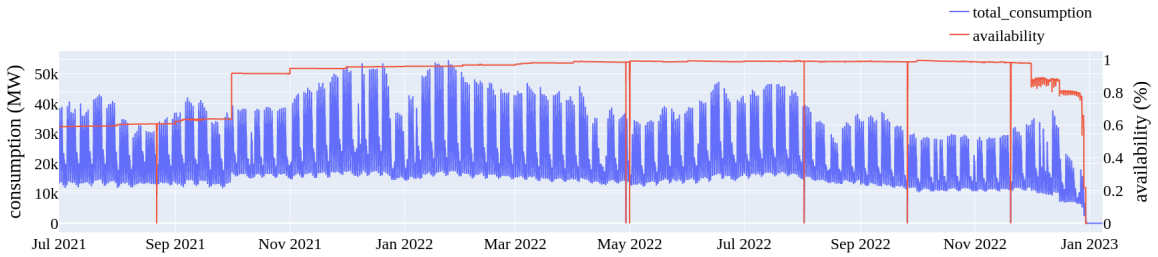
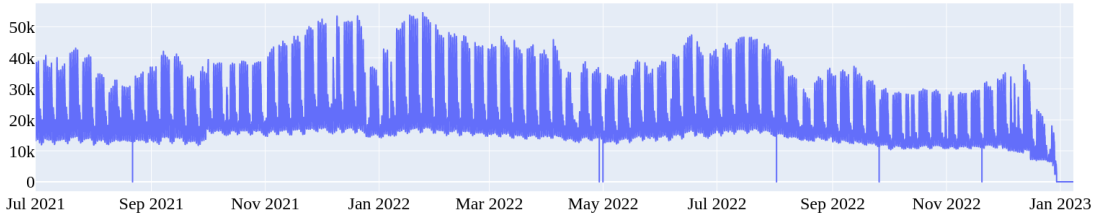


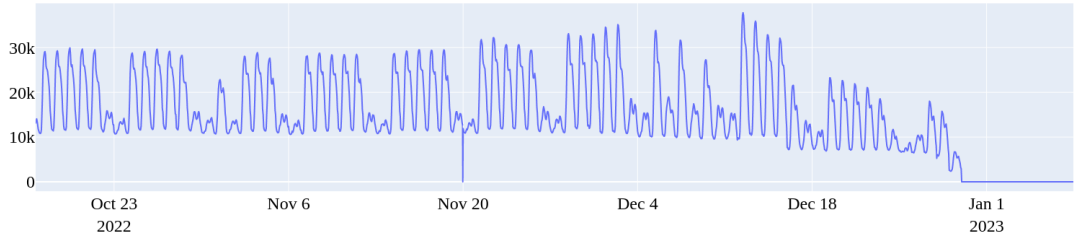
Figure 3.4: Plot of the time series data of *total_consumption* and *availability* overlapped.

Figure 3.4 shows the time series data of the *total_consumption* and *availability* overlapped in time. In there, it is easy to see the window of time with *availability* greater or equal than 94%. Figure 3.5 shows the time series of *total_consumption* before and after applying the filter. The difference between (a)-(b) and between (c)-(d) is there is a zoom in applied. To conclude, (a) is the complete raw time series, (b) is a zoom in on (a), (c) is the time series after applying the filter and (d) is a zoom in on (c).

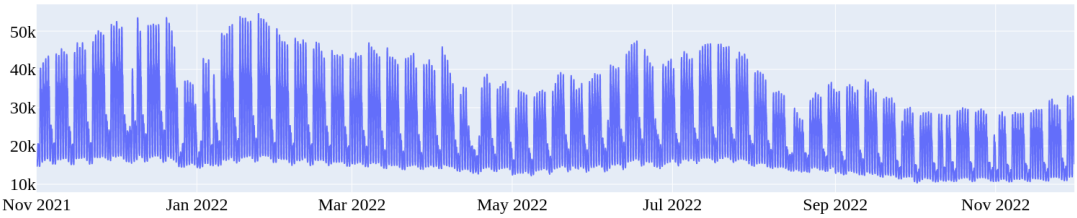
Later in Figure 3.5 (a) shows the time series of the target, *total_consumption*. One can observe that at the end of the time series the value decays. The reason is that the consumption of each CUPS is not in the data system immediately after its occurrence, it takes some days to be uploaded. Hence, the aggregated value of this variable is growing every day until all CUPS information is registered in the data system.



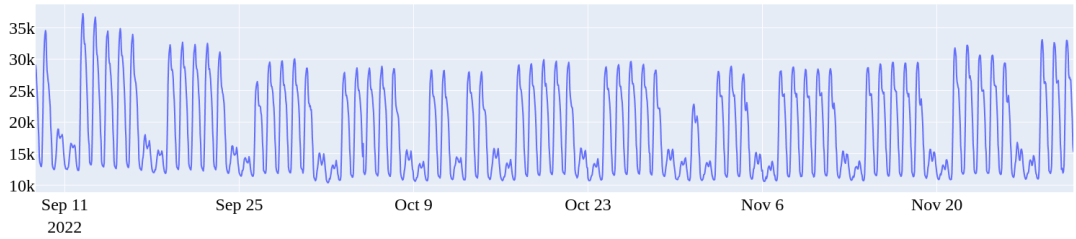
(a)



(b)



(c)



(d)

Figure 3.5: Time series: (a) raw dataset (b) zoom in (c) dataset filtered by *availability* > 0.94 (d) zoom in.

3.2 Specific Data Preprocessing

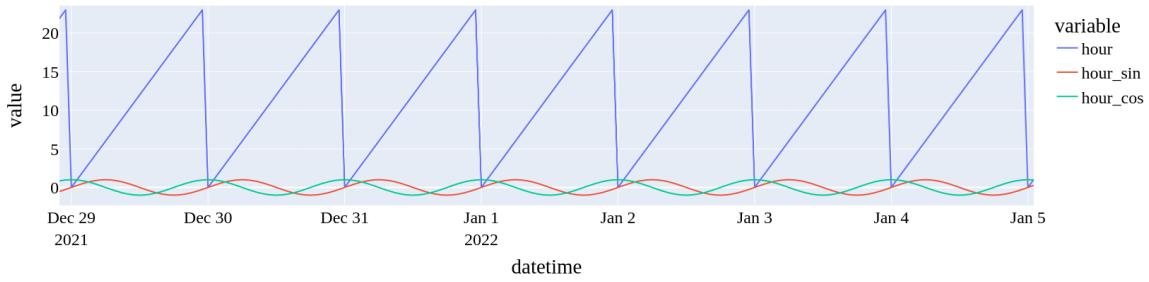
Cyclic time features, such as the day of the week for example, can be reformulated in a way that allows them to be effectively used as inputs for deep learning models. One common approach is to use a technique called ”feature encoding”.

Encoding is the process of converting data from one format into another. In the context of machine learning and deep learning, encoding is often used to convert categorical data, such as text or dates, into a numerical format that can be used as input for a model.

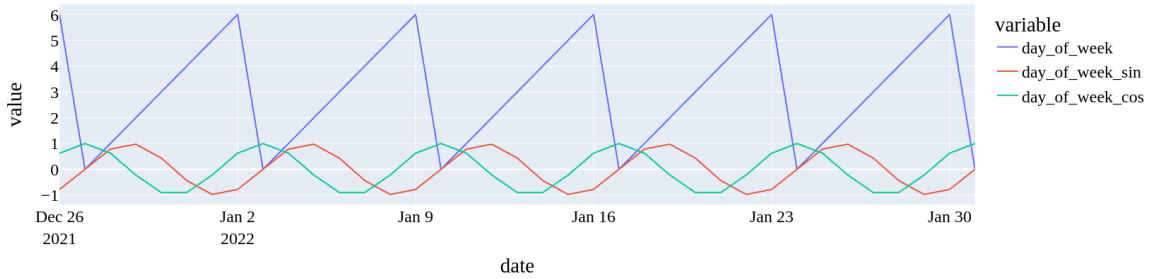
There are several different types of encoding methods, each with its own strengths and weaknesses depending on the specific characteristics of the data. Some common types of encoding include:

- One-hot encoding: This method converts a categorical feature with n possible values into n binary features, where each feature corresponds to a possible value of the original feature.
- Ordinal encoding: This method assigns a unique integer value to each category, such as 1, 2, 3, etc. This allows the model to understand the relative ordering of the categories.
- Count encoding: This method replaces a categorical feature with the count of how many times it appears in the dataset.
- Target encoding: This method replaces a categorical feature with the mean of the target variable for that category.
- Sine-cosine encoding: This method creates two new features, one that represents the sine and another that represents the cosine of the feature, allowing the model to capture the cyclical nature of the original feature.

Cyclic time features is use to encode via sine-cosine encoding. These new features can be used as inputs for the model, and they effectively capture the cyclical nature of the original feature while being in a range of [-1,1].



(a)



(b)

Figure 3.6: Encoded time features to cyclical with sinus and cosine: (a) *hour* and (b) *day of week*.

One method which can be used to encode cyclical features, x is to map them with sine and cosine transformations:

- $x_{sin} = \sin \dfrac{2\pi x}{\max(x)}$
- $x_{cos} = \cos \dfrac{2\pi x}{\max(x)}$

Figure 3.6 shows the result of this encoding. Raw features: *hour* and *day of week* are transformed using these formulas. For instance, *hour* is plotting the representation of their values along time but the problem is that there are jump discontinuities in the plot at the end of each day dropping from 23 to 0. With this behaviour the algorithm is not going to learn that the distance from 10 to 11 is the same than from 23h to 0h. The same happens for the *day of week* feature. In this case, the jump discontinuity happens at the end of the week from 6 dropping to 0. Therefore, using both sin and cosine transformations the cyclic behaviour can be transmitted to the models.

Accordingly, these time encoders have been added to the dataset used the presented problem. However, cyclical time features have not always been included in the dataset to train the algorithms, specially for the deep learning algorithms which already include time

feature encoders, a deeper explanation is given on Section 5.1. Therefore, sometimes time features have been included and other they value not. The time features that have been cyclical encoded on purpose on the dataset are the following:

Raw time feature	Cyclical encoding
`hour`	`hour_sin`, `hour_cos`
`day_of_week`	`day_of_week_sin`, `day_of_week_cos`
`day`	`day_sin`, `day_cos`
`month`	`month_sin`, `month_cos`
`year`	None

3.3 Exploratory Data Analysis

Exploratory data analysis (EDA) is an important step in the data analysis process as it allows you to gain a deeper understanding of the data and its underlying structure. There are several reasons why EDA is important:

- Identifying patterns and relationships: EDA can help to reveal patterns and relationships within the data that might not be immediately apparent. This can lead to new insights and hypotheses about the problem at hand, which can then be tested using statistical or machine learning methods.
- Detecting outliers and anomalies: EDA can help to identify outliers and anomalies in the data, which can have a significant impact on the performance of a model. By identifying and addressing these issues early on, you can improve the overall accuracy and reliability of your analysis.
- Assessing the quality of the data: EDA can help to assess the quality of the data and identify any issues, such as missing values or inaccuracies, that may need to be addressed before modeling.
- Preparing the data for modeling: EDA can help to prepare the data for modeling by identifying appropriate input features, transforming the data to meet the assumptions of the model, and dealing with missing or outliers.
- Communicating results effectively: EDA can help to communicate the results of the analysis effectively by providing visualizations and summaries of the data that can be easily understood by a wide audience.

Overall, EDA is an essential step in the data analysis process as it allows you to gain a deeper understanding of the data and its underlying structure, which can lead to more accurate and reliable results. It also makes the process of identifying patterns, relationships, and outliers more efficient, and allows you to prepare the data for modeling while giving a clear idea of the quality of the data.

3.3.1 Patterns Identification

Histograms are a powerful tool for EDA as they provide a visual representation of the distribution of the data and allow for the easy identification of patterns, outliers, and other important characteristics of the data. They can also provide information about central tendency, dispersion, and skewness of the data, making them a valuable tool for understanding and analyzing the data.

Figure 3.7 presents the histograms of each feature in the raw dataset (excluding the time related ones). A histogram is plot that shows the distribution of a numerical variable with multiple bars. Each bar represent a range of the variable values, in general called bin, with a height proportional to the frequency of points in the dataset with values within the corresponding bin. In this sense, they can be called frequency distribution plots.

The shape of the distribution plot shows where the most common values (one peak) if the data is skewed to one side or if it is symmetric, if it is bimodal (two peaks), multimodal (several peaks) etc. Figure 3.7 shows different behaviours in each feature. For instance, *heat_index*, *wind_speed* and *total_consumption* are apparently unimodal, while *demand*, *relative_humidity*, *temperature*, *temperature_min*, *temperature_max* and *windchill* are clearly bimodal. The rest are not clear.

Sometimes, bimodal distributions are actually two different unimodal mixed. Usually, this could mean that there are groups in the data. Moreover, two peaks can be due to sinusoidal patterns in the data, i.e. data might be following a seasonal pattern. In order to check the possibility of this last fact Figure 3.8 shows the box plots of *total_consumption*, which is the most important feature, grouped by *hour* and by *day_of_week*. Note that *day_of_week* values corresponds to Monday as the ”0” till Sunday as the ”6”. It can bee seen that boxplots are changing in each *hour* and *day_of_week* thus this can cause the multimodality in the frequency distribution of *total_consumption* without grouping. Furthermore, these distributions suggest that there exist variability of *total_consumption* along *hour* and *day_of_week*.

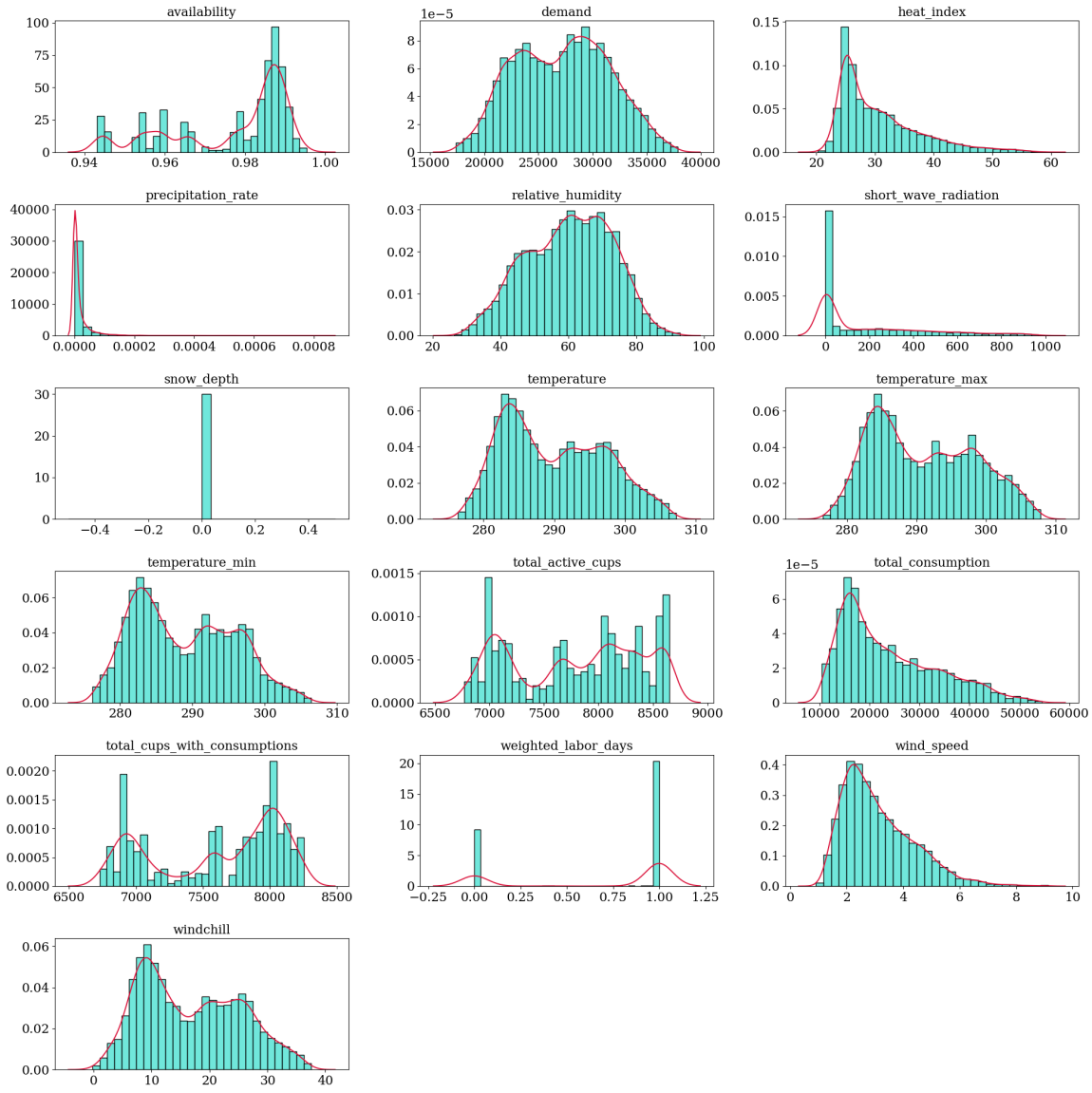


Figure 3.7: Features histogram and density function.

3.3.2 Outliers Detection

Outliers are data points that fall significantly outside of the expected range of values for a dataset. They are unusual observations that can have a significant impact on the performance of a model.

Outliers can occur for a variety of reasons, such as measurement errors, data entry errors, or even genuine extreme observations. They can be caused by mistakes, errors or simply by natural variability in the data. Outlier detection is important for several reasons:

- Model performance: Outliers can have a significant impact on the performance of a

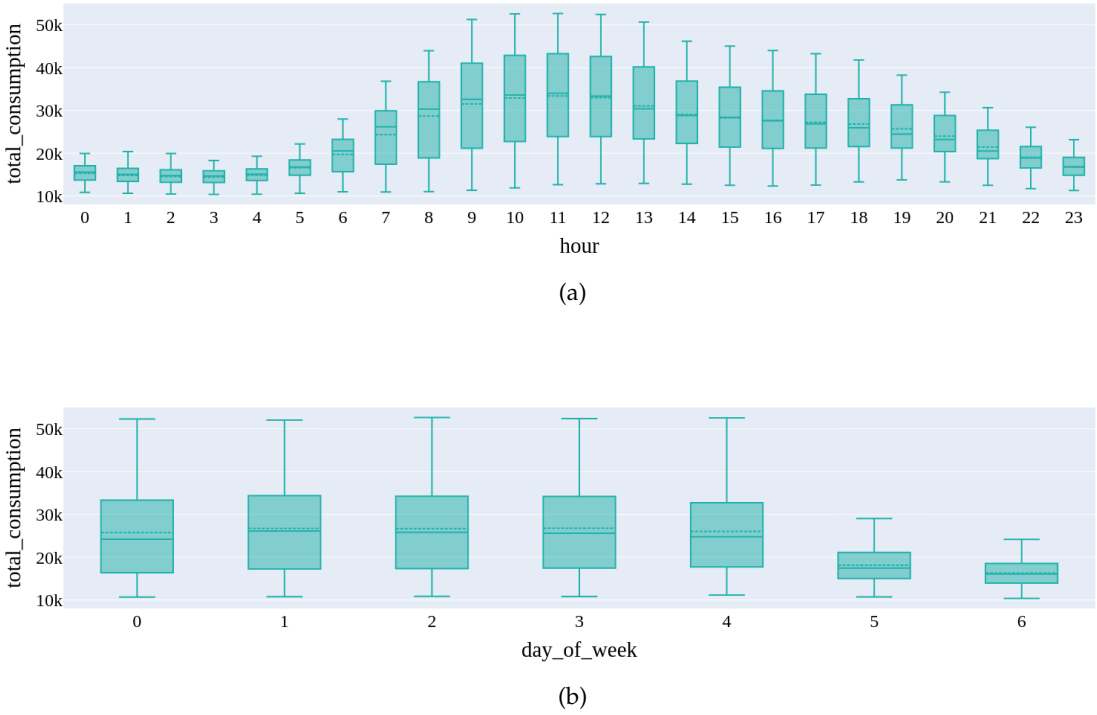


Figure 3.8: Distribution of *total_consumption* per (a) *hour* and per (b) *day_of_week*.

model. They can skew the results and lead to inaccurate or unreliable predictions. By identifying and addressing outliers, the performance of the model can be improved.

- Data Quality: Outliers can indicate errors or inaccuracies in the data, such as measurement errors or data entry errors. Identifying outliers can help to identify and correct these issues and improve the overall quality of the data.

- Data Understanding: Outliers can provide valuable insights into the data and the underlying processes that generated it. They can reveal patterns or relationships that would not be apparent from the data alone.

- Cost: Outliers can have a significant impact on the costs of any process that relies on the data, such as manufacturing, finance, and insurance. Identifying outliers can help to minimize costs and improve efficiency.

- Business Decisions: Outliers can also be very important in business decisions, they can be a sign of a problem in the business, such as a fraud, in this case, it is crucial to detect them as soon as possible.

There are several techniques that can be used to identify outliers in a dataset, some of the most common include:

- Visualization techniques: Visualization techniques such as box plots, scatter plots, and histograms can be used to identify outliers in a dataset. These techniques can help to identify observations that fall outside of the expected range of values and can be easily identified as dots outside of the general data distribution.

