UCLouvain

École polytechnique de Louvain

Time series forecasting in Odoo

Author : **Antoine Balthazar**
Supervisor : **Axel Legay**
Readers : **Charles Pecheur**, **Arnold Moyaux**, **Serena Lucca**
Academic year 2022–2023
Master [120] in Computer Science and Engineering

Abstract

Time series forecasting is a popular branch in data analysis that allows to detect possible trends or seasonal patterns in an ordered list of values and enables to predict future values. This thesis presents the implementation of an automatic time series forecasting algorithm in the Odoo software and, more specifically, an automatic exponential smoothing forecasting algorithm. Automatic means that the model is trained automatically and the parameters of the model are automatically computed. The main goal of this algorithm is to predict future demand based on recorded sales, and with this to extend a feature in Odoo that schedules the future production or replenishment of a company's products. The thesis contains a theoretical review of time series and exponential smoothing methods, a description of the algorithm, and validation tests to show the accuracy of the predictions.

Acknowledgements

Firstly, I would like to express my deepest gratitude to my thesis supervisor, Prof. Axel Legay, for his guidance and advice, for offering the possibility of working in collaboration with a real company on a practical project, and for his availability.

I am also immensely grateful to my supervisor at Odoo company, Arnold Moyaux, for finding a very interesting topic. His advice helped me to dive into the Odoo framework more easily. His proposals and suggestions have been crucial to the successful completion of this thesis.

I would also like to thank Odoo company for providing the possibility to realize a thesis in their company and for accepting me at their office one day a week. I can confirm the good reputation of the food at Odoo.

Then, I would also like to acknowledge the people who proofreaded, namely, my godparent Jean-Christophe, my uncle Vincent and finally Audrey and Vincent.

Furthermore, I would like to thank the readers and members of the jury, Prof. Charles Pecheur and Serena Lucca for giving the time to review this master thesis.

Finally, I am grateful to my family for their unwavering support, and encouragement throughout this year.

Contents

Chapter 1

Introduction

Time series forecasting is a branch of data analysis that focuses on predicting future values based on historical patterns and trends. It is a powerful tool used in various fields, including finance, economics, weather forecasting, stock market analysis, sales forecasting, and many more. The primary objective of time series forecasting is to make accurate predictions about future data points, enabling companies and researchers to make informed decisions and plan accordingly.

A time series is a collection of observations or data points taken at regular or irregular intervals over time. These observations could be recorded daily, weekly, monthly, or at any other defined intervals. Time series data often exhibit certain patterns, such as trends, seasonality, and cyclicality. By analyzing these patterns and relationships within the data, time series forecasting methods can provide valuable insights into future behavior and help for anticipating changes or fluctuations.

In this master thesis study case, we will try to implement a robust time series forecasting algorithm in Odoo software to predict future demand based on recorded sales. The algorithm must be able to identify the patterns cited previously and must be applicable to any company's product. Of course, as Odoo users are not supposed to be time series forecasting experts, the algorithm must automatically adapt its parameters between the different time series analyzed. Therefore, this is called an automatic time series forecasting algorithm.

There are several approaches to time series forecasting, ranging from simple techniques like moving averages and exponential smoothing to more advanced methods such as autoregressive integrated moving average (ARIMA), and machine learning algorithms like recurrent neural networks (RNNs). Considering the difficulty of implementing such algorithms, we have to select a technique at the beginning of the thesis. Therefore, we decide to focus on exponential smoothing, and this choice is discussed in the next chapters.

The thesis is structured as follows:

- In Chapter 2, the task specifications as given by Odoo are described.
- In Chapter 3, a theoretical review of what a time series is and an introduction to time series forecasting is provided.
- In Chapter 4, the exponential smoothing model to forecast time series is explained. All the steps and difficulties of the automatic modeling and forecasting process are covered.
- In Chapter 5, the implementation architecture is explained, as well as the final results in the Odoo application.
- In Chapter 6, validation tests are performed to assess first the accuracy of the automatic exponential smoothing model selection, and then the accuracy of the forecasting points and confidence interval.
- In Chapter 7, possible future improvements are introduced and a conclusion of this thesis is done.