- Statistical methods: Statistical methods such as the Z-score and the Interquartile Range (IQR) method can be used to identify outliers. The Z-score method calculates the number of standard deviations from the mean a data point is. The IQR method calculates the range between the first and the third quartile, any data point that falls outside of this range can be considered an outlier.

- Distance-based method: Distance-based methods such as the Local Outlier Factor (LOF) and the DBSCAN algorithm can be used to identify outliers. These methods use distance metrics to identify observations that are significantly different from the rest of the data.

- Clustering: Clustering algorithms such as k-means can be used to identify outliers by identifying observations that do not belong to any cluster.

- Machine Learning: Some machine learning algorithms such as Isolation Forest and One-class SVM are designed to detect outliers. They can be trained on the data to identify observations that are significantly different from the rest of the data.

In this work *Interquartile Range* (IQR) method was used because of its advantages :

- Robustness: The IQR method is considered to be a robust method of outlier detection, as it is not affected by the presence of extreme values in the data. This makes it a suitable method for datasets with skewed distributions according to [Tukey 1977].

- Simplicity: The IQR method is relatively simple to implement, as it only requires calculations of the first and third quartiles (Q1 and Q3) of the data. This makes it easy to understand and interpret.

- Flexibility: The IQR method can be used for both univariate and multivariate data, making it a flexible method for outlier detection.

- Consistency: The IQR method is consistent, meaning that the same result will be obtained regardless of the dataset.
- Minimizes the effect of outliers: The IQR method is based on the range between the first and the third quartile, which minimizes the effect of outliers on the detection process.
- Easy to interpret: The IQR method provides a clear and easy to interpret measure of the spread of the data.

The Interquartile Range is a measure of the dispersion of a dataset. It is defined as the difference between the 75th and 25th percentiles of a dataset. The 75th percentile is also known as the third quartile ($Q3$), and the 25th percentile is known as the first quartile ($Q1$). It is often used as a measure of spread in box plots and to identify outliers in a dataset. The formula of Interquantile Range is:

$$IQR = Q3 - Q1$$

Then the *decision range* is:

- Lower Bound $= Q1 - 1.5 \cdot IQR$
- Upper Bound $= Q3 + 1.5 \cdot IQR$

Data points outside the Lower and Upper Boundaries are removed from the dataset.

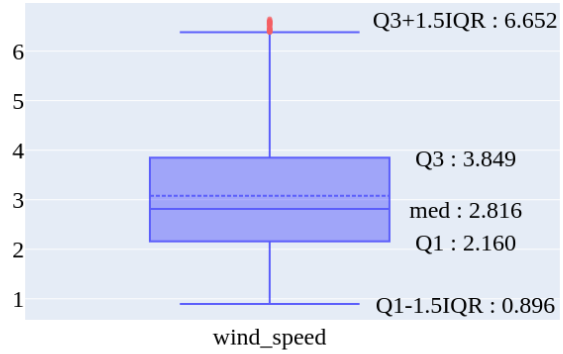


Figure 3.9: Outliers of *wind_speed*.

This method has been applied to the dataset features, separately, in order to find outliers. Due to the strong seasonal behaviour that data presents on `hour` and `day_of_week`,

these two features have been used for the outlier detection. Hence, the IQR method has been applied on segments of data defined by each `hour` and `day_of_week` combinations. First, filtering the dataset by each feature combination. Then, calculating the lower and upper bounds to find the outliers and, finally, set them to NaN. At the end, the percentage of outliers per feature has been between 1-2% the size of the dataset.

Figure 3.9 shows an example of the outliers detected in *wind_speed*. The plot shows a box where the bottom height defines the $Q1$ and the upper height the $Q3$, i.e. the box height defines the IQR of the feature. Then the whiskers, extremes outside the box, delimit the lower and upper bounds of the *decision range*. Outliers are marked in red, which are the points outside the decision range.

3.3.3 Handling Missing Values

Handling missing values is an important step in the data analysis process, as missing values can have a significant impact on the performance of a model. There are several techniques that can be used to handle missing values, including:

- Deleting rows or columns: This technique involves removing rows or columns that contain missing values. This method is simple and easy to implement, but it can lead to a loss of important information if too many observations are removed.
- Imputation: This technique involves replacing missing values with estimates based on the available data. Common imputation methods include mean imputation, median imputation, and multiple imputation.
- Interpolation: This technique involves estimating missing values based on the values of other observations in the dataset. Linear interpolation and spline interpolation are common interpolation methods.
- Prediction: This technique involves using machine learning algorithms to predict missing values based on the available data.
- Data augmentation: This technique involves creating new data based on the existing data to fill in missing values.

The choice of technique will depend on the specific characteristics of the data and the problem at hand. In our case if there is a substantially amount of missing data at the beginning and/or at the end of the time series we just trim it to diminish NaN values. However, we have a substantial amount of missing values concentrated in weather data

at specific days of the time series, from the 5th till the 12th of April. Interpolation method should be used in this case. Since it is a large period of time the model which imputes must be something more complex than a simple linear interpolation. In this case *K-Nearest Neighbours*, which is an Unsupervised Model, is implemented as the imputer model.

K-nearest neighbors (KNN) is a popular machine learning algorithm that can also be used as an interpolation method to handle missing values. The basic idea behind KNN is that an observation's value can be estimated based on the values of its 'k' closest neighbors.

The steps to use KNN for interpolation are:

- Identify the k-nearest neighbors for each observation with missing values.
- Use the values of the k-nearest neighbors to estimate the missing value.
- There are different ways to estimate the missing value, one way is to take the average of the values of the k-nearest neighbors.

One of the advantages of using KNN as an interpolation method is that it is non-parametric, meaning it makes no assumptions about the underlying distribution of the data. This makes it suitable for datasets with complex or non-linear relationships. Additionally, KNN can also be used for multivariate data, allowing to estimate missing values based on multiple features.

However, it's worth noting that KNN is sensitive to the choice of k, the number of nearest neighbors used to estimate the missing values. A small value of k will lead to high variance in the estimates, while a large value of k will lead to high bias. Therefore, it's important to choose the appropriate value of k for your dataset.

It is also important to note that KNN is a computationally intensive method, and it may not be suitable for very large datasets.

Due to the strong seasonal behaviour that data presents on `hour` and `day_of_week`, K-Nearest Neighbours has been applied per each combination of these to features. First, the combinations have been pre-computed, then, data has been filtered per each of them. The percentage of imputed values per feature has been between 3-4% the size of the dataset.

Figure 3.10 shows an example of the imputation result carried out with the *temperature*. The time series with missing values of the variable is plotted in red. Then, time series in blue is the time series after applying the imputation technique. Note that since both time

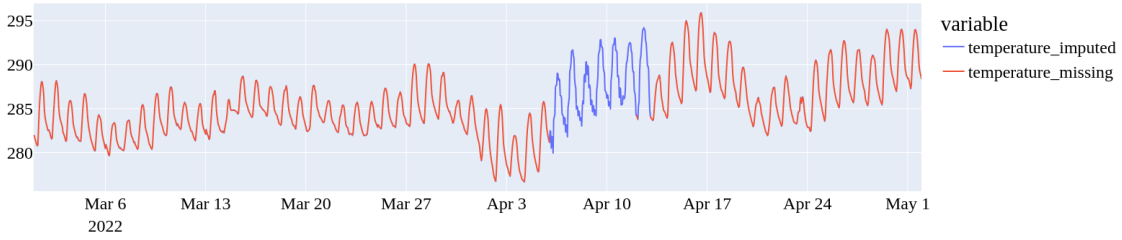


Figure 3.10: Example of Imputation.

series are overlapped the complete time series is not visible, but just the important part which are the imputed values.

3.3.4 Data Correlation

Data correlation refers to the degree to which two or more variables are related to each other. It is an important concept in data analysis as it can provide valuable insights into the relationships between different variables in a dataset.

There are several ways to measure data correlation, including:

- Pearson correlation coefficient: This is a measure of the linear relationship between two variables. It ranges from -1 to 1, where -1 indicates a perfect negative linear relationship, 0 indicates no relationship, and 1 indicates a perfect positive linear relationship.
- Spearman rank correlation: This is a non-parametric measure of the monotonic relationship between two variables. It ranges from -1 to 1, where -1 indicates a perfect negative monotonic relationship, 0 indicates no relationship, and 1 indicates a perfect positive monotonic relationship.
- Kendall's tau: This is also a non-parametric measure of the monotonic relationship between two variables. It ranges from -1 to 1, where -1 indicates a perfect negative monotonic relationship, 0 indicates no relationship, and 1 indicates a perfect positive monotonic relationship.
- Mutual Information: This is a measure of the dependence between two variables, it can be used for both continuous and discrete variables.

Data correlation can provide valuable insights into the relationships between different variables in a dataset. By understanding the correlation between variables, you can iden-

tify which variables are most important for explaining the variation in the data and which variables may be redundant or irrelevant. This information can be used to improve the performance of a model and make more informed decisions.

It's also worth noting that correlation doesn't imply causation, meaning that just because two variables are correlated doesn't mean that one variable causes the other. It's important to use other techniques, such as causal inference methods, to establish causality.

We use *Pearson Coefficient* ρ to measure a correlation of variable X respect to Y:

$$\rho = \frac{cov(X,Y)}{\sigma(X)\sigma(Y)}$$

where:

cov is the covariance
$\sigma(X)$ is the standard deviation of X
$\sigma(Y)$ is the standard deviation of Y

The formula for covariance is:

$$cov(X,Y) = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

Standard deviation is given by:

$$\sigma(X) = \sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$
$$\sigma(Y) = \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Which gives the Pearson correlation coefficient as:

$$\rho(X,Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where n is the sample size. Correlation coefficient values range from -1 to +1. The sign indicates the direction of the correlation and the absolute value defines the magnitude. The correlation matrix is traditionally used to show multiple variables correlations in a single view. Each element in the matrix gives the correlation of the element in the row, X, with the element in the column Y.

Figure 3.12 gives a representation of the correlation matrix as a heat map plot. A heat map plot is a data visualization technique to show the magnitude of a measure as a color in two dimensions. In this case the color indicates the magnitude of the absolute correlation between X and Y. The more intensive is the color the more correlation exists. Dark purple

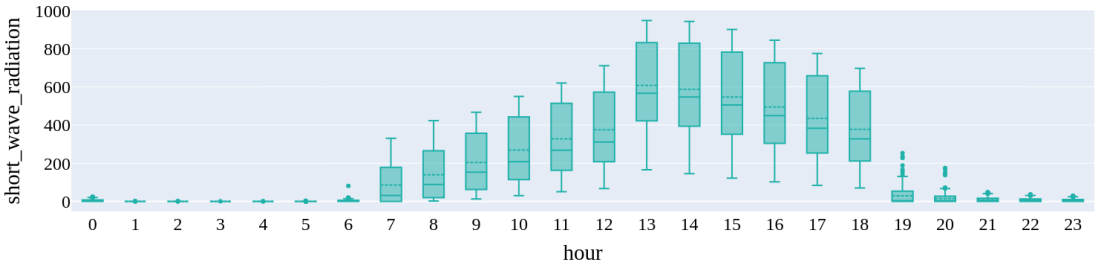


Figure 3.11: Sort wave radiation distribution per hour of day.

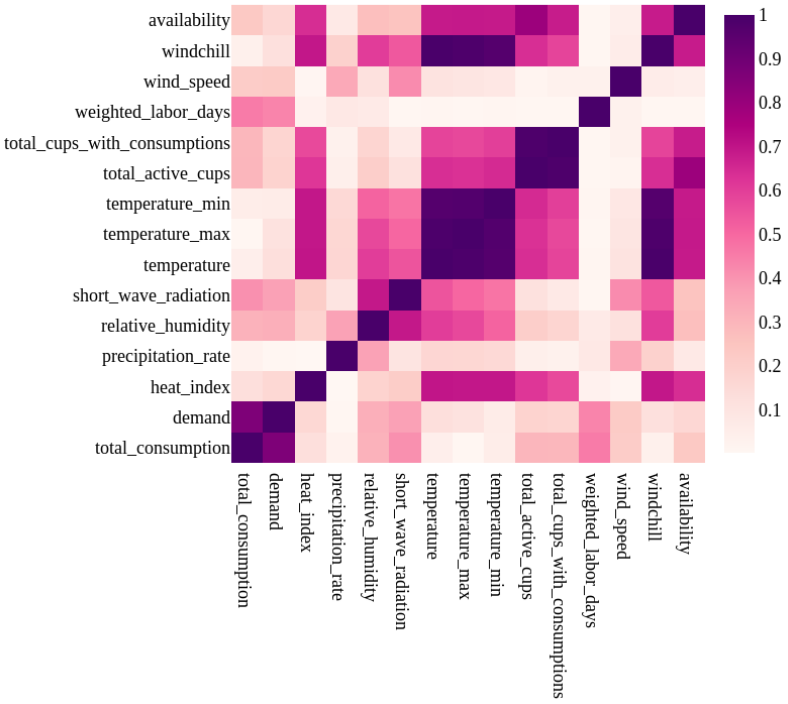


Figure 3.12: Correlation Matrix.

is the equivalent to $|\rho(X,Y)| = 1$ and the lighter it gets arriving to almost white, it is equivalent to $|\rho(X,Y)| = 0$.

Moreover, Figure 3.12 shows that some features are highly related with other. Of course, some high correlation is already expected between *temperature*, *temperature_min* and *temperature_max*, between *temperature* and *heat_index* or between *temperature* and *windchill*. While others are quite surprising such as *total_active_cups* or *total_cups_with_consumptions* with respect to the *heat_index*. However, most important correlations are those related with the *total_consumption*, the target.

Feature	Correlation (%)
demand	0.86
weighted_labor_days	0.45
short_wave_radiation	0.41
relative_humidity	0.31
total_active_cups	0.30
total_cups_with_consumptions	0.29
availability	0.23
wind_speed	0.21
heat_index	0.13
temperature_min	0.05
temperature	0.05
windchill	0.04
precipitation_rate	0.03
temperature_max	0.01

Table 3.2: Correlation in percentage between the candidate predictors and the target value. Source: Own

Table 3.2 shows the absolute values of the correlations between *total_consumption* and the other variables in descending order. As one could expect, *demand* is the most correlated variable due to the fact that this is prediction of Red Eléctrica for this the *total_consumption*. Then it comes the *weighted_labor_days* which have sense because on holidays many industries close. Following it, the *short_wave_radiation* which is to some extend related with the hours of sun is also in a high position in the rank of correlation because when the working hours usually starts with the sunrise and ends on the sunset. In fact, Figure 3.11 shows the distribution of the *short_wave_radiation* along *hour*, there is a clear similarity or relation with Figure 3.8 (a). The rest of correlations are also meaningful but not as strong.

In [Krehbiel 2004] statistically demonstrates that the if the correlation coefficient is grater than $\frac{2}{\sqrt{sample\_size}}$ then there exist linear relationship. In this dataset, with a sample size of 9,489 a correlation greater than 0.02 is enough to say that there is a linear relationship. To conclude, all features in the dataset can be denoted as linearly related with the *total_consumption* except from *precipitation_rate* and *temperature_max*.

3.3.5 Stationarity and Seasonality Hypothesis

Stationarity and seasonality are important concepts in time series analysis, as they can have a significant impact on the performance of a model, Section 2.1.1 give detailed explanations on these concepts.

Stationarity refers to the property of a time series where the statistical properties (such as the mean and variance) remain constant over time. A stationary time series has a constant mean, variance and autocovariance, which is important for a lot of time series models and techniques. Non-stationary time series are more difficult to model and analyze, because their statistical properties change over time.

Seasonality refers to the presence of regular and predictable patterns in a time series that repeat over a specific time interval, such as daily, weekly, or yearly. Seasonality can have a significant impact on the performance of a model, as it can introduce bias and increase the variance of the predictions.

It is important to identify and address both stationarity and seasonality in a time series dataset to ensure that the model is able to make accurate predictions. There are several techniques that can be used to address these issues, such as differencing, which can be used to make a time series stationary, and seasonal decomposition, which can be used to identify and remove seasonality from a time series.

Additionally, if the time series is not stationary, the first step is usually to make it stationary, and after that, the seasonality can be handled. Some techniques such as ARIMA and Prophet are designed to handle both stationarity and seasonality, but in other techniques it's necessary to make the series stationary before handling seasonality.

3.3.5.1 Stationarity

Statistical tests make strong assumptions about the data and they are able to set a degree to which a null hypothesis could be rejected or not. A unit root test to statistically test the stationarity of a time series is the *Augmented Dickey-Fuller test*.

The Augmented Dickey-Fuller (ADF) test is a statistical test that can be used to determine whether a time series is stationary or non-stationary. It is a widely used test in time series analysis and econometrics.

The ADF test is based on the idea that a non-stationary time series can be made stationary by differencing it a sufficient number of times. The test statistic measures the difference between the sample mean of the differenced series and the mean of the original series. The null hypothesis of the ADF test is that the time series is non-stationary, and the alternative hypothesis is that the time series is stationary.

The steps of the ADF test are:

- Determine the level of differencing needed to make the time series stationary.
- Estimate the parameters of the regression model including the constant term, the trend term (if needed) and the error term.
- Calculate the test statistic and compare it to the critical values from the ADF distribution table.
- Based on the calculated test statistic and the critical values from the ADF distribution table, reject or fail to reject the null hypothesis.

If the calculated test statistic is less than the critical value from the ADF distribution table, then the null hypothesis is rejected, and the time series is considered to be stationary. If the calculated test statistic is greater than the critical value, then the null hypothesis is not rejected, and the time series is considered to be non-stationary.

It is worth noting that ADF test assumes that the error term is normally distributed and it may not be suitable for all types of time series.

The *Augmented Dickey-Fuller test* applied on the hole time series of *total_consumption* gives a $p - value = 1.29$ on the whole dataset. Repeating the test for small windows such as 30 days, $p - value$ reduces to 0.2 in average. In both cases, these values do not reject the null hypothesis hence, the data is non-stationary considering this test.

The distribution of the data can reveal important patterns when grouped by specific time intervals, such as "hour of day" and "day of week", as discussed in section 3.3.1. These patterns can be seen in Figure 3.8. The plots show that there are different levels of stability in the statistical values for each "hour of day" and "day of week". Given that a time series can be stationary or non-stationary for different time intervals, as explained in Section 2.1.1, it is necessary to apply the *Augmented Dickey-Fuller test* by dividing data by "hour of day" and "day of week". The results of these tests are summarized in Table 3.3.

hour	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
ADF Statistic	-0.7	-0.6	-0.6	-0.6	-0.7	-1.0	-1.3	-2.0	-1.9	-1.9	-1.6	-1.7	-1.7	-1.7	-1.7	-1.6	-1.7	-1.6	-1.5	-1.2	-1.1	-1.1	-0.9	-1.2
p-value	0.9	0.9	0.9	0.9	0.8	0.7	0.6	0.3	0.3	0.3	0.5	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.5	0.7	0.7	0.7	0.8	0.7

day_of_week	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
ADF Statistic	-6.5	-4.9	-5.0	-4.6	-4.8	-2.4	-0.7
p-value	0.0	0.0	0.0	0.0	0.0	0.1	0.8

Table 3.3: ADF test results grouping data by day and hour.

The results of the *Augmented Dickey-Fuller test* indicate that when grouping the data by "day of week", the $p - value$ is below 0.05, which means that the null hypothesis is rejected (that this fact is true for all days of week except for Sunday, but we consider this as unrelevant). Results indicates that the data is stationary on a daily basis. On the other hand, when dividing the data by "hour", the $p - value$ is not below 0.05 and the null hypothesis is not rejected i.e. data is non-stationary. While this suggests that the data is less non-stationary, but it is not stationary. However, these $p - values$ are definite lower than when executing the test for the entire time series.

3.3.5.2 Seasonality

Autocorrelation, also known as serial correlation, is the correlation between a time series and a lagged version of itself. It measures the similarity between a value and the value that precedes or follows it in time. Hence is a measure of the relationship between the current value and values in the past. Values above zero mean positive correlation and values bellow zero negative correlation.Autocorrelation is an important concept in time series analysis, as it can have a significant impact on the performance of a model.

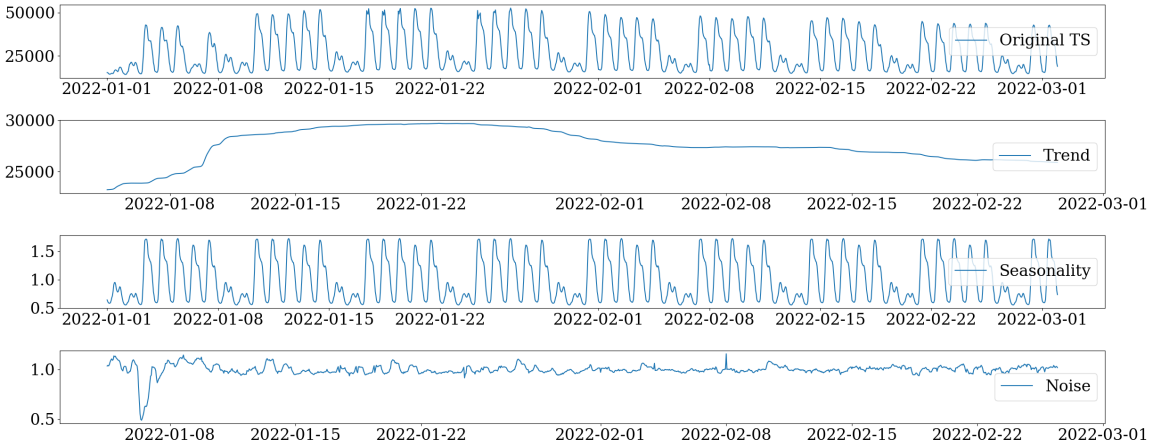


Figure 3.13: Time series decomposition plot for *total_consumption*.

There are two types of autocorrelation: positive autocorrelation and negative autocorrelation. Positive autocorrelation occurs when values that are close in time are positively correlated, meaning that if the value at time t is high, the value at time $t + 1$ is also likely to be high. Negative autocorrelation occurs when values that are close in time are negatively correlated, meaning that if the value at time t is high, the value at time $t + 1$ is likely to be low.

Autocorrelation is useful to uncover hidden patterns in the data and to help tuning

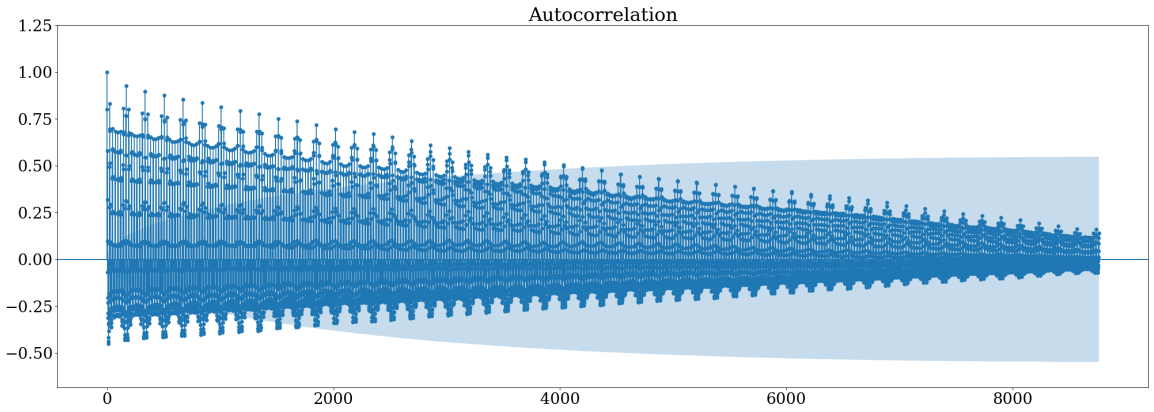


Figure 3.14: Autocorrelation plot for *total_consumption*.

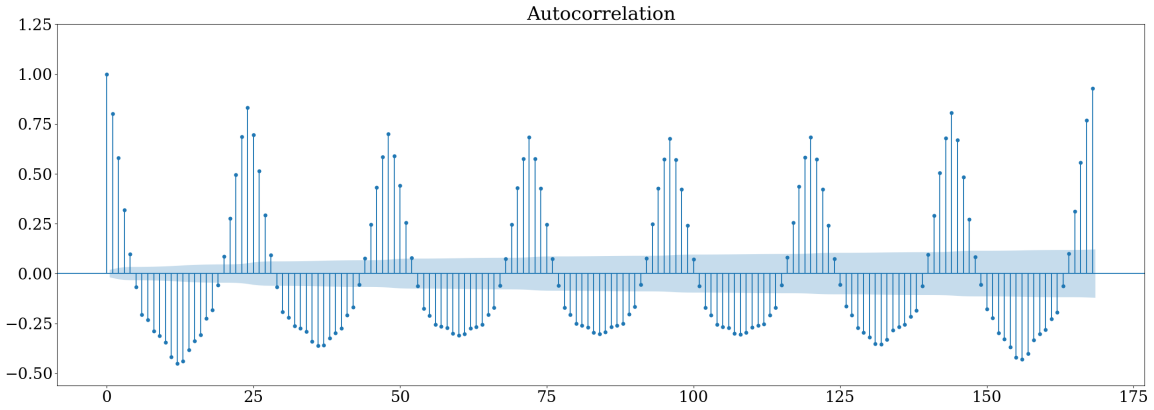


Figure 3.15: Autocorrelation plot limiting lags up to the total hours in a week, for *total_consumption*.

the forecasting methods. It is also useful to identify seasonality in the time series. Figure 3.14 represents the autocorrelation function plot (ACF plot) of *total_consumption*. The ACF plot is used to measure the correlation between a time series and a lagged version of itself. It helps to identify the number of autoregressive (AR) terms (p) in an ARIMA model, see Section 2.2.1.3 for more details. The ACF plot shows the correlation between the time series and its lags, and if there is a significant correlation between the time series and its lags, it is likely that the time series is autoregressive, and an AR term should be included in the model. In other words, if the autocorrelation of a time series with its lags is high, it means that there is a pattern that is repeating itself at regular intervals, which is an indication of the presence of an autoregressive term in the model.

In Figure 3.14, the shaded blue area on either side of the abscissas represent the 95% confidence intervals, and any correlation value that falls outside of these lines is considered statistically significant. It can be seen that as the lag increases, the correlation between the time series and its lags decreases. The plot shows that there is a significant positive cor-

relation between the time series and the 4000 first hours, and negative correlation between time and the first 2000 hours, approximately. Then Figure 3.15 is the same plot but limited to the firsts 7 days. Repeated correlation patter indicates that there is likely a pattern in the time series that is repeating itself at regular intervals, in this case in 24h.

Finally, Figure 3.13 illustrates the decomposition of the *total_consumption* over time, noting that it only displays a specific period of the entire time series. This is a method known as "seasonal decomposition" which separates the time series into its trend, seasonal component and noise, as previously discussed in Section 2.1.1. The figure evidently displays a seasonal component, characterized by a repeating pattern of 24 hours per day. Additionally, a trend component is visible, which may contribute to the non-stationary behavior previously discussed in Section 3.3.5.1.

3.4 EDA Insights

Here is the most important insights that are gained from EDA:

- Distribution of data: The original data indicates that certain features of the dataset have unimodal and multimodal distributions, which may suggest the presence of groups or seasonal patterns in the data.
- Relationships between variables: To investigate the relationships between variables, Pearson's coefficient is calculated for each pair of variables in the dataset to reveal the correlation between the target value in *total_consumption* and the other features. The analysis shows that almost all features have a linear relationship, with the exception of two. It is worth noting that a strong relationship is found between the dataset features and ”hour of day” and ”day of week”.
- Missing data: The original dataset does not contain any missing data, with the exception of weather data for a specific time interval, one week in April 2022.
- Outliers: Outliers are present in the dataset, but they represent less than 1% of the data. These outliers were detected using the IQR method for each feature and were subsequently removed from the dataset.
- Imputation of missing data: Imputation of the missing data was necessary because the forecasting algorithms require it. To accomplish this, the dataset was divided according to ”hour of day” and ”day of week” and the KNN method was applied for each feature.
- Stationarity: To detect stationarity in the target time series, the Augmented Dickey-Fuller test was used. The results of the test indicate that there is no stationarity in the target time series overall, but by separating the time series by ”day of week” it is discovered that daily stationarity exists.
- Seasonality: To analyze seasonality, autocorrelation plots were drawn. This plot shows strong autocorrelation between the time series and the lags, with autocorrelation decreasing as the lags increase. The autocorrelation plot reveals a pattern that suggests the presence of a 24-hour seasonal pattern.

Chapter 4

Model Enhancement

This chapter aims to detail the research conducted to enhance the current Holaluz algorithm for customer consumption forecasting. In pursuit of this goal, various algorithms were trained using different configurations in order to identify the optimal model. A brief description of the model implementations is provided in the Section 4.2, 4.3, 4.4 and 4.5.

The complete dataset was divided into three sets: a train set, a validation set, and a test set. As previously described in Section 2.3.2. Two types of splits are applied, Table 4.1 shows their characteristics.

Table 4.1: The tables represents the split of the time series dataset. Column shuffled means that the dataset has been randomly shufled, hence the second table indicates that train and validation dataset are complementary, starting and ending on same date and that rows are shufled.

dataset split	start_date	end_date	portion (∼%)	shuffled
train	00:00h November 8th, 2022	23:00h August 10th, 202	70	False
validation	00:00h August 11th, 2022	23:00h October 8th, 2022	15	False
test	00:00h October 9th, 2022	23:00h November 30th, 2022	15	False

dataset split	start_date	end_date	portion (∼%)	shuffled
train	00:00h November 8th, 2022	23:00h October 8th, 2022	70	True
validation	00:00h November 8th, 2022	23:00h October 8th, 2022	15	True
test	00:00h October 9th, 2022	23:00h November 30th, 2022	15	False

This study contains implementation of 4 algorithms with different architecture to find an optimal forecasting algorithm:

- two ML algorithms: Prophet and XGBoost, and
- two DL algorithms: DeepAR and Temporal Fusion Transformer (TFT),

All of these algorithms were implemented, trained, and fine-tuned utilizing the Amazon SageMaker software development kit (SDK). Specifically, the DeepAR and XGBoost models were implemented utilizing the *built-in models* provided by Amazon SageMaker. On the other hand, the TFT and Prophet models, which originate from the GluonTS package, were implemented utilizing the SageMaker MXNet, which is a *pre-made image* specifically designed for ML and DL models. Section 4.1 describes in more details what are *built-in models* and *pre-made images*.

Therefore, the code to develop the models was written in Python by using Jupyter Notebooks. The training and tuning tasks were implemented using the Amazon SageMaker Python SDK, which facilitates interaction with SageMaker from within Python code as an API, Figure 4.1 helps to understand this process. The training and tuning process for each model was developed locally, but executed on AWS instances. This approach allowed for faster completion of the tasks, as compared to running the process on a common laptop with 16GB of RAM. Each model was trained with multiple dataset configurations regard-

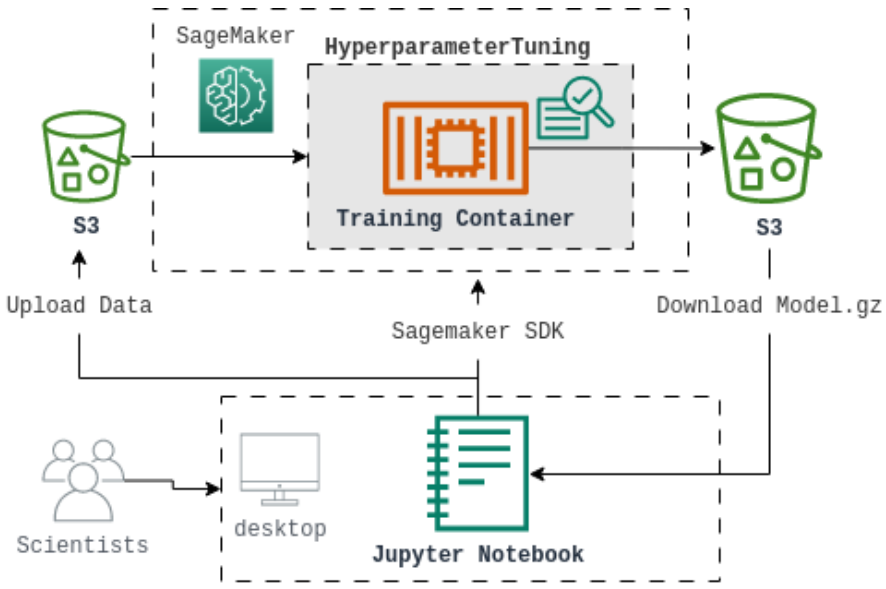


Figure 4.1: Interaction using Sagemaker SDK.

ing the dataset features and the order of the training dataset rows. Section 5.1 defines each model setup. The names of the trained models are codified according to their characteristics.

To carry out the training and tuning tasks, the train set was utilized tot train the models and the validation set to calculate the Hyperparameter optimization metrics. Hyperparameter tuning was applied to each algorithm to obtain the best possible model. It should be noted that failure to tune the hyperparameters may result in suboptimal performance,

however, certain hyperparameters may not be modifiable. The hyperparameters of the models, as well as the domains for the optimization process, will be described in the following sections.

After having all the desired models trained and tuned, to finish the process, the test set was used to evaluate the models by applying inference to it. The resulting target predictions were then compared to the original values, the observations, in the Section 5. To quantify the performance of the models, the Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE) and Mean Absolute Error (MAE) have been calculated using the predicted targets and the true values for the entire test period.

4.1 Support Framework: Sagemaker

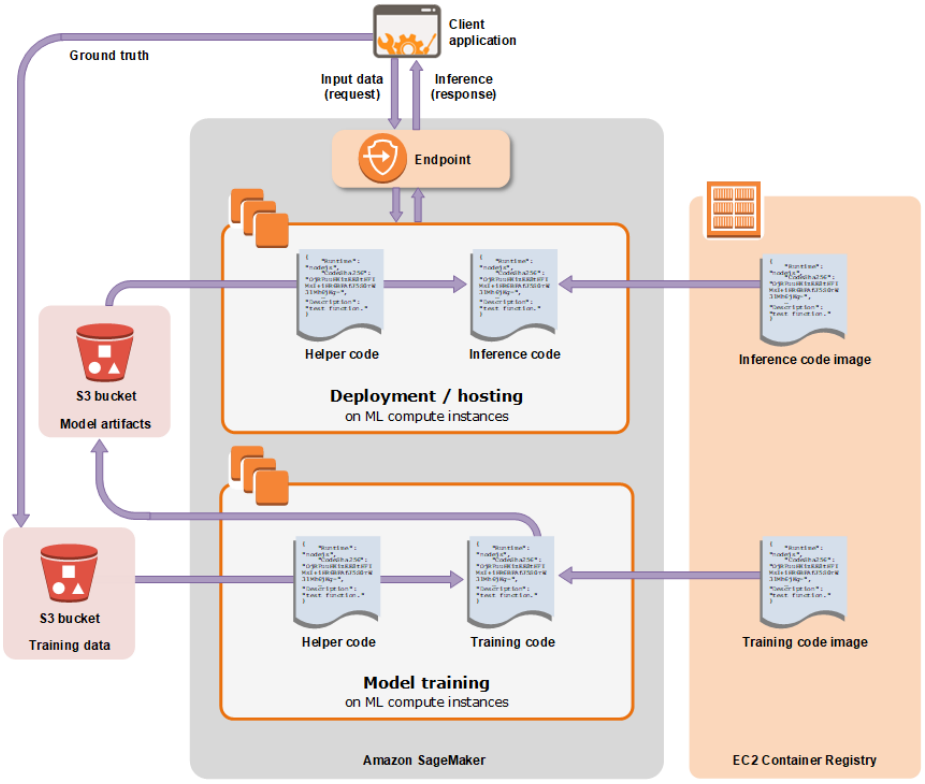


Figure 4.2: Amazon Sagemaker Generic Flow. Source: Amazon Sagemaker.

Amazon SageMaker is a fully managed platform for machine learning that provides developers and data scientists with the ability to build, train, and deploy ML models at scale, see Figure 4.2. One of the capabilities of SageMaker is model tuning, which is the process of finding the best set of hyperparameters for a machine learning model to improve its performance on a specific task or dataset. Pros of using SageMaker for developing machine learning models include:

- It is fully managed, which means that developers do not have to worry about the underlying infrastructure and can focus on the machine learning model.
- It provides a variety of built-in algorithms for hyperparameter optimization, such as Bayesian optimization and random search, which can make the tuning process more efficient.
- It allows for parallel and distributed training, which can make the tuning process faster. SageMaker provides an easy way to monitor the training process and save the best model.
- It is easy to deploy the fine-tuned model to production on SageMaker.

Cons of using SageMaker for developing machine learning models include:

- It is a proprietary platform, which means that the models and data need to be hosted on AWS.
- It may be more expensive than other open-source platforms, depending on the usage.
- The pre-trained models may not always be suitable for the specific task or domain, which can make fine-tuning more difficult.

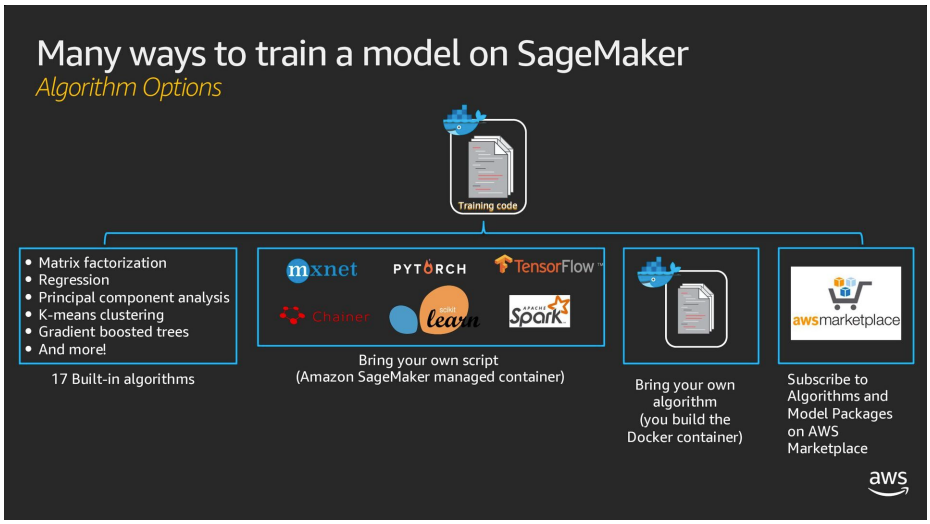


Figure 4.3: Amazon Sagemaker Training Options. Source: Amazon Sagemaker.

As mentioned before, the training and tuning tasks were implemented using the Amazon SageMaker SDK is a software development kit (SDK) that allows to interact with the SageMaker platform using a programming language of their choice, such as Python, R, or

Java. The SDK allows users to use SageMaker functionalities and features in a programmatic way, which makes it easy to automate, integrate and scale the machine learning process. This makes the SDK a great tool to use for developers and data scientists who want to use SageMaker and incorporate it into their machine learning workflows.

SageMaker facilitates several different options for training and fine-tuning algorithms. The most accessible options range from low to high levels of coding complexity and include the use of *built-in algorithms*, *pre-made images*, and *custom Docker images*. Figure 4.3 illustrates Sagemaker training and fine-tuning options.

4.1.1 Built-In algorithms

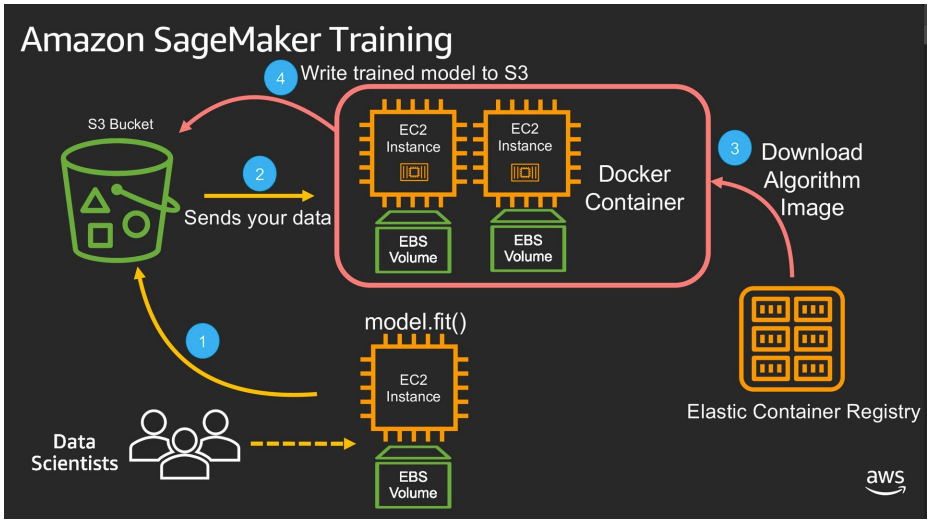


Figure 4.4: Amazon Sagemaker Buil-In Model. Source: Amazon Sagemaker.

Amazon SageMaker provides a set of built-in algorithms that can be used for training and deploying machine learning models. These algorithms are pre-built, pre-configured, and optimized for specific use cases, and can be easily accessed and used through the SageMaker platform or by code with Sagemaker SDK. Some examples of the built-in algorithms available in SageMaker include: Linear Learner, XGBoost, DeepAR, K-Means, Object2Vec, etc. These are some of the examples, available algorithms may change over time as new algorithms are added, and older ones are deprecated. These built-in algorithms are useful for developers and data scientists who want to quickly train and deploy machine learning models without having to build and configure the algorithms from scratch, see Figure 4.4

4.1.2 Pre-made images for machine learning frameworks

Amazon SageMaker also provides a set of pre-made images for popular machine and deep learning frameworks, which can be used to train and deploy models on the SageMaker

platform. These pre-made images include popular deep learning frameworks such as TensorFlow, PyTorch, and MXNet, as well as other machine learning frameworks like scikit-learn, R, and XGBoost. Pre-made images in Amazon SageMaker come equipped with a variety of additional Python packages, such as Pandas and NumPy. However, it is also possible to install any other Python package hosted on the Python Package Index (PyPI) by creating a "requirements.txt" file within the image class. Several popular machine learning frameworks, such as scikit-learn, Spark ML, MXNet, as well as more advanced frameworks such as TensorFlow and PyTorch, are also available for use within these images.