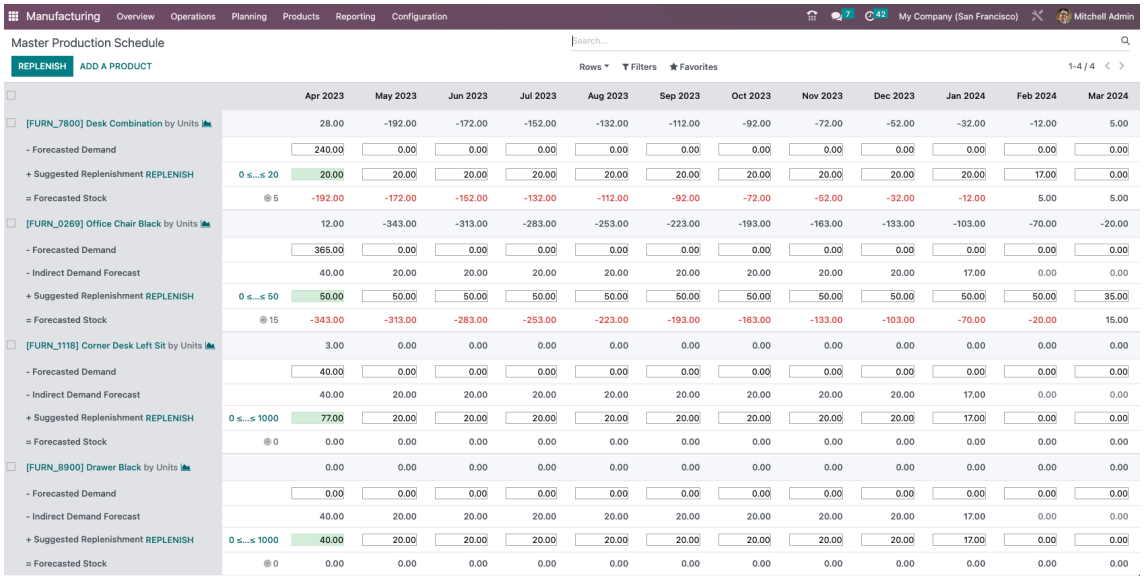
Chapter 2

Specifications

Firstly, let us present the specifications of the task. The main goal is to add a new functionality to the Odoo Master Production Schedule (MPS) feature.

2.1 Context: introduction to the MPS

The MPS is located in the manufacturing module, also called the MRP module. Odoo [1] describes it as: "The Master Production Schedule (MPS) is a valuable tool for planning your production based on your demand forecast". It is presented as a large table with many information and number inputs displayed (Fig. 2.1).



		Apr 2023	May 2023	Jun 2023	Jul 2023	Aug 2023	Sep 2023	Oct 2023	Nov 2023	Dec 2023	Jan 2024	Feb 2024	Mar 2024
[FURN_7800] Desk Combination by Units		28.00	-192.00	-172.00	-152.00	-132.00	-112.00	-92.00	-72.00	-52.00	-32.00	-12.00	5.00
- Forecasted Demand		240.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
+ Suggested Replenishment REPLENISH	0 ≤...≤ 20	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	17.00	0.00
= Forecasted Stock	5	-192.00	-172.00	-152.00	-132.00	-112.00	-92.00	-72.00	-52.00	-32.00	-12.00	5.00	5.00
[FURN_0269] Office Chair Black by Units		12.00	-343.00	-313.00	-283.00	-253.00	-223.00	-193.00	-163.00	-133.00	-103.00	-70.00	-20.00
- Forecasted Demand		365.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
- Indirect Demand Forecast		40.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	17.00	0.00	0.00
+ Suggested Replenishment REPLENISH	0 ≤...≤ 50	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	35.00
= Forecasted Stock	15	-343.00	-313.00	-283.00	-253.00	-223.00	-193.00	-163.00	-133.00	-103.00	-70.00	-20.00	15.00
[FURN_1118] Corner Desk Left Sit by Units		3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
- Forecasted Demand		40.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
- Indirect Demand Forecast		40.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	17.00	0.00	0.00
+ Suggested Replenishment REPLENISH	0 ≤...≤ 1000	77.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	17.00	0.00	0.00
= Forecasted Stock	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
[FURN_8900] Drawer Black by Units		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
- Forecasted Demand		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
- Indirect Demand Forecast		40.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	17.00	0.00	0.00
+ Suggested Replenishment REPLENISH	0 ≤...≤ 1000	40.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	17.00	0.00	0.00
= Forecasted Stock	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 2.1: MPS main view

The columns represent the production period of the product. This period is editable by the user. A group of several sub-rows is dedicated to a specific product. The user can select which product to show. Each sub-row displays a specific quantity related to the product. There exist several different quantities. The most important are:

- **Starting inventory** is the current quantity in the inventory of a product.
- **Demand Forecasting** is the planned demand for the period.
- **Indirect Demand Forecasting** is the indirect planned demand for the period. If a product relies on another product to be built, the demand for the main product adds an indirect demand in the second product.
- **Suggested Replenishment** is the quantity suggested to replenish based on the safety stock target (the safety stock is the minimum stock to have in inventory to face an unpredictable peak of demand), the minimum and maximum quantity to replenish set by the user.
- **Forecasted Stock** is the resulting stock after adding replenishment and subtracting the forecast demand.
- **Actual Demand Y-1 and Y-2** show the demand for the same period one or two year ago.

The user has the possibility to choose the sub-row to display. All the possible quantities are shown in Figure 2.2.

Figure 2.2: Possible quantities to display

When the user changes the quantity in a cell, all related cells are recomputed based on the change. There can exist a relation between different products. For example, a given final product can require other products to be built. The user can quickly see if he will have enough stock to satisfy all the forecasted demand. A more detailed view of some products is shown in Figure 2.3.

		Apr 2023	May 2023
[FURN_7800] Desk Combination by Units		28.00	-192.00
- Forecasted Demand		240.00	0.00
+ Suggested Replenishment REPLENISH	0 ≤...≤ 20	20.00	20.00
= Forecasted Stock	5	-192.00	-172.00
[FURN_0269] Office Chair Black by Units		12.00	-343.00
- Forecasted Demand		365.00	0.00
- Indirect Demand Forecast		40.00	20.00
+ Suggested Replenishment REPLENISH	0 ≤...≤ 50	50.00	50.00
= Forecasted Stock	15	-343.00	-313.00

Figure 2.3: MPS product detailed view

2.2 Objective of the task

Presently, the main limitation when using the tool is that the user must enter the forecast demand himself. It is obviously difficult and time-consuming for a vendor to manually plan the future demand. It is even more if the company sells many different products. Therefore, the main objective of the task is to implement a feature that aims to automatically compute the forecast demand based on past sales recorded.

2.3 Functional specifications

The first functional need is to give and show the user an estimate of the forecast demand for a chosen product. This forecast must be based on previous sales and take into account the possible seasonal trend or global trend.

Next to this main requirement, there are also several additional constraints.

- First, in the MPS the user can change the period of production. There exist three different periods: monthly, weekly, and daily. The algorithm must automatically adapt to the chosen period.
- Another point of attention is the time complexity; it must remain in $O(n)$, where n is the number of periods recorded in the past, to avoid computational time to become excessive for the product that has existed for a long time.

- Another important constraint is to not use an external library for the implementation. It is only possible to import the basic Python library.

Chapter 3

Introduction to Time series

A time series is simply a series of data points ordered in time [2]. There exist two types of time series :

- Time series where data are spaced at **regular** time intervals
- Time series where data are spaced at **irregular** time intervals

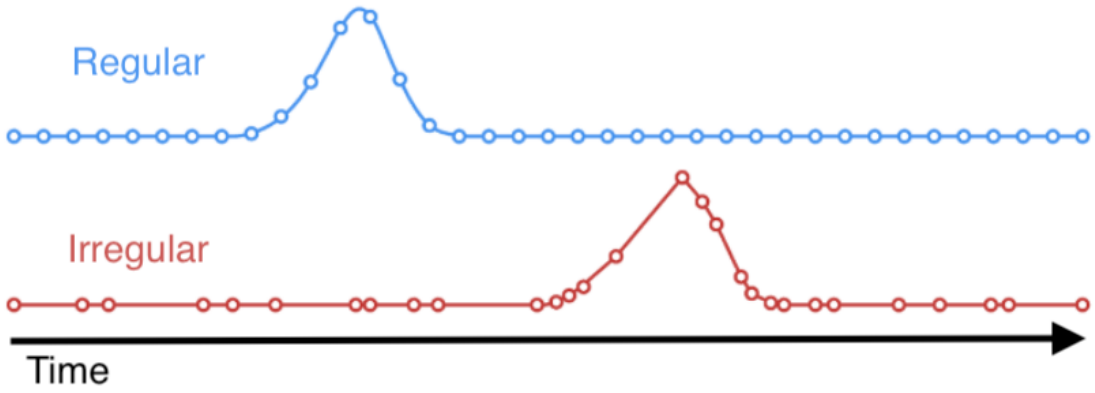


Figure 3.1: Regular and irregular time series [3]

A regular time series is easier to model because we are sure to have enough data anytime.