4.1.3 Custom Docker images

In Amazon SageMaker, custom Docker images are user-defined Docker images that can be used to train and deploy machine learning models. Custom Docker images can be created by building a Dockerfile, which is a script that specifies the base image, software dependencies, and other configurations for the image. Once the Dockerfile is created, it can be used to build the custom image using the Docker command line tool. The resulting image can then be pushed to a container registry, such as Amazon Elastic Container Registry (ECR), and used to train and deploy models on SageMaker.

When utilizing Amazon SageMaker, pre-made images and built-in algorithms may be sufficient for many machine learning tasks. However, in certain cases it may be necessary to use a custom Docker image. One such scenario is when the desired algorithm or framework is not included within the built-in algorithms or pre-made images offered by Amazon SageMaker. Additionally, using a custom Docker image allows for greater flexibility in terms of the packages and dependencies that are included within the image. This can be useful when working with specialized or proprietary algorithms, or when the specific versions of packages and dependencies are required. It is also useful when the researcher wants to have more control over the environment of the container and the resources that are allocated to it. This can be particularly useful when working with memory-intensive or computationally-intensive algorithms.

4.2 Prophet Launch

Prophet is an open-source forecasting library developed by Facebook, that provides a simple and flexible approach to forecasting time series data. It uses a decomposable time series model with three main components: trend, seasonality, and holidays. Theory is explained in Section 2.2.2.1. Prophet has been implemented with GluonTS package, which has a wrapper fed with Prophet Package. MXNet is an open-source deep learning framework that provides a flexible and efficient way to implement a wide range of neural network architectures. GluonTS is a probabilistic time series modelling library built on top of MXNet.

Prophet has several hyperparameters that can be adjusted to customize the model's performance and behavior. By adjusting these hyperparameters, the user can fine-tune the model to better fit their specific use case and dataset. Some of the key hyperparameters include:

- **changepoints**: A list of dates at which the rate of change in the trend is allowed to change. By default, Prophet will automatically select changepoints based on the data, but they can also be specified manually.
- **changepoint_prior_scale**: A scalar that controls the strength of the prior for automatic changepoint selection. Larger values will make the model more sensitive to changes in the data.
- **daily_seasonality**: A Boolean that controls whether to model daily seasonality. By default, it is set to "True".
- **growth**: The model's growth function, which can be either "linear" or "logistic". Linear growth is the default and assumes that the rate of change in the time series is constant. Logistic growth allows for the rate of change to vary over time.
- **holidays**: A DataFrame containing information about holidays and events that should be included in the model.
- **n_changepoints**: The number of changepoints to include in the model. Increasing this number will make the model more flexible but also increase the risk of overfitting.
- **seasonality_prior_scale**: A scalar that controls the strength of the prior for the seasonal components. Larger values will make the model more sensitive to changes in the data.

- **weekly_seasonality**: A Boolean that controls whether to model weekly seasonality. By default, it is set to ”True”.
- **yearly_seasonality**: A Boolean that controls whether to model yearly seasonality. By default, it is set to ”True”.

4.2.1 Hyperparameters Prophet

In this section, two tables are presented which outline the hyperparameters configured in Prophet hyperparameter tuning with Sagemaker MXNet image. The first table lists the hyperparameters that were fine-tuned, Figure 4.2, while the second table lists the hyperparameters that were kept constant, Figure 4.3.

Table 4.2: Description of the Hyperparameters to fine-tune of Prophet.

Hyperparameter	Description	Values
`changepoint_prior_scale`	Quantifies changes at the trend change points	[0.001, 0.5]
`seasonality_prior_scale`	Quantifies seasonality flexibility	[0.01, 10]
`holidays_prior_scale`	Quantifies holidays flexibility	[0.01, 10]
`seasonality_mode`	Options are ”additive” or ”multiplicative”	

Table 4.3: Description of non optimized Hyperparameters of Prophet.

Hyperparameter	Description	Values
`growth`	Model’s growth function ”Linear” or ”Logistic”	”Linear”
`changepoints`	Locations of change points	`None`
`n_changepoints`	Number of automatically placed change points	25
`changepoint_range`	Prediction uncertainty interval in so much per one.	0.8
`yearly_seasonality`	`True` if there are more than a year of data, and `False` otherwise.	`True`
`weekly_seasonality`	`True` if there is more than a week of data, and `False` otherwise.	`True`
`daily_seasonality`	`True` if there is more than a day of data, and `False` otherwise.	`True`
`interval_width`	Prediction uncertainty interval in so much per one.	0.8
`uncertainty_samples`	Number of samples to get the uncertainty intervals predictions	1000

4.3 XGBoost Launch

As explained in Section 2.2.2.2 XGBoost is a deep learning-based algorithm for time series forecasting provided by Amazon SageMaker, it is a built-in algorithm.

XGBoost (eXtreme Gradient Boosting) is a popular machine learning library for gradient boosting, theory is explained in Section 2.2.2.2. It is a built-in algorithm provided by Amazon Sagemaker based on the original package. It has several hyperparameters that can be tuned to improve the performance of the model. Some of the most important hyperparameters in XGBoost are:

- **alpha**: L1 regularization term on weights. Increasing this value will make model more conservative.
- **colsample_bytree**: controls the fraction of features used in each iteration of the gradient boosting algorithm. Lower values of colsample_bytree lead to less overfitting but longer training times, while higher values of colsample_bytree lead to more overfitting but faster training times. A common value for colsample_bytree is around 0.8.
- **eta** (also known as learning rate): controls the step size of the gradient descent algorithm. Lower values of eta lead to slower convergence but more accurate models, while higher values of eta lead to faster convergence but less accurate models. A common value for eta is around 0.1.
- **lambda**: L2 regularization term on weights. Increasing this value will make model more conservative.
- **max_depth**: controls the maximum depth of the decision trees in the ensemble. Increasing max_depth leads to more complex models but can also lead to overfitting. A typical value for max_depth is around 6.
- **min_child_weight**: controls the minimum number of samples required to create a new leaf node in the decision tree. Increasing min_child_weight leads to simpler models but can also lead to underfitting. A typical value for min_child_weight is around 1.
- **num_boost_round**: the number of boosting iterations.

- **subsample**: controls the fraction of samples used in each iteration of the gradient boosting algorithm. Lower values of subsample lead to less overfitting but longer training times, while higher values of subsample lead to more overfitting but faster training times. A common value for subsample is around 0.8.

4.3.1 Hyperparameters XGBoost

In this section, two tables are presented which outline the hyperparameters utilized in XGBoost hyperparameter tuning with Sagemaker XGBoost built-in model. The first table lists the hyperparameters that were fine- tuned, Figure 4.4, while the second table lists the hyperparameters that were kept constant, Figure 4.5.

Table 4.4: Description of the Hyperparameters to optimize of XGBoost.

Hyperparameter	Description	Values
alpha	L1 regularization term on weights	[0.01,... ,10]
colsample by level	Subsample ratio of columns for each split, in each level	[0.1,... ,1]
colsample by node	Subsample ratio of columns from each node	[0.5,... ,1]
colsample by tree	Subsample ratio of columns when constructing each tree	[0.5,... ,1]
eta	Step size shrinkage used in updates to prevent overfitting	[0.1,... , 0.5]
gamma	Minimum loss reduction	[0.01,... , 5]
lambda	L2 regularization term on weights	[0.01,... , 1000]
max. delta step	Maximum delta step allowed for each tree's weight estimation	[0,... ,10]
max. delta	Maximum depth of a tree	[6,... ,18]
min. child weight	Minimum sum of instance weight (hessian) needed in a child	[0.01,... ,120]
num. round	The number of rounds to run the training	[100,... ,4000]

Table 4.5: Description of the Hyperparameters which are fix of XGBoost.

Hyperparameter	Description	Value
subsample	Subsample ratio of the training instance	0.8

4.4 DeepAR Launch

As explained in Section 2.2.3.1 DeepAR is a deep learning-based algorithm for time series forecasting provided by Amazon SageMaker, it is a built-in algorithm.

The following are some of the hyperparameters that can be configured when training a DeepAR model using SageMaker:

- **context_length**: Specifies the number of time steps to use as context for forecasting.
- **dropout_rate**: Specifies the dropout rate to use in the network.
- **early_stopping_patience**: Specifies the number of epochs to wait before stopping if the validation loss doesn't improve.
- **epochs**: Specifies the number of training iterations.
- **learning_rate**: Specifies the learning rate for the optimizer.
- **mini_batch_size**: Specifies the number of samples per mini-batch during training.
- **num_cells**: Specifies the number of LSTM cells to use in the network.
- **num_layers**: Specifies the number of layers to use in the network.
- **prediction_length**: Specifies the number of time steps to forecast.
- **time_freq**: Specifies the time frequency of the data, such as 'H' for hourly data or `D` for daily data.

4.4.1 Hyperparameters DeepAR

In this section, two tables are presented which outline the hyperparameters utilized in DeepAR hyperparameter tuning Sagemaker DeepAR built-in model. The first table lists the hyperparameters that were fine- tuned, Figure 4.6, while the second table lists the hyperparameters that were kept constant, Figure 4.7.

Table 4.6: Description of the Hyperparameters to fine-tune of DeepAR.

Hyperparameter	Description	Values
epochs	Number of epochs to train the model	[100,...,300]
learning rate	Step size at each iteration	[1e-5,...1e-1]
dropout rate	Rate of randomly ignored nodes in the network model	[0.1,... ,0.5]
batch size	Number of time series used in each training pass	[50,... ,128]
num. heads	Number of attention heads	[1,2]
hidden. dim	Hidden size or the processing continuous variable	[12,32,64,128]

Table 4.7: Description of the fix Hyperparameters of DeepAR.

Hyperparameter	Description	Values
freq	Frequency of time series	”H” (hourly)
prediction length	Number of data points to predict	24h
context length	Length of each input time series	168h
num. outputs	Number of outputs, or the number of quantiles for QuantileLoss	3

4.5 Temporal Fusion Transformer Launch

The Temporal Fusion Transformer (TFT) is a neural network architecture for time series forecasting, detailed theory is explained in Section 2.2.3.2. TFT has been implemented with GluonTS package. It has been trained with an Amazon build Docker container, which incorporates GluonTS package with MXNet as the backend deep learning framework.

The TFT model has several hyperparameters that can be adjusted to customize the model's performance and behavior. Some of the key hyperparameters include:

- **batch_size**: The batch size used for training the model.
- **dilation_rate**: The dilation rate used in the temporal convolutional layers. This controls the temporal resolution of the model.
- **d_model**: The dimension of the model's hidden state, which controls the capacity of the model.
- **dropout**: The dropout rate used in the model, which controls the regularization.
- **epochs**: The number of training iterations.
- **kernel_size**: The size of the kernel used in the temporal convolutional layers.
- **learning_rate**: The learning rate used for training the model.
- **nhead**: The number of heads in the self-attention layers. This controls the number of parallel attention mechanisms used in the model.
- **num_layers**: The number of layers in the Transformer. This controls the depth of the model.

4.5.1 Hyperparameters of TFT

In this section, two tables are presented which outline the hyperparameters configured in TFT hyperparameter tuning with Sagemaker MXNet image. The first table lists the hyperparameters that were fine- tuned, Figure 4.8, while the second table lists the hyperparameters that were kept constant, Figure 4.9.

Table 4.8: Description of the Hyperparameters to fine-tune of TFT.

Hyperparameter	Description	Values
`epochs`	Number of epochs to train the model	[100,...,300]
`learning_rate`	Step size at each iteration	[1e-5,...1e-1]
`dropout_rate`	Rate of randomly ignored nodes in the network model	[0.1,..., 0.5]
`batch_size`	Number of time series used in each training pass	[50,..., 128]
`num_heads`	Number of attention heads	[1,2]
`hidden_dim`	Hidden size or the processing continuous variable	[12,32,64,128]

Table 4.9: Description of the fix Hyperparameters of TFT Model.

Hyperparameter	Description	Values
`freq`	Frequency of time series	″H″ (hourly)
`prediction_length`	Number of data points to predict	24h
`context_length`	Length of each input time series	168h
`num_outputs`	Number of outputs of the number of quantiles for Quantile Loss	3

Chapter 5

Model Selection

In this chapter, we will evaluate four different models (see Chapter 2 for the architecture details and Chapter 4 for the setup details) with varying architectures to determine the best model for our task. The models we will be evaluating are:

- *Prophet:* is a time series forecasting model developed by Facebook. It is based on decomposing a time series into three components: trend, seasonality, and holidays. Prophet uses a Bayesian additive regression model to fit these components and then makes predictions using a Monte Carlo simulation.

- *XGBoost:* (eXtreme Gradient Boosting) is a gradient boosting algorithm that uses decision trees as the base model. It is known for its high accuracy and efficiency, and is often used in machine learning competitions and large-scale data mining tasks. XGBoost uses an ensemble of decision trees and an optimization algorithm called gradient boosting to improve the performance of the model.

- *DeepAR:* is a deep learning-based model for time series forecasting developed by Amazon. It is based on a recurrent neural network architecture using LSTM units, with a special attention to the dependencies among multiple time series. It uses a combination of past observations and external features to make predictions.

- *TFT:* is a neural network architecture for time series forecasting. It is based on the transformer architecture, which is known for its ability to handle sequences of variable length and its attention mechanism. TFT extends the transformer architecture by adding a temporal attention mechanism that allows the model to focus on different parts of the time series at different levels of granularity.

Evaluation Criteria:

- *MAPE:* is a measure of the average percentage error of a set of predictions. It is calculated as the average of the absolute percentage differences between the predicted values and the actual values. MAPE is commonly used in finance and economics to measure the accuracy of a forecast, but it is not recommended to use it when the true values contain zeros.

- *MAE:* is a measure of the average magnitude of the errors in a set of predictions, without considering their direction. It is calculated as the sum of the absolute differences between the predicted values and the actual values, divided by the number of predictions. MAE is a popular metric for regression problems because it is easy to interpret, but it is sensitive to outliers.

- *RMSE:* is a measure of the average magnitude of the errors in a set of predictions, but it gives more weight to larger errors. It is calculated as the square root of the mean of the squared differences between the predicted values and the actual values.

Each model has been trained with multiple dataset configurations regarding the dataset features and the order of the training dataset rows. Section 5.1 defines each model setup, regarding the dataset configuration, relating them with the model names for an easy understanding of the metrics results tables.

Note that there is an important consideration important to understand the results presented in this section. As mentioned in Chapter 3, *demand* variable is a special feature due to the fact that it is the same concept as the target, *total_consumption*. However, *demand* is a forecasting made by REE for a higher aggregated level, it is a prediction for the hole Spanish people living in the Iberian Peninsula but *total_consumption* is the demand of all CUPS, clients of Holaluz, which lives in the Iberican Peninsula and have "3.0TD" product contracted. Therefore, it is interesting to check if the *demand* feature is actually contributing to the *total_consumption* prediction or not, i.e. some models incorporate it and some others do not.

5.1 Comparison setup

As mentioned before, each model has been trained with multiple dataset configurations regarding the dataset features and the order of the training dataset rows. Table 5.1 shows the codification used per each dataset configuration. Hence, the model names are written according to this notation: `model_name` = {`first_term`} - {`mid_term`} - {`last_term`}

Table 5.1: Model name codifications according to the dataset setup. An example of a model name is given is the last column.

Term	Description	Code	Example
`first_term`	Prophet	prophet	prophet-u-b
	XGBoost	xgb	xgb-u-b
	DeepAR	deepar	deepar-u-c
	TimeFusionTransformer	tft	tft-u-c

Term	Description	Code	Features	Example
`mid_term`	Univariate	u	*None*	tft-u-c
	Multivariate	m7nD	labor days + 4 weather+ 2 customers_info	tft-m7nD-b
		m8	labor days + 4 weather+ 2 customers_info + "demand"	tft-m8-b
		mallnD	labor days + 9 weather+ 3 customers_info	tft-mallnD-b
		mall	labor days + 9 weather+ 3 customers_info + "demand"	tft-mall-b

Term	Description	Code	Time related features	Example
`last_term`	Shuffling	a	hour+day_of_week+day_of_month+month+year	xgb-mall-a
	No Shuffling	b	hour+day_of_week+day_of_month+month+year	xgb-mall-b
		c	*None*	deepar-m8-c

A table containing the model name and setups is showed before presenting the results obtained with a specific algorithm in order to clarify the configuration of the trained models.

5.2 Comparison metrics

The evaluation of the models is carried out using both *MAE*, *MAPE* and *RMSE* metrics, see Section 2.3.6. Nevertheless, results metrics tables are sorted by *MAPE* which have been the principal metric to look at but not the only one indeed. Note that *MAE* has been added to the tables as it is a popular metric but is only useful for readers who are familiar with the target values scale.

Recalling the theory once more, *MAPE* is denoted as the mean absolute percentage error so that it is the average multiplicative effect between the observed data and prediction i.e. their units, percentage, facilitates the understanding of the value. On the other hand,

RMSE, denoted as the root mean squared error, is the standard deviation of the observed and the predicted target i.e. same units and magnitude as the target. Notice that this metric penalizes more large errors than smaller, thus, it can be said that *RMSE* is stricter about how the model fits to the observed data compared to *MAPE*. To conclude, both metrics are interesting for the comparison but *MAPE* would be the first one to check and the tables which shows the evaluation metrics per model are sorted by this metric.

To understand the following section, Section 5.3, there are three important concepts to be clarified. A table of evaluation metrics is presented for each trained model displaying the *MAPE*, *MAE* and *RMSE* after applying inference to the training and the test set. The table illustrates the *Training Error* metrics, the *Validation Error* and *Test error* which correspond to comparison of the observed and predicted target values using each dataset split i.e. train, validation and test datasets. Hence, they can be defined as:

- *Training Error*: A metric representing the performance of a model predicting the target value (i.e. in this case the target, "total consumption") using the dataset already seen by the model i.e. the dataset used for training. In the literature it is also known in other scientific areas as *Empirical Error* as described in [Mohri et al. 2018].

- *Validation Error*: A metric representing the performance of a model predicting the target value using a dataset not seen during training process but used to calculate the objective function (which in a Regression Problem is usually the *MAPE* or the *RMSE*) to optimize the model hyperparameters, i.e. carry out the Hyperparameter Tuning.

- *Test Error*: A metric representing the performance of a model predicting the target value using a dataset never seen by the model during the whole training process, including Hyperparameter Tuning. In the literature it is commonly known as *Generalization Error* as described in [Mohri et al. 2018].

The differences in evaluation metrics between the training set and the test set can provide insight into how well the model is generalizing to new, unseen data.

- Overfitting: A large difference in evaluation metrics between the training set and the test set could indicate that the model is overfitting to the training data. This means that the model is memorizing the training data, rather than learning the underlying patterns. As a result, the model will perform well on the training data but poorly on new, unseen data.

- Model Complexity: A large difference in evaluation metrics between the training set and the test set could also indicate that the model is too complex for the data. A complex model may fit the training data very well, but struggle to generalize to new data.

- Data Distribution: The difference in evaluation metrics between the training set and the test set can also be caused by differences in the data distribution. If the test data is significantly different from the training data, the model may perform poorly on the test set even if it has not overfitted.

- Evaluation of the model: The difference in evaluation metrics between the training set and the test set can also be caused by the fact that the test data has not been seen by the model before, this can lead to lower metrics on the test set, even if the model is not overfitting.

It is important to keep in mind that the goal of machine learning is to generalize well to new unseen data, so lower evaluation metrics on the test set compared to the training set are desirable as it suggests that the model is generalizing well to new unseen data and not just memorizing the training data.

Nevertheless, it is generally considered normal to see a slightly higher evaluation metric on the training set compared to the test set, this is because the model has seen the training data before, and it has learned the underlying patterns, thus it is expected to perform better on the training set. However, if the difference between training and test evaluation metrics is too large, it can indicate that the model is overfitting to the training data, which means that the model is not generalizing well to new unseen data.

A general rule of thumb is to aim for a difference of less than 10% between the training and test evaluation metrics. According to [Geman et al. 1992] this is a good rule of thumb for the generalization error on unseen data. However, it is important to keep in mind that this can vary depending on the specific task and the dataset.

It is important to use techniques such as cross-validation and regularization to prevent overfitting, and to use techniques such as grid search or random search to find the optimal values for the hyperparameters. Definitions of cross-validation, regularization techniques and hyperparameter tuning can be found in 2.3.3, 2.3.4 and 2.3.5 respectively.

In conclusion, the normal rate between training and test evaluation metrics can vary depending on the specific task and dataset, but it is generally considered normal to see a

slightly higher evaluation metric on the training set compared to the test set.

5.3 Model Comparison

This section is devoted to present the results of the models. First, the model setup configuration tables are shown. Then comparison of the results of different model setups is carried out per each model in the corresponding section.

5.3.1 Prophet

Model Setup configuration

Table 5.2 presents the name of each Prophet model setup, with the name codification explained in Section 5.1. Differences between each setup variation are the dataset features.

Table 5.2: Features used per each variation of Prophet model.

	prophet-u-c	prophet-m7nD-c	prophet-m8-c	prophet-mallnD-c	prophet-mall-c	prophet-u-b	prophet-m7nD-b	prophet-m8-b	prophet-mallnD-b	prophet-mall-b
demand			x		x			x		x
weighted_labor_days		x	x	x	x		x	x	x	x
short_wave_radiation		x	x	x	x		x	x	x	x
relative_humidity		x	x	x	x		x	x	x	x
total_cups_with_consumptions		x	x	x	x		x	x	x	x
total_active_cups		x	x	x	x		x	x	x	x
wind_speed		x	x	x	x		x	x	x	x
heat_index		x	x	x	x		x	x	x	x
availability				x	x				x	x
temperature				x	x				x	x
windchill				x	x				x	x
temperature_min				x	x				x	x
temperature_max				x	x				x	x
precipitation_rate				x	x				x	x
hour_sin						x	x	x	x	x
hour_cos						x	x	x	x	x
day_of_week_sin						x	x	x	x	x
day_of_week_cos						x	x	x	x	x
day_sin						x	x	x	x	x
day_cos						x	x	x	x	x
month_sin						x	x	x	x	x
month_cos						x	x	x	x	x
year						x	x	x	x	x

Metrics results

In this section, Prophet algorithm results are presented. Table 5.3 shows the evaluation metrics of Prophet model for each of the model setup. Firstly, it has to be said that the values of the metrics are not so good, they clearly indicate that Prophet is not a great model to use for this problem on this dataset.

Table 5.3: MAPEs train and test prophet model.

	MAPE (%)		MAE (MW)		RMSE (MW)	
split	test	train	test	train	test	train
MODEL						
prophet-m8-c	13.35	13.10	2,098.46	2,971.49	2,867.81	3,948.33
prophet-mall-c	13.91	13.27	2,224.47	3,269.34	3,089.87	4,565.72
prophet-mall-b	14.81	14.02	2,406.90	3,248.03	3,248.45	4,480.42
prophet-m8-b	15.24	13.26	2,439.09	3,044.39	3,200.56	3,979.34
prophet-m7nD-c	16.78	15.50	2,620.63	4,335.53	3,301.33	5,680.47
prophet-mallnD-c	18.49	17.51	2,985.73	4,998.97	3,863.54	6,848.09
prophet-m7nD-b	19.64	18.36	3,131.83	4,777.86	3,985.99	6,304.64
prophet-mallnD-b	22.15	20.07	3,277.15	5,170.64	4,285.86	7,100.34
prophet-u-c	31.14	29.58	5,054.17	6,699.84	6,296.64	8,571.56
prophet-u-b	32.87	31.59	5,297.20	7,200.76	6,539.77	8,911.37

Looking at the MAPE (%) column, globally models with time cyclic variables (`-c`) are not contributing in the model performance compared with models with them (`-b`). On the other hand, models which do not contain the feature *demand* (`-nD-`) in general are worst than the others. Then, related to the number of features, it can be seen that there is not a clear difference between models with half the features (`-m7nD-` and `-m8-`) and the complex ones with all features (`-mall-` and `-mallnD-`). Finally, the model which gets better results, both looking MAPE and RMSE metrics, on the test dataset is definitely `prophet-m8-c` with 13.35% of MAPE. Additionally, it can be seen that train set MAPEs are smaller than in the test set as it is expected, explanation is in Section 2.3.6. Furthermore, there is no big differences between them, which allows stating that apparently there is no over-fitting.

5.3.2 XGBoost

Model setup configuration

Table 5.4 presents the name of each TFT model setup with the name codification explained in Section 5.1. Differences between each setup variation are the dataset features. Note that last row of Table 5.4 is not containing the name of a feature, *Shuffled train and validation datasets* is just indicating when resampling have been applied to the dataset of training.

Table 5.4: Features used per each variation of XGBoost model.

	xgboost-u-a	xgboost-m7nD-a	xgboost-m8-a	xgboost-mallnD-a	xgboost-mall-a	xgboost-u-b	xgboost-m7nD-b	xgboost-m8-b	xgboost-mallnD-b	xgboost-mall-b
demand			x		x			x		x
weighted_labor_days		x	x	x	x		x	x	x	x
short_wave_radiation		x	x	x	x		x	x	x	x
relative_humidity		x	x	x	x		x	x	x	x
total_cups_with_consumptions		x	x	x	x		x	x	x	x
total_active_cups		x	x	x	x		x	x	x	x
wind_speed		x	x	x	x		x	x	x	x
heat_index		x	x	x	x		x	x	x	x
availability				x	x				x	x
temperature				x	x				x	x
windchill				x	x				x	x
temperature_min				x	x				x	x
temperature_max				x	x				x	x
precipitation_rate				x	x				x	x
hour_sin	x	x	x	x	x	x	x	x	x	x
hour_cos	x	x	x	x	x	x	x	x	x	x
day_of_week_sin	x	x	x	x	x	x	x	x	x	x
day_of_week_cos	x	x	x	x	x	x	x	x	x	x
day_sin	x	x	x	x	x	x	x	x	x	x
day_cos	x	x	x	x	x	x	x	x	x	x
month_sin	x	x	x	x	x	x	x	x	x	x
month_cos	x	x	x	x	x	x	x	x	x	x
year	x	x	x	x	x	x	x	x	x	x
Shuffled train and validation datasets	x	x	x	x	x					

Metrics results

In this section, XGBoost results are presented. Table 5.7 shows the evaluation metrics of XGBoost model for each of the model setup.

Table 5.5: MAPEs train and test xgb model.

	MAPE (%)		MAE (MW)		RMSE (MW)	
split	test	train	test	train	test	train
MODEL						
xgb-mallnD-a	7.95	0.10	1,494.14	21.54	1,901.91	32.50
xgb-m8-a	8.54	0.00	1,530.18	0.62	1,990.46	0.78
xgb-m8-b	9.79	0.25	1,568.34	63.69	1,820.65	127.12
xgb-mall-a	10.30	0.37	1,820.98	90.10	2,303.41	137.40
xgb-mall-b	11.76	0.10	1,835.20	26.98	2,108.06	71.64
xgb-m7nD-a	13.99	0.40	2,393.34	87.10	2,867.71	128.96
xgb-mallnD-b	15.28	0.10	2,448.97	21.67	2,988.06	60.22
xgb-m7nD-b	25.29	1.25	4,053.08	297.10	4,402.88	426.60
xgb-u-a	47.21	0.10	8,714.98	22.07	9,527.32	42.28
xgb-u-b	51.52	0.11	9,260.02	25.16	10,255.93	44.06

Looking at the MAPE (%) column, globally models with train and validation set shuffling (`-a`) clearly outperforms the not shuffled (`-b`). It can be checked by looking values per each model setup pair, it is clear that `-a` appears before `-b`. Moreover, at first sight models which do not contain features (`xgb-u-`) are having extremely bad results compared to the others, which do contain features. On the other hand, in general models which do not contain the feature *demand* (`-nD-`) are worst than the the others, but the best model actually have no *demand*. Then, related to the number of features, it can be seen that there is not a clear difference between models with half the features (`-m7nD-` and `-m8-`) and the complex ones with all features (`-mall-` and `-mallnD-`). Finally, the model which gets better results, both looking MAPE and RMSE metrics, on the test dataset is definitely `xgb-mallnD-a` with 7.95% of MAPE. Additionally, it can be seen that both the train set MAPEs and RMSEs are so much smaller than in the test set. Differences shows that there is a clear overfitting and the model is not generalizing well, explanation is in Section 2.3.6.

5.3.3 DeepAR

Table 5.6 presents the name of each DeepAR model setup with the name codification explained in Section 5.1. Differences between each setup variation are the dataset features.

Model setup configurations

Table 5.6 presents the name of each DeepAR model setup with the name codification explained in Section 5.1. Differences between each setup variation are the dataset features.

Table 5.6: Features used per each variation of DeepAR model.

	deepar-u-c	deepar-m7nD-c	deepar-m8-c	deepar-mallnD-c	deepar-mall-c	deepar-u-b	deepar-m7nD-b	deepar-m8-b	deepar-mallnD-b	deepar-mall-b
demand			x		x			x		x
weighted_labor_days		x	x	x	x		x	x	x	x
short_wave_radiation		x	x	x	x		x	x	x	x
relative_humidity		x	x	x	x		x	x	x	x
total_cups_with_consumptions		x	x	x	x		x	x	x	x
total_active_cups		x	x	x	x		x	x	x	x
wind_speed		x	x	x	x		x	x	x	x
heat_index		x	x	x	x		x	x	x	x
availability				x	x				x	x
temperature				x	x				x	x
windchill				x	x				x	x
temperature_min				x	x				x	x
temperature_max				x	x				x	x
precipitation_rate				x	x				x	x
hour_sin						x	x	x	x	x
hour_cos						x	x	x	x	x
day_of_week_sin						x	x	x	x	x
day_of_week_cos						x	x	x	x	x
day_sin						x	x	x	x	x
day_cos						x	x	x	x	x
month_sin						x	x	x	x	x
month_cos						x	x	x	x	x
year						x	x	x	x	x

Metrics results

In this section, DeepAR results are presented. Table 5.7 shows the evaluation metrics of DeepAR model for each of the model setup.

Table 5.7: MAPEs train and test deepAR model.

	MAPE (%)		MAE (MW)		RMSE (MW)	
split	test	train	test	train	test	train
MODEL						
deepar-u-c	11.05	6.45	1,799.23	1,862.90	2,421.96	3,016.71
deepar-m8-b	11.55	10.43	2,023.31	2,572.47	2,594.61	3,441.13
deepar-u-b	14.79	11.90	2,565.56	3,136.69	3,441.09	4,289.49
deepar-m8-c	14.93	10.20	2,429.29	2,629.38	3,075.30	3,534.40
deepar-mall-b	17.00	18.82	3,555.00	5,660.38	5,074.61	8,022.64
deepar-m7nD-c	24.99	21.61	3,979.24	5,091.35	5,044.31	6,731.77
deepar-m7nD-b	28.51	20.94	4,785.72	4,897.66	5,661.56	6,787.85
deepar-mallnD-b	28.68	22.21	4,810.13	5,286.36	5,660.40	6,955.45
deepar-mallnD-c	32.35	24.60	5,384.62	5,644.91	6,270.13	7,449.52
deepar-mall-c	32.45	33.84	5,560.64	8,107.87	6,841.15	10,006.58

Looking at the MAPE (%) column, globally models with time cyclic variables (`-c`) achieve better results than models with them (`-b`) checking values per each pair, in general `-b` apears before `-c`. Moreover, at first sight models which do not contain features (`deepar-u-`) significantly outperform the others, which do contain features (`deepar-m`). Furthermore, there is the worst performance from the models which do not contain the feature *demand* (`-nD-`) with the others, it can be concluded that *demand* is an important feature for XGBoost affecting positively the forecasting performance. Finally, the model which gets better results, both looking MAPE and RMSE metrics, on the test dataset is definitely `deepar-u-c` with 11.05% of MAPE.

To conclude, recalling the explanations given in Section 5.2, it can be seen that train set MAPEs are smaller in the training set than in the test set. However, RMSE values are greater in the training set than in the test set for most of the models. Therefore, since RMSE metric is stricter than MAPE it is concluded that the DeepAR is not presenting overfitting, and it is generalizing well.

5.3.4 Temporal Fusion Transformer

Model setup configurations

Table 5.8 presents the name of each TFT model setup with the name codification explained in Section 5.1. Differences between each setup variation are the dataset features.

Table 5.8: Features used per each variation of TFT model.

	tft-u-c	tft-m7nD-c	tft-m8-c	tft-mallnD-c	tft-mall-c	tft-u-b	tft-m7nD-b	tft-m8-b	tft-mallnD-b	tft-mall-b
demand			x		x			x		x
weighted_labor_days		x	x	x	x		x	x	x	x
short_wave_radiation		x	x	x	x		x	x	x	x
relative_humidity		x	x	x	x		x	x	x	x
total_cups_with_consumptions		x	x	x	x		x	x	x	x
total_active_cups		x	x	x	x		x	x	x	x
wind_speed		x	x	x	x		x	x	x	x
heat_index		x	x	x	x		x	x	x	x
availability				x	x				x	x
temperature				x	x				x	x
windchill				x	x				x	x
temperature_min				x	x				x	x
temperature_max				x	x				x	x
precipitation_rate				x	x				x	x
hour_sin						x	x	x	x	x
hour_cos						x	x	x	x	x
day_of_week_sin						x	x	x	x	x
day_of_week_cos						x	x	x	x	x
day_sin						x	x	x	x	x
day_cos						x	x	x	x	x
month_sin						x	x	x	x	x
month_cos						x	x	x	x	x
year						x	x	x	x	x

Metrics results

Table 5.9: MAPEs dataset train and test TFT model.

	MAPE (%)		MAE (MW)		RMSE (MW)	
split	test	train	test	train	test	train
MODEL						
tft-m7nD-c	4.77	3.59	840.66	904.65	1,286.75	1,588.02
tft-m8-c	4.99	3.67	884.70	950.04	1,423.66	1,523.90
tft-mallnD-c	5.42	3.79	1,005.30	982.66	1,578.02	1,538.16
tft-mall-c	5.55	4.37	967.89	1,122.43	1,477.00	1,841.61
tft-m7nD-b	5.56	3.22	958.39	788.83	1,780.39	1,269.83
tft-mallnD-b	5.61	3.46	966.98	876.57	1,840.89	1,477.73
tft-u-c	5.70	4.35	960.95	1,104.88	1,793.46	2,106.22
tft-mall-b	5.72	3.78	1,026.91	967.86	1,659.39	1,427.21
tft-u-b	5.80	3.69	980.13	922.66	1,728.70	1,510.30
tft-m8-b	6.27	3.36	1,061.98	840.60	1,857.56	1,369.48

In this section TFT results are presented. Table 5.9 shows the evaluation metrics of TFT model for each of the model setups. Firstly, it has to be said that the values of the metrics are quite similar and there are just small differences between models. Note that current differences are not relevant enough to grant strict conclusions.

Looking at the MAPE (%) column, globally models without time cyclic variables (`-c`) achieve better results, with an average MAPE of 5.3%, than models with them (`-b`) with an average MAPE of 5.8%. Moreover, at first sight, models which do not contain features (`tft-u-`) behave less good than the others. However, models with half of the features (`-m7nD-` and `-m8-`) or all (`-mall-` and `-mallnD-`) get similar results in average 5.4% and 5.6% respectively, i.e. having more features is not improving results. The same happens when comparing models which do not contain the feature *demand* (`-nD-`) with the others, it can be concluded that *demand* is not affecting the forecasting performance. Finally, the model which gets better results, both looking MAPE and RMSE metrics, on the test dataset is definitely `tft-m7nD-c` with 4.77% of MAPE.

To conclude, recalling the explanations given in Section 5.2, it can be seen that train set MAPEs are smaller in the training set than in the test set. However, RMSE values are greater in the training set than in the test set for the best models. Therefore, since RMSE metric is stricter than MAPE it is concluded that the best model is not presenting overfitting and could be generalizing well.

5.3.5 Algorithm Comparison

In this section, the best results of each algorithm are collected. Table 5.10 shows the evaluation metrics for the train and test datasets with the best model of each algorithm.

Table 5.10: Evaluation metrics of the best models of each algorithm.

	MAPE (%)		MAE (MW)		RMSE (MW)	
split	test	train	test	train	test	train
MODEL						
tft-m7nD-c	4.77	3.59	840.66	904.65	1,286.75	1,588.02
xgb-mallnD-a	7.95	0.10	1,494.14	21.54	1,901.91	32.50
deepar-u-c	11.05	6.45	1,799.23	1,862.90	2,421.96	3,016.71
prophet-m8-c	13.35	13.10	2,098.46	2,971.49	2,867.81	3,948.33

From Table 5.10 it can be seen that the best model is achieved with Temporal Fusion Transformer algorithm, the second one is achieved by XGBoost, the third position is achieved by DeepAR and the last one for Prophet. Note that only one of the models (XGBoost) is including the time cyclic encoders, however it is the only model that has not time encoding transformations included in the algorithm itself. On the other hand, there is no coincidence on the setup features in each model, each one of them is using different features. An important observation regarding the *demand* contribution on the models results, models with greater errors, i.e. DeepAR and Prophet, are those which performs better with this feature. However, TFT which is understanding the time series fluctuations do not incorporate it. In essence, DeepAR and Prophet are generalizing well and do present better results on the test than in the training set if checking RMSE an MAE metrics, it is not happening with the MAPE metric though. Contrary, XGBoost is remarkably overfitting due to tha contrast on train and test evaluation metrics i.e. it most certainly is not generalizing well. Finally, TFT model is slighly less good on the test set according to MAPE and MAE metrics but the opposite happens with the RMSE i.e. overfitting is discarded since it is a common fact as explained in Section 5.2.

The best model found in the results is the `tft-m7nD-c`, which is a TemporalFusionTransformer model with weighted labor days, short wave radiation, relative humidity, total cups with consumptions, total activa cups, windspeed and heat index as features and without cyclic time encoders. It turns out that the MAPE of this model is less than a 5% which is definitely a great value for being a time series forecasting model.

Table 5.11 presents the minimum and the maximum values on the evaluation metrics per each algorithm. Based on the data presented, TFT is a good algorithm to solve this

Table 5.11: Minimum and maximum evaluation metrics of the best models of each algorithm.

	MAPE (%)		MAE (MW)		RMSE (MW)	
	min	max	min	max	min	max
ALGORITHM						
TFT	4.77	6.27	840.66	1,061.98	1,286.75	1,857.56
XGBoost	7.95	51.52	1,494.14	9,260.02	1,901.91	10,255.93
DeepAR	11.05	32.45	1,799.23	5,644.91	2,421.96	6,481.15
Prophet	13.35	32.87	2,098.46	5,297.20	2,867.81	6,539.77

problem due to that the minimum and maximum evaluation metrics are not far from each other. Overall it appears that the other models are not as good as this minimum-maximum range is definitely larger.

The following Figures 5.1 and 5.2 illustrates the times series forecasting the test dataset features for the best models of: Prophet, XGBoost, DeepAR and TFT algorithms. The blue line represents the observation of *total_consumption* and the red line represents the predicted value.

The analysis of the Prophet model's predictions, Figure 5.1 (a), reveals that the model exhibits a moderate understanding of the seasonal patterns present in the data. However, it is not highly accurate in predicting the magnitude of peaks and valleys. Further examination reveals that the model does not fully capture the gradual changes in level observed throughout the data, particularly in relation to midday peaks. A potential explanation for this is that the model's changepoint locations are not well-calibrated. Additionally, it can be observed that the values on most peaks exhibit a triangular pattern, suggesting that the model may not fully capture the nuances of the data. A positive aspect of the model is its ability to accurately identify weekends, as it is able to discern the differing patterns present on these days compared to weekdays. Additionally, the model has been able to identify that November 1st was a holiday, as the pattern observed on this day is lower than that observed on weekdays.

In Figure 5.1 (b), the predictions generated by the XGBoost model indicate that it exhibits a strong understanding of the seasonal patterns present in the data. However, the model is not highly accurate in predicting the magnitude of peaks and valleys. Analysis shows that the model tends to shift the valleys and peaks upward, potentially due to significant overfitting. Additionally, it can be observed that the training dataset generally has higher *total_consumption* values, including in the peaks, valleys, and general trend, as can be seen in Figure 3.5.

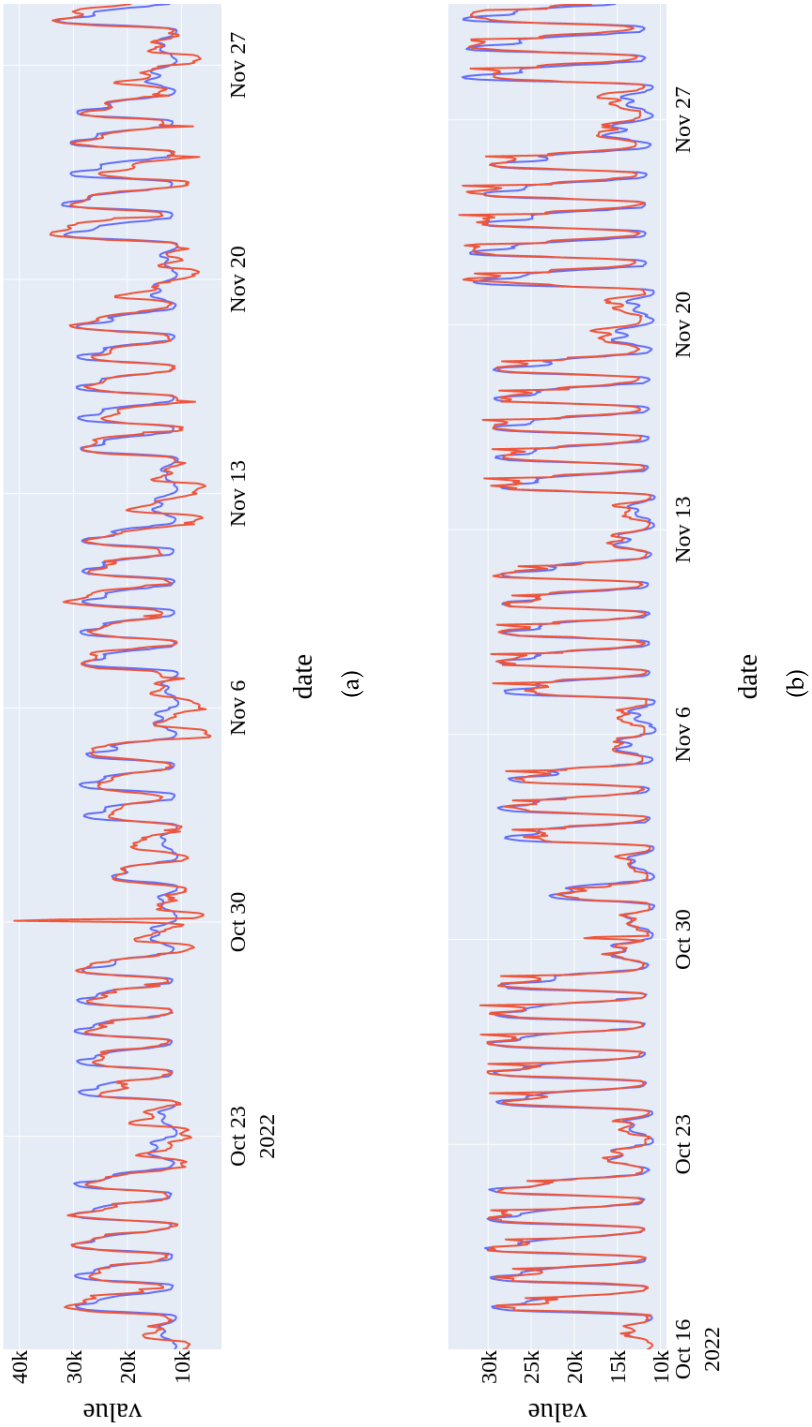


Figure 5.1: Time series of the observed *total_consumption* (blue) and predictions (red) of the best models per algorithm: (a) Prophet (b) XGBoost.

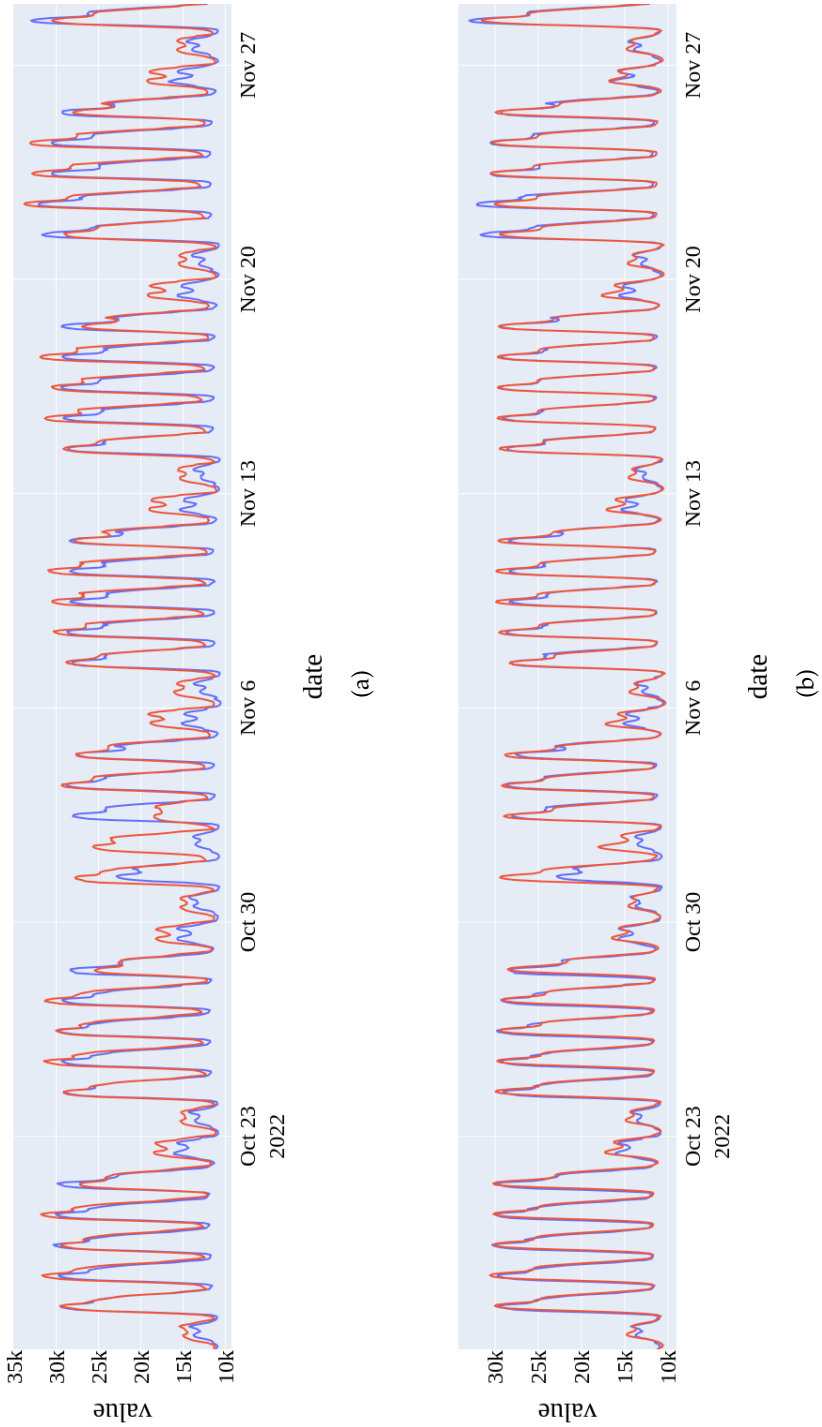


Figure 5.2: Time series of the observed *total_consumption* (blue) and predictions (red) of the best models per algorithm: (a) DeepAR and (b) TFT.

Figure 5.2 (a), which depicts the prediction of the DeepAR model, illustrates that the model has a good understanding of the seasonality of the data. However, similar to the XGBoost model, there are small errors present. Specifically, there is a slight shift upwards in the peaks and valleys, but it is not as exaggerated as the errors observed in the XGBoost model. Additionally, the model has not been able to identify that November 1st was a holiday, despite the fact that the feature *weighted_labor_days* was provided to the model. Even with these errors, the overall shape of the prediction is well-fitted. However, small scaling differences on the peaks and valleys result in increased errors in the prediction.

Figure 5.2 (b) shows the prediction of the TFT model. It appears to be of good quality almost fitting the observations almost perfectly. Just a few errors are present. One error that can be identified is on the prediction for October 31st, the day before the holiday. This day is a labor day, but it is unique in that it is between two non-labor days. As a result, the observed value was lower than what is typical. The model has not identified this pattern of labor days between non-labor days. It is suggested that training it with more years of data would likely increase the prediction accuracy. Furthermore, in broad terms, errors tend to concentrate on the weekends and there is a moderate over-prediction.

Last but not least, it has been assumed that a model which give satisfactory predictions on a 24h-window might be able to predict on a larger horizon using their own predictions as input. The best model, `tft-m7nD-c`, which is the best TemporalFusionTransformer model has been tested as well to predict on a forecast horizon of 7, 8 and 14 days, and it surprisingly showed in an average MAPE of 5.7% for the "test dataset", see 5.12.

Table 5.12: MAPE on predictions with a forecast horizon of 7, 8 and 14 days

Forecast horizon days	MAPE (%)
7	5.21
8	5.28
14	5.73

Chapter 6

Summary and Future work

Problem statement

Current thesis was conducted using the data from HolaLuz Company as a part of a master project agreement. HolaLuz is one of the relevant electricity providers in Spain with almost 400,000 users dedicated to the commercialization of electrical energy of 100 per cent renewable origin and gas. The company classifies its customers into predefined segments and uses a forecasting algorithm for the electricity consumption prediction for each segment, "total_consumption".

It is important to mention that HolaLuz has access to the hourly forecasting of electricity demand provided by REE. This prediction is the aggregated forecasted demand of the Spanish population residing in the Iberian Peninsula. The algorithms which is Holaluz is currently using are: an XGBoost model and an Heuristic approach. The XGBoost is trained with a dataset of features including weather, labor days and customers info and the "demand". The Heuristic approach is basically the "demand" provided by REE scaled according to the amount of CUPS per segment of Holaluz and the moving average on a fixed time span.

Current implementations have their weaknesses which have a significant effect on the precision. Firstly, XGBoost has been used profitably for forecasting time series. Still, being a tree-based model, it has no capability to extrapolate beyond the range of the training data. This is essential for non-stationary time-series. Hence, the model occasionally predicts unreliable "total_consumption". On the other hand, the Heuristic approach, which in average is always having good results, is almost only depending on the "demand" from REE, and it is constrained to the performance of REE model which is a black box for HolaLuz. Note that XGBoost is trained with "demand" as a feature as well. Finally and most

important, the model is making a prediction based on hourly data such as previous "total_consumptions", temporal features (day of week, festivity day, month), weather data (heat index, max temperature, precipitation), and customer's information. The algorithm is executed every day to predict the clients' hourly consumption for the next 8 days to give time to HolaLuz's electricity buyers to purchase the required electricity per product in the following days (actually, only 2 days are needed, the current and the following day). However, it takes about 7 to 10 days to get the total clients' consumption, "total_consumption", which means that at the "execution_time" only a fraction of the total information is available.

Research hypothesis

In the current thesis, an improvement of current HolaLuz's forecasting models of hourly electricity consumption is achieved by using advanced Deep Learning techniques.

Design of solution

Current models are an XGBoost which retrains daily with new data available (day before) and an Heuristic approach which is a scaling of "demand" prediction from REE. In order to be able to compare current models with the study proposed models a time span is defined to evaluate their predictions, data from this time span is denoted as "test dataset".

The first model, XGBoost, has an average MAPE on the past predictions vs. observations (also named "real") about 9.0%. Then, using the "test dataset" in the evaluation interval, it reaches a MAPE of 11.5%. Meanwhile, the Heuristic model has an average MAPE of 9.6% on past predictions vs. observations and shows a MAPE of 8.1% using the "test dataset".

The current study presents several models to improve the current models' performance. Four different algorithms have been implemented. There are two Machine Learning (ML) models: Prophet and XGBoost, and two Deep Learning (DL) models: DeepAR and Time Fusion Transformer (TFT). Firstly, Prophet is proposed due to its capability of detecting non-linear trends and fit with yearly, weekly and daily seasonality. Despite the fact that XGBoost is already a current model, it is developed including hyperparameter tuning, which was not implemented in the current HolaLuz's XGBoost. Then, DeepAR and Time Fusion Transformed are implemented for their known generalized high accuracy in many fields, the hability of handling multiple timeseries, complexity of the model allowing understanding more complex patterns.

Moreover, the algorithms have been trained using different dataset configurations which derived to several model setups. Different datasets has been used for the training containing different features and time cyclic encoders have been optionally added as well. Additionally, in the case of XGBoost, due to the architecture of the algorithm it has been possible to shuffle the training dataset and check possible improvements by applying this.

It is important to note that the forecast horizon of the training has been of 24 hours despite the fact that the current problem requirements might output predictions for the next 2 days at least, i.e. 48h. Moreover, as aforementioned, "total_consumption" of the last 10 days (at most) is not complete at the "execution_date", hence if the algorithm requires the complete time series till the "execution_date" the horizon of forecasts becomes 10 days plus 48h (at least 2 days in the future are needed to be predicted), i.e. 12 days. This requirement applies for Prophet, DeepAR and Temporal Fusion Transformer. However, it has been assumed that a model which give satisfactory predictions on a 24h-window might be able to predict on a larger horizon using their own predictions as input.

The multiple model setups have been trained, and hyperparameter tuning has been carried out. Afterwards, the final models have been used to predict the "total_consumption" with the "test dataset". A comparison of the different model setups per algorithm has been performed. Also, a comparison between the best model setups per each of the 4 algorithms. Finally, by using MAPE and RMSE evalutaion metrics, the best solution is proposed.

Results

The best model according to the "test dataset" evaluation was Temporal Fusion Transformer (TFT) using a dataset configuration which included features related with: labor days, 4 weather measures, 2 CUPS metadata measures and time cyclic encodings. It is important to mention that the proposed solution reads out the "demand" feature, i.e. it is independent of predictions from REE. The MAPE which presented the best model is about 4.8%. Therefore, this model is selected as the proposed model.

Last but not least, the proposed model has been tested as well to predict on a forecast horizon of 14 days, and it surprisingly showed in an average MAPE of 5.7% for the "test dataset". The model is clearly outperforming the current best model of HolaLuz for two reasons:

- it is showing a MAPE smaller than the current best model, 5.7% with TFT in contrast to 8.1% with the Heuristic approach,
- it is not including the ”demand” variable, the prediction from REE.

Future Work

It is important to note that the cross-validation of the models has not been carried out and should be included in future work. However, the model has been applied to the ”training dataset” to compare the predictions of ”already seen data”. The MAPE resulting from the ”train dataset” showed a 3.6% of error and this does not suggest the presence of overfitting in the model comparing it with the test error 4.8%. Nevertheless, bias can be incorporated through the time range which comprises the dataset which can not be representative enough to guarantee the performance of the model in the future due to that the target timeseries is non-stationarity, hence an improvement of the model can be carried out once more data has been collected. Finally, the performance of the model can be enhanced by doing an exhaustive analysis of the feature combinations in the model setup which can lead to even better results, including the addition of external data such economic indicators, demographic factors or other indicators that may impact electricity consumption.

Appendix A

Prediction of current HolaLuz model

The predictions of test set for the models that HolaLuz currently has deployed are presented in Figure A.1. In there ”[Prod]” corresponds to the results of the Heuristic Model and ”[207]” the results of the XGBoost Model.

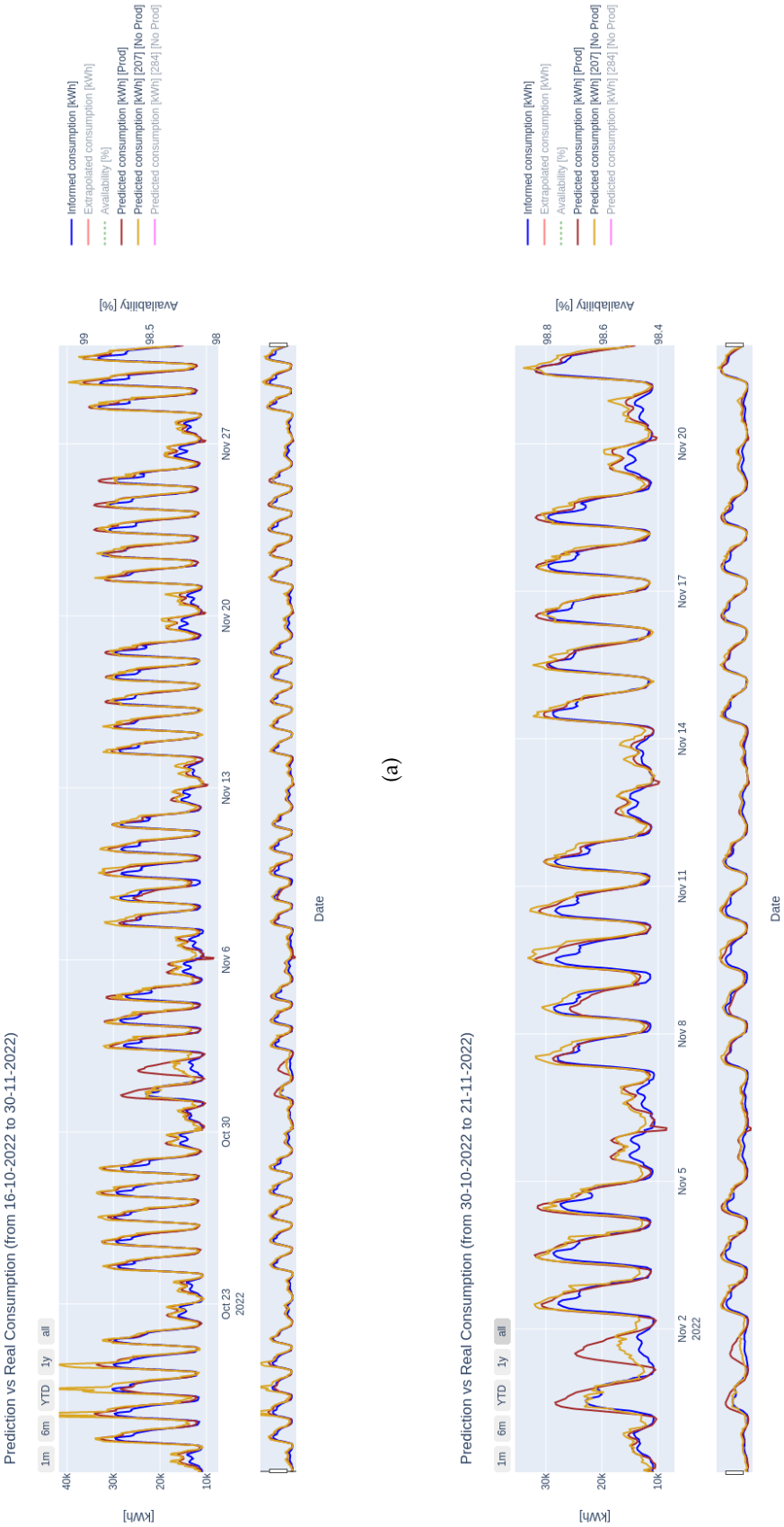
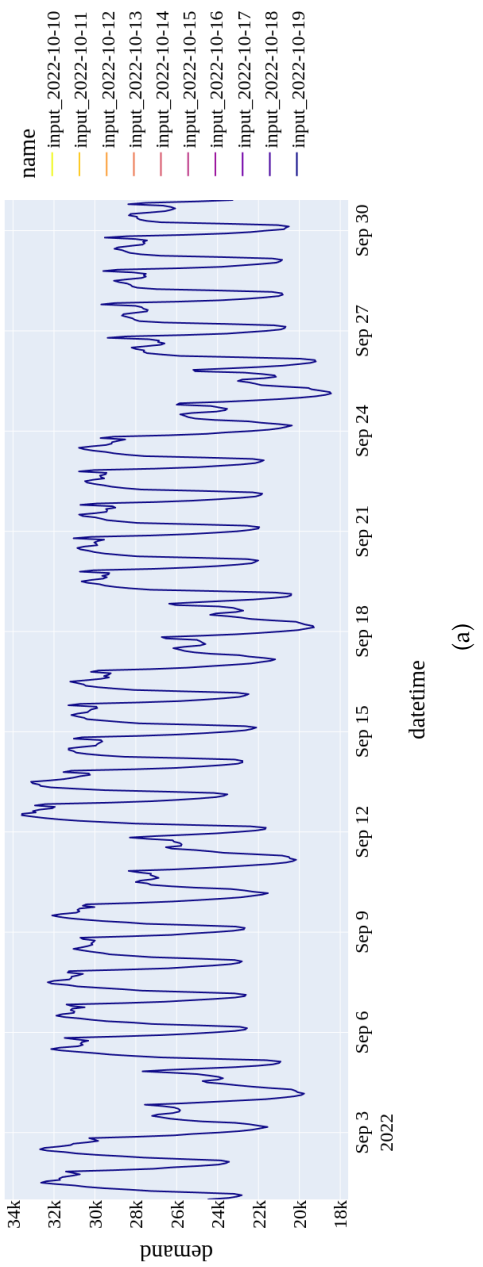


Figure A.1: Current Holaluz predictions per model on test dataset.

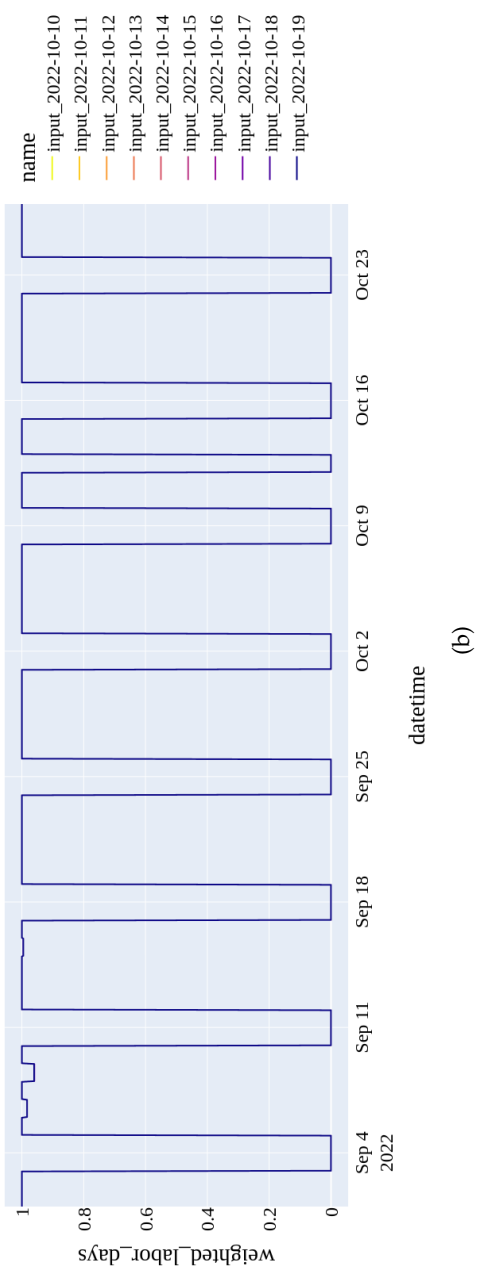
Appendix B

Inputs analysis over time

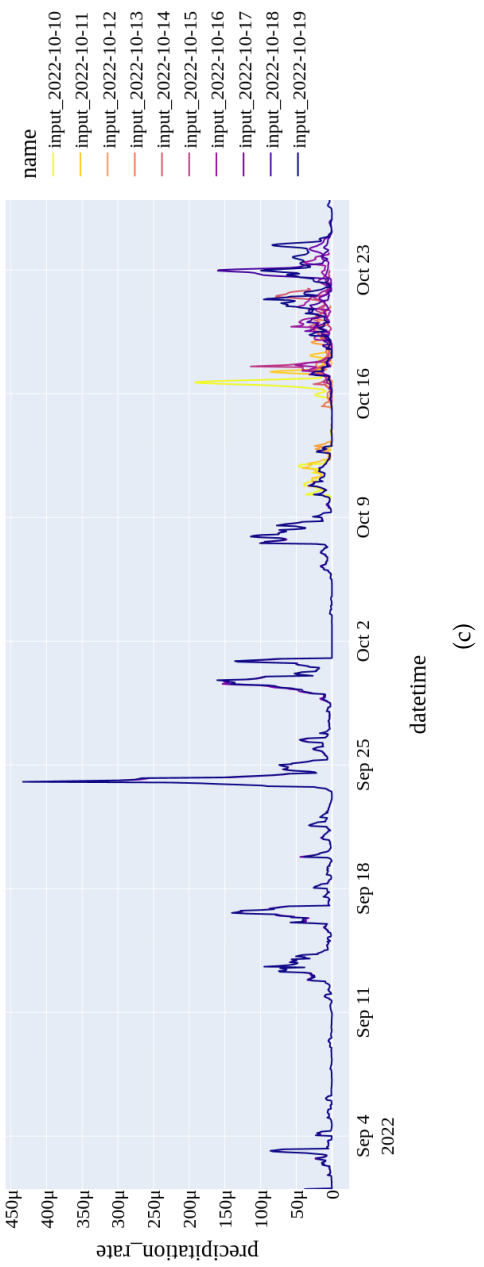
The following figures show per each variable the different values that are recorded every day to the system. Note that some variables change every day and other values remain always the same.


name
input_2022-10-10
input_2022-10-11
input_2022-10-12
input_2022-10-13
input_2022-10-14
input_2022-10-15
input_2022-10-16
input_2022-10-17
input_2022-10-18
input_2022-10-19
demand
34k
32k
30k
28k
26k
24k
22k
20k
18k
Sep 3
2022
Sep 6
Sep 9
Sep 12
Sep 15
Sep 18
Sep 21
Sep 24
Sep 27
Sep 30
datetime

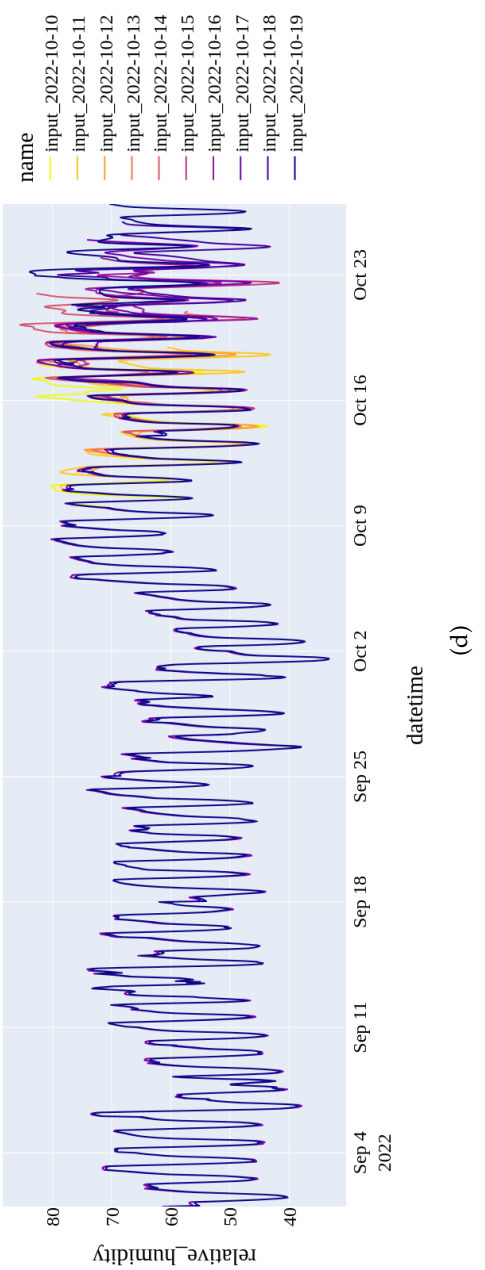
(a)


name
input_2022-10-10
input_2022-10-11
input_2022-10-12
input_2022-10-13
input_2022-10-14
input_2022-10-15
input_2022-10-16
input_2022-10-17
input_2022-10-18
input_2022-10-19
weighted_labor_days
1
0.8
0.6
0.4
0.2
0
Sep 4
2022
Sep 11
Sep 18
Sep 25
Oct 2
Oct 9
Oct 16
Oct 23
datetime

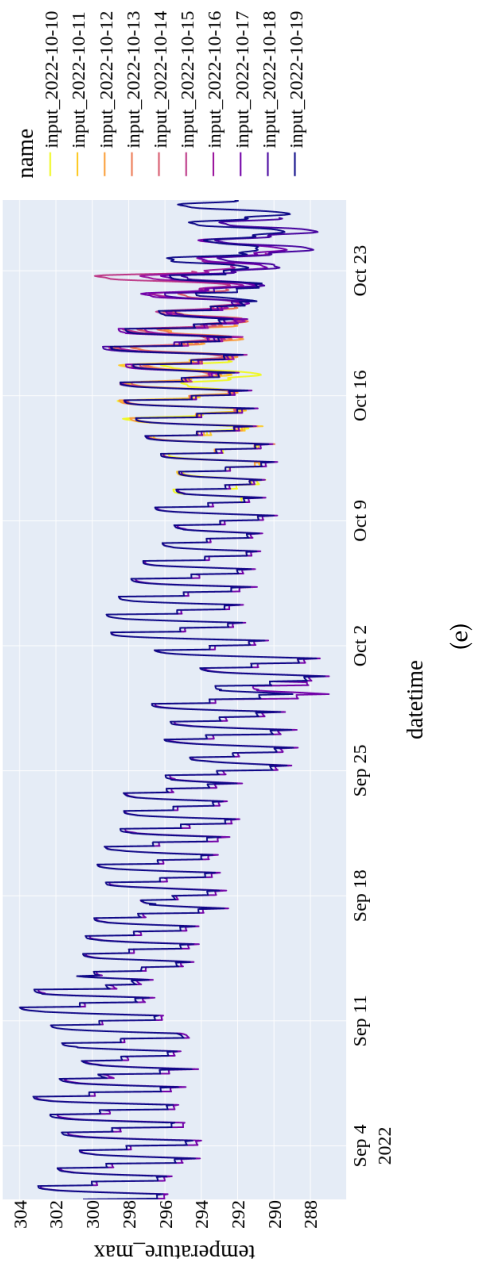
(b)


name
input_2022-10-10
input_2022-10-11
input_2022-10-12
input_2022-10-13
input_2022-10-14
input_2022-10-15
input_2022-10-16
input_2022-10-17
input_2022-10-18
input_2022-10-19
precipitation_rate
0
50µ
100µ
150µ
200µ
250µ
300µ
350µ
400µ
450µ
Sep 4
2022
Sep 11
Sep 18
Sep 25
Oct 2
Oct 9
Oct 16
Oct 23
datetime

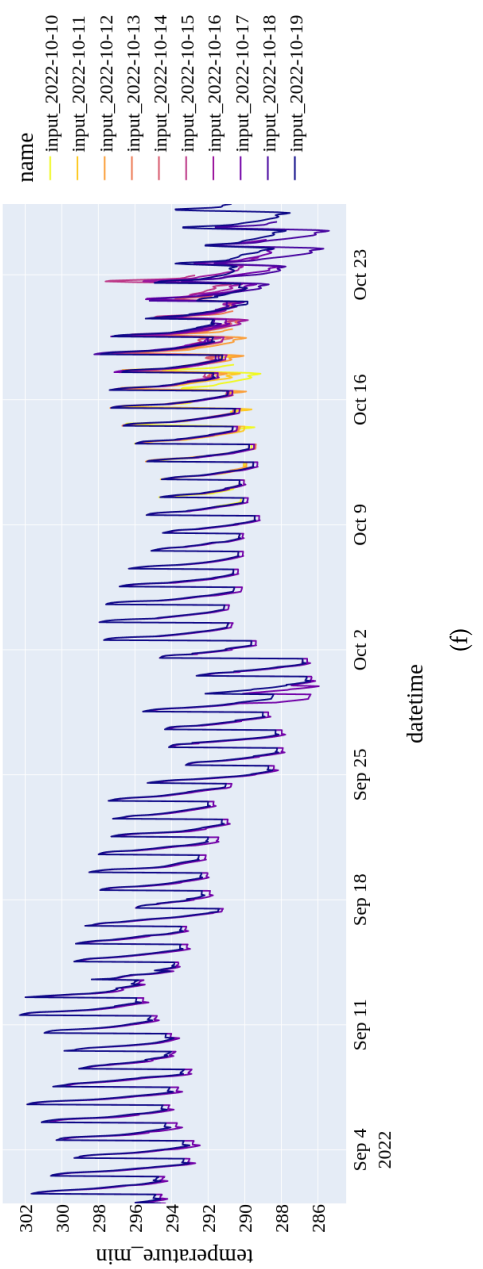
(c)


name
input_2022-10-10
input_2022-10-11
input_2022-10-12
input_2022-10-13
input_2022-10-14
input_2022-10-15
input_2022-10-16
input_2022-10-17
input_2022-10-18
input_2022-10-19
relative_humidity
40
50
60
70
80
Sep 4
2022
Sep 11
Sep 18
Sep 25
Oct 2
Oct 9
Oct 16
Oct 23
datetime

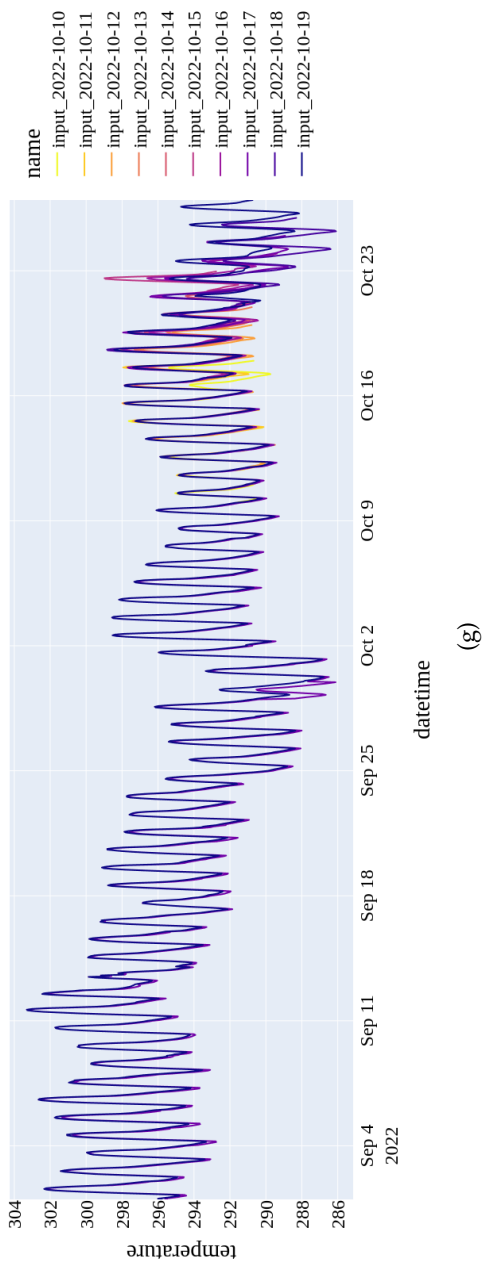
(d)


name
input_2022-10-10
input_2022-10-11
input_2022-10-12
input_2022-10-13
input_2022-10-14
input_2022-10-15
input_2022-10-16
input_2022-10-17
input_2022-10-18
input_2022-10-19
temperature_max
304
302
300
298
296
294
292
290
288
Sep 4
2022
Sep 11
Sep 18
Sep 25
Oct 2
Oct 9
Oct 16
Oct 23
datetime

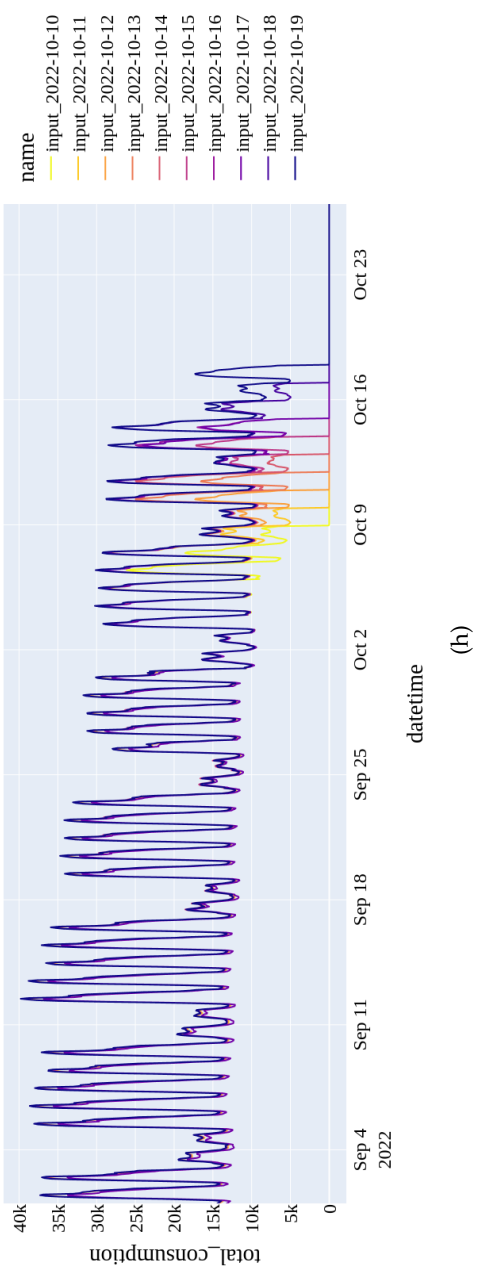
(e)


name
input_2022-10-10
input_2022-10-11
input_2022-10-12
input_2022-10-13
input_2022-10-14
input_2022-10-15
input_2022-10-16
input_2022-10-17
input_2022-10-18
input_2022-10-19
temperature_min
302
300
298
296
294
292
290
288
286
Sep 4
2022
Sep 11
Sep 18
Sep 25
Oct 2
Oct 9
Oct 16
Oct 23
datetime

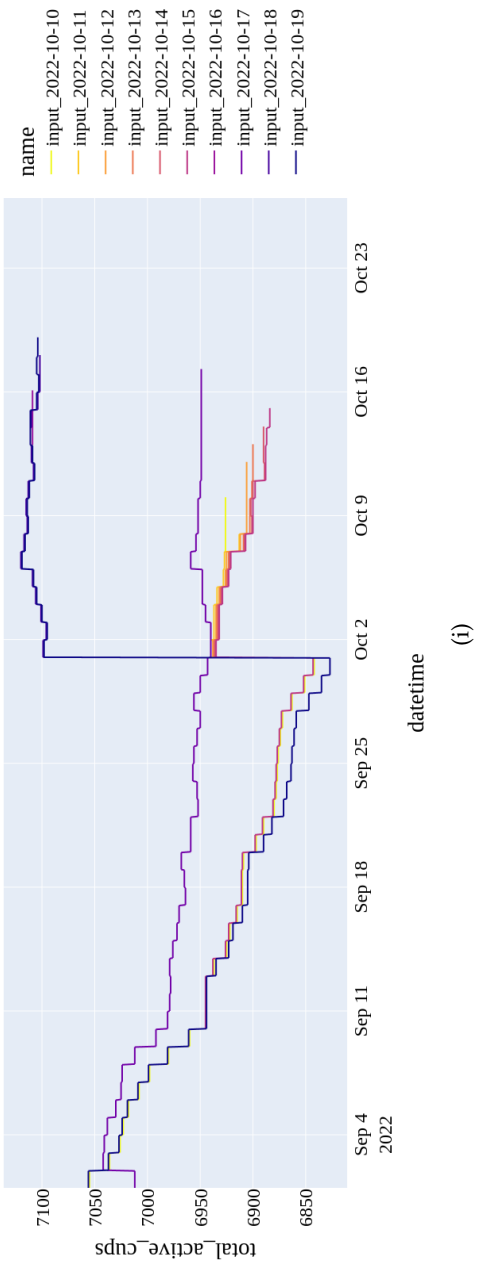
(f)


name
input_2022-10-10
input_2022-10-11
input_2022-10-12
input_2022-10-13
input_2022-10-14
input_2022-10-15
input_2022-10-16
input_2022-10-17
input_2022-10-18
input_2022-10-19
temperature
304
302
300
298
296
294
292
290
288
286
Sep 4
2022
Sep 11
Sep 18
Sep 25
Oct 2
Oct 9
Oct 16
Oct 23
datetime

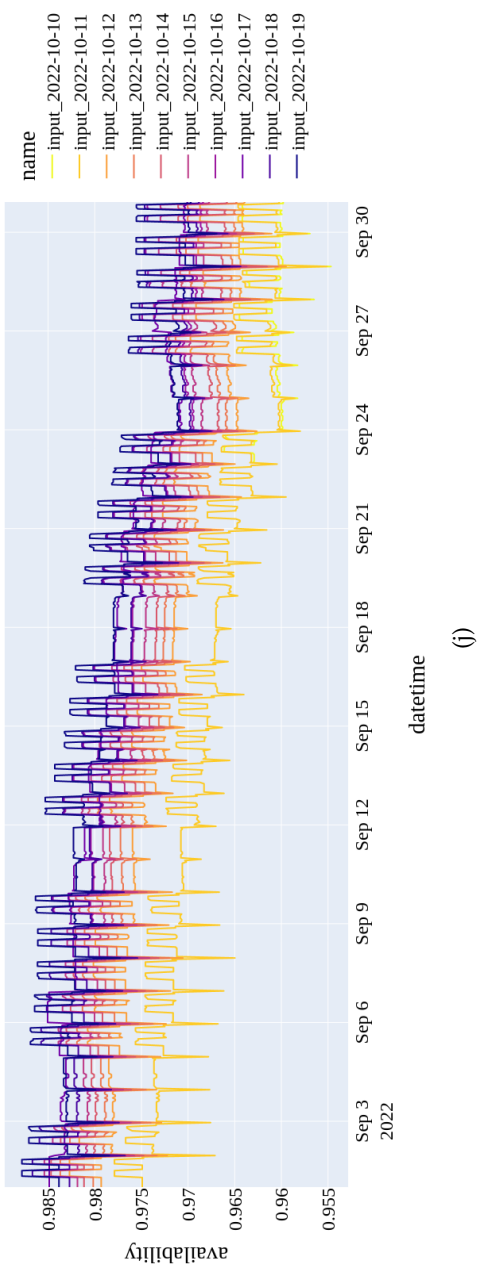
(g)


name
input_2022-10-10
input_2022-10-11
input_2022-10-12
input_2022-10-13
input_2022-10-14
input_2022-10-15
input_2022-10-16
input_2022-10-17
input_2022-10-18
input_2022-10-19
total_consumption
40k
35k
30k
25k
20k
15k
10k
5k
0
Sep 4
2022
Sep 11
Sep 18
Sep 25
Oct 2
Oct 9
Oct 16
Oct 23
datetime

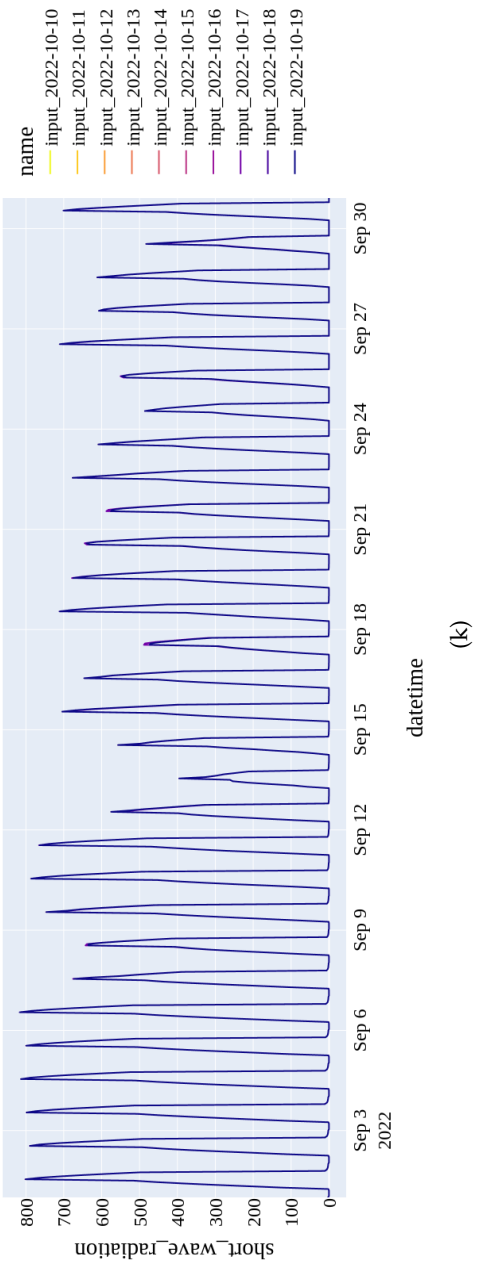
(h)


name
input_2022-10-10
input_2022-10-11
input_2022-10-12
input_2022-10-13
input_2022-10-14
input_2022-10-15
input_2022-10-16
input_2022-10-17
input_2022-10-18
input_2022-10-19
total_active_cups
7100
7050
7000
6950
6900
6850
Sep 4
2022
Sep 11
Sep 18
Sep 25
Oct 2
Oct 9
Oct 16
Oct 23
datetime

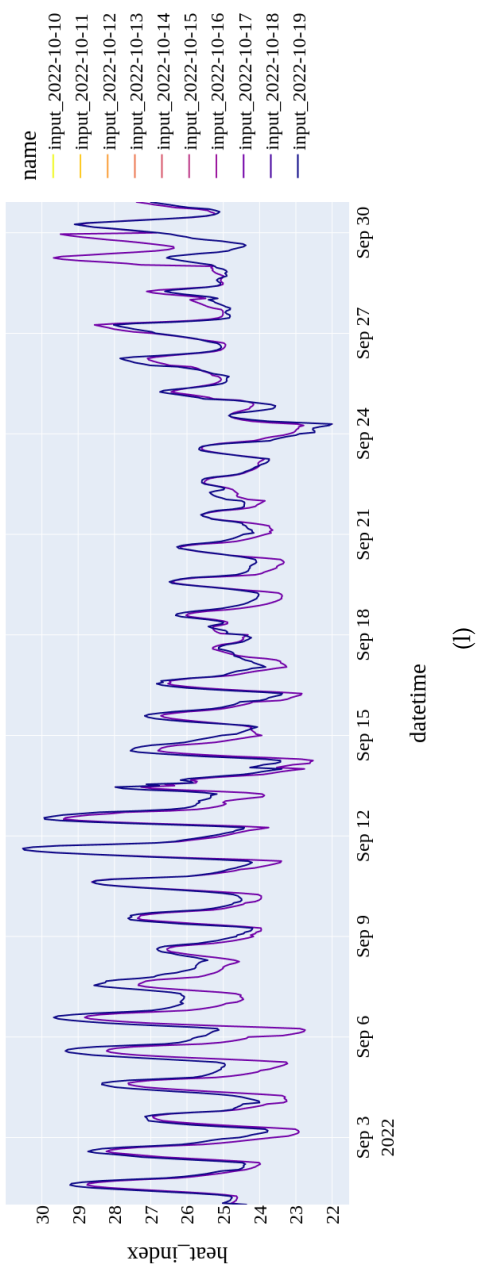
(i)


name
input_2022-10-10
input_2022-10-11
input_2022-10-12
input_2022-10-13
input_2022-10-14
input_2022-10-15
input_2022-10-16
input_2022-10-17
input_2022-10-18
input_2022-10-19
availability
0.985
0.98
0.975
0.97
0.965
0.96
0.955
Sep 3
2022
Sep 6
Sep 9
Sep 12
Sep 15
Sep 18
Sep 21
Sep 24
Sep 27
Sep 30
datetime

(j)


name
input_2022-10-10
input_2022-10-11
input_2022-10-12
input_2022-10-13
input_2022-10-14
input_2022-10-15
input_2022-10-16
input_2022-10-17
input_2022-10-18
input_2022-10-19
short_wave_radiation
0
100
200
300
400
500
600
700
800
Sep 3
2022
Sep 6
Sep 9
Sep 12
Sep 15
Sep 18
Sep 21
Sep 24
Sep 27
Sep 30
datetime

(k)


name
input_2022-10-10
input_2022-10-11
input_2022-10-12
input_2022-10-13
input_2022-10-14
input_2022-10-15
input_2022-10-16
input_2022-10-17
input_2022-10-18
input_2022-10-19
heat_index
22
23
24
25
26
27
28
29
30
Sep 3
2022
Sep 6
Sep 9
Sep 12
Sep 15
Sep 18
Sep 21
Sep 24
Sep 27
Sep 30
datetime

(l)

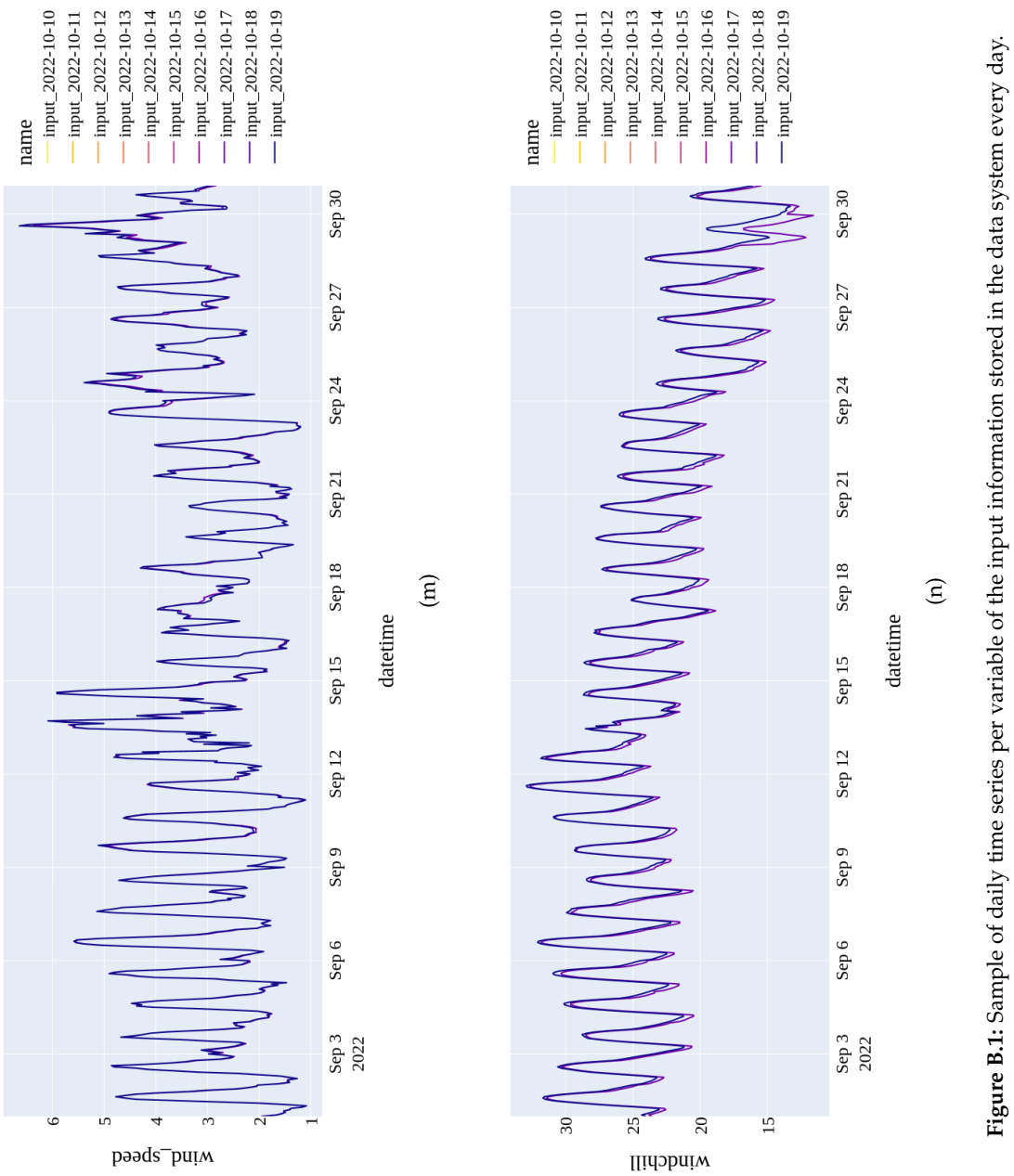


Figure B.1: Sample of daily time series per variable of the input information stored in the data system every day.

Appendix C

Predictions and results of all models setup

The predictions of each model and setup are presented in the following plots, Figures C.1 and C.2. Note that the color indicates from lighter to darker, from the worst to the best model.

Figure C.1: Interval of the test time series of the observed *total_consumption* (red) and predictions (green) of the all models per algorithm: (a) Prophet (b) XGBoost.

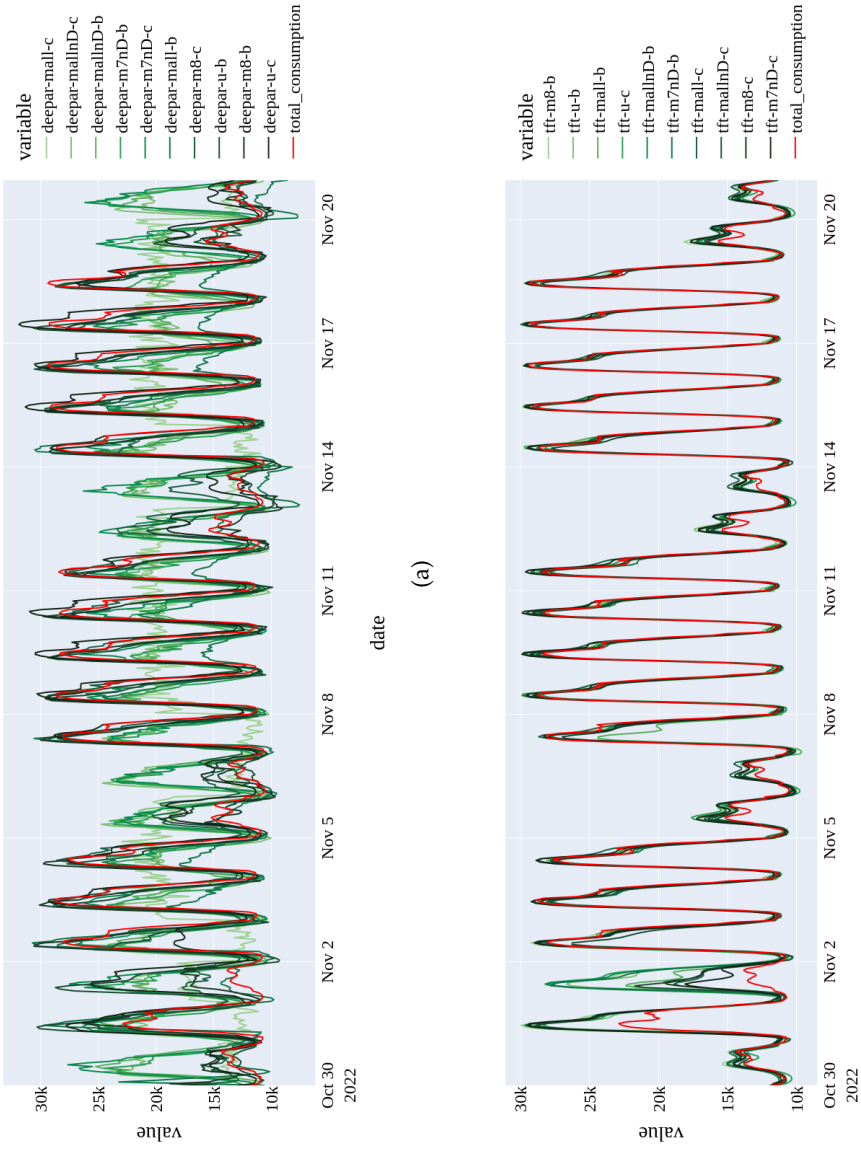


Figure C.2: Interval of the test time series of the observed *total_consumption* (red) and predictions (green) of all models per algorithm: (a) DeepAR and (b) TFT.

Bibliography

M. A. Alotaibi. Machine learning approach for short-term load forecasting using deep neural network. *Energies*, 15(17):6261, Aug. 2022. doi: 10.3390/en15176261.

A. Alrasheedi and A. Almalaq. Hybrid deep learning applied on saudi smart grids for short-term load forecasting. *Mathematics*, 10(15):2666, July 2022. doi: 10.3390/math10152666.

A. N. Baraldi and C. K. Enders. An introduction to modern missing data analyses. *Journal of School Psychology*, 48(1):5–37, Feb. 2010. doi: 10.1016/j.jsp.2009.10.001.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

G. Box and G. M. Jenkins. *The Theory of Autoregressive Time Series*. Holden-Day, 1976a.

G. Box and G. M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1976b.

S. Brubacher and G. T. Wilson. Interpolating time series with application to the estimation of holiday effects on electricity demand. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 25(2):107–116, 1976.

C. Chatfield. *The analysis of time series*. Chapman & Hall/CRC Texts in Statistical Science. Chapman and Hall, London, England, 1989.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

J. Connor. A robust neural network filter for electricity demand prediction. *Journal of Forecasting*, 15(6):437–458, 1996.

H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, Mar. 2019. doi: 10.1007/s10618-019-00619-1.

Z. Fazlipour, E. Mashhour, and M. Joorabian. A deep model for short-term load forecasting applying a stacked autoencoder based on lstm supported by a multi-stage attention mechanism. *Applied Energy*, 327:120063, 2022. ISSN 0306-2619. doi: 10.1016/j.apenergy.2022.120063.

E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 134:75–88, 2019. ISSN 0743-7315. doi: 10.1016/j.jpdc.2019.07.007.

S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, Jan. 1992. doi: 10.1162/neco.1992.4.1.1.

M. Gokce and E. Duman. Performance comparison of simple regression, random forest and xgboost algorithms for forecasting electricity demand, 2022.

A. González-Briones, G. Hernández, J. M. Corchado, S. Omatu, and M. S. Mohamad. Machine learning models for electricity consumption forecasting: A review. In *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, pages 1–6, 2019. doi: 10.1109/CAIS.2019.8769508.

J. D. Hamilton. *Time Series Analysis*. Princeton University Press, Dec. 1994. doi: 10.1515/9780691218632.

D. M. Hernández, L. H. Callejo, M. Solís, Ángel Zorita Lamadrid, Óscar Duque Perez, L. G. Morales, and F. S. García. A data-driven forecasting strategy to predict continuous hourly energy demand in smart buildings. *Applied Sciences*, 11(17), 2021. ISSN 2076-3417. doi: 10.3390/app11177886.

C. E. Holt. *Forecasting Seasonal and Trends by Exponentially Weighted Moving Averages*. Carnegie Institute of Technology, Pittsburgh PA, 1957.

R. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Australia, 2nd edition, 2018.

R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: the forecast package for r. *Journal of Statistical Software*, 27(3), 2008. doi: 10.18637/jss.v027.i03.

A. Jain. Macro forecasting using alternative data. In *Handbook of US Consumer Economics*, pages 273–327. Elsevier, 2019. doi: 10.1016/b978-0-12-813524-2.00011-1.

P. D. Jong. A cross-validation filter for time series models. *Biometrika*, 75(3):594–600, 1988.

D. C. D. Kirkpatrick and J. A. Dahlquist. *Technical Analysis: The Complete Resource for Financial Market Technicians*. Tim Moore, 2010. ISBN 978-0-13-705944-7.

R. V. Klyuev, I. D. Morgoev, A. D. Morgoeva, O. A. Gavrina, N. V. Martyushev, E. A. Efremenkov, and Q. Mengxu. Methods of forecasting electric energy consumption: A literature review. *Energies*, 15(23):8919, Nov. 2022. doi: 10.3390/en15238919.

T. Krehbiel. Correlation coefficient rule of thumb. *Decision Sciences Journal of Innovative Education*, 2:97–100, 01 2004. doi: 10.1111/j.0011-7315.2004.00025.x.

M. Lehna, F. Scheller, and H. Herwartz. Forecasting day-ahead electricity prices: A comparison of time series and neural network models taking external regressors into account. *Energy Economics*, 106:105742, Feb. 2022. doi: 10.1016/j.eneco.2021.105742.

C. Li, X. Zheng, Z. Yang, and L. Kuang. Predicting short-term electricity demand by combining the advantages of ARMA and XGBoost in fog computing environment. *Wireless Communications and Mobile Computing*, 2018:1–18, 2018. doi: 10.1155/2018/5018053.

B. Lim, S. O. Arik, N. Loeff, and T. Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting, 2019.

D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning, 2015.

F. Mahia, A. R. Dey, M. A. Masud, and M. S. Mahmud. Forecasting electricity consumption using ARIMA model. In *2019 International Conference on Sustainable Technologies for Industry 4.0 (STI)*. IEEE, Dec. 2019. doi: 10.1109/sti47673.2019.9068076.

S. Makridakis, E. Spiliotis, and V. Assimakopoulos. The m4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4):802–808, 2018. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2018.06.001.

M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2 edition, 2018. ISBN 978-0-262-03940-6.

A. Mosavi and A. Bahmani. Energy consumption prediction using machine learning; a review, 2019.

C. Nichiforov, I. Stamatescu, I. Fagarasan, and G. Stamatescu. Energy consumption forecasting using ARIMA and neural network models. In *2017 5th International Symposium on Electrical and Electronics Engineering (ISEEE)*. IEEE, Oct. 2017. doi: 10.1109/iseee.2017.8170657.

A. Nugaliyadde, U. Somaratne, and K. W. Wong. Predicting electricity consumption using deep recurrent neural networks, 2019.

E. Parzen. An approach to time series modeling and forecasting illustrated by hourly electricity demands, 01 1976.

S. Rafayal, M. Çevik, and D. Kici. An empirical study on probabilistic forecasting for predicting city-wide electricity consumption. *Proceedings of the Canadian Conference on Artificial Intelligence*, 05 2022. doi: 10.21428/594757db.8e8477a9.

Y. Ruan, G. Wang, H. Meng, and F. Qian. A hybrid model for power consumption forecasting using VMD-based the long short-term memory neural network. *Frontiers in Energy Research*, 9, Jan. 2022. doi: 10.3389/fenrg.2021.772508.

F. D. Rueda, J. D. Suárez, and A. del Real Torres. Short-term load forecasting using encoder-decoder WaveNet: Application to the french grid. *Energies*, 14(9):2524, Apr. 2021. doi: 10.3390/en14092524.

A. Salam and A. E. Hibaoui. Comparison of machine learning algorithms for the power consumption prediction : - case study of tetouan city –. In *2018 6th International Renewable and Sustainable Energy Conference (IRSEC)*, pages 1–5, 2018. doi: 10.1109/IRSEC.2018.8703007.

D. Salinas, V. Flunkert, and J. Gasthaus. Deepar: Probabilistic forecasting with autoregressive recurrent networks, 2017.

R. H. Shumway and D. S. Stoffer. *Time Series Analysis and Its Applications*. Springer International Publishing, 2017. doi: 10.1007/978-3-319-52452-8.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

S. Sridevi, S. Rajaram, C. Parthiban, S. SibiArasan, and C. Swadhikar. Imputation for the analysis of missing values and prediction of time series data. In *2011 International Conference on Recent Trends in Information Technology (ICRTIT)*, pages 1158–1163, 2011. doi: 10.1109/ICRTIT.2011.5972466.

S. Taylor and B. Letham. Forecasting at scale. *The American Statistician*, 72, 09 2017. doi: 10.1080/00031305.2017.1380080.

J. Tukey. *Exploratory Data Analysis*. Massachusetts: Addison-Wesley, 1977. ISBN 978-0-13-705944-7.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

V. S. Vineeth, H. Kusetogullari, and A. Boone. Forecasting sales of truck components: A machine learning approach. In *2020 IEEE 10th International Conference on Intelligent Systems (IS)*. IEEE, Aug. 2020. doi: 10.1109/is48319.2020.9200128.

W. Wang, Y. Shi, G. Lyu, and W. Deng. Electricity consumption prediction using XGBoost based on discrete wavelet transform. *DEStech Transactions on Computer Science and Engineering*, Oct. 2017. doi: 10.12783/dtcse/aiea2017/15003.

P. R. Winters. *Forecasting sales by exponentially weighted moving averages.* Carnegie Institute of Technology, Pittsburgh PA, 1960.

G. Zhang, C. Wei, C. Jing, and Y. Wang. Short-term electrical load forecasting based on time augmented transformer. *International Journal of Computational Intelligence Systems*, 15(1), Aug. 2022. doi: 10.1007/s44196-022-00128-y.

Z. Zhao, C. Xia, L. Chi, X. Chang, W. Li, T. Yang, and A. Y. Zomaya. Short-term load forecasting based on the transformer model. *Information*, 12(12):516, Dec. 2021. doi: 10.3390/info12120516.