3.1 Time series decomposition

Any time series can be decomposed into four components [4] [5]. These components can be defined as

- **trend component** (T): The long-term direction of the series
- **seasonal component** (S): A pattern that repeats with a known periodicity (e.g, 12 months per year or 7 days per week)
- **cycle component** (C): A pattern that repeats with an unknown and changing periodicity

- **error component** (E): The unpredictable component of the series

As in [5], the cycle component will be ignored and subsumed within the trend component. We can see the graphical representation of the decomposition in Figure 3.2

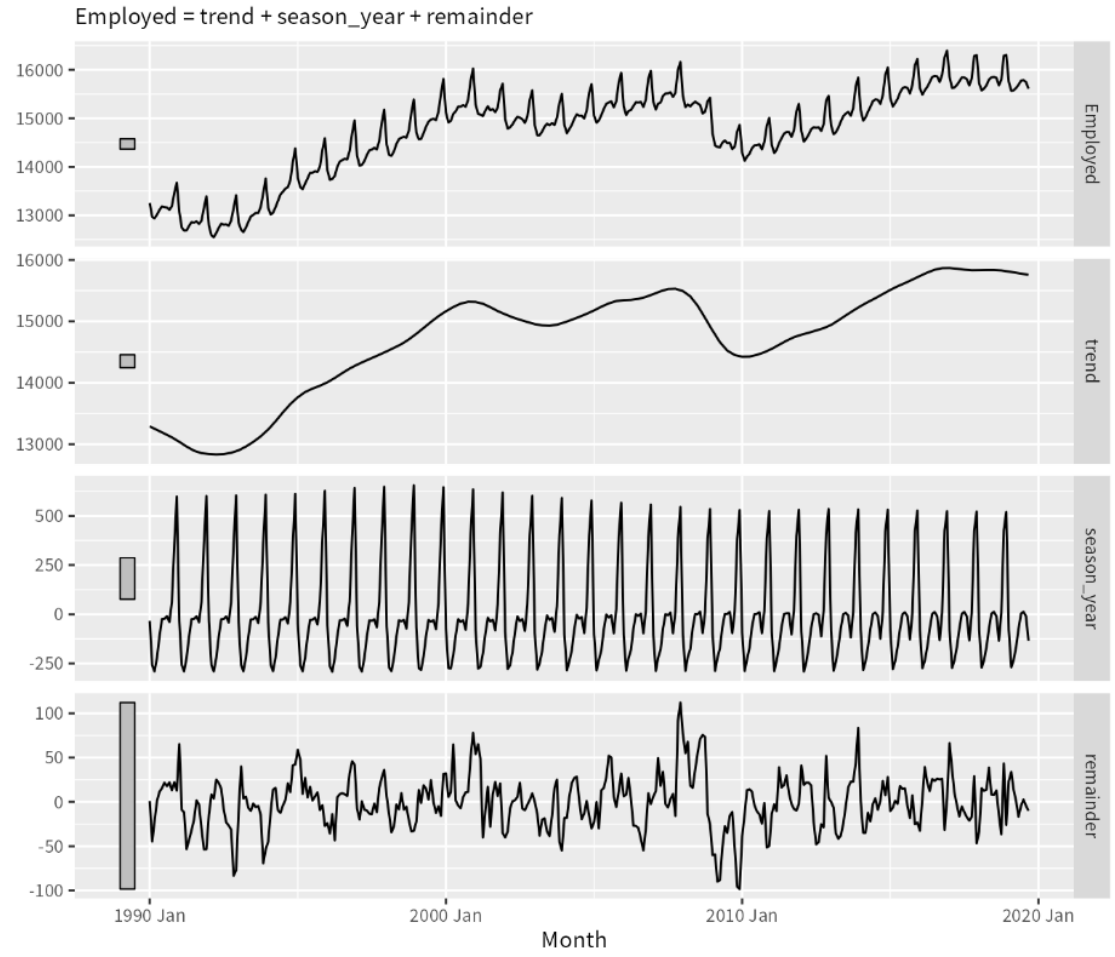


Figure 3.7: The total number of persons employed in US retail (top) and its three additive components.

Figure 3.2: Time series decomposition [4]

We can combine these three components in many different ways. If we assume an additive decomposition, then we can write:

$$Y = T + S + E \tag{3.1}$$

where Y is the original time series. Alternatively, if we assume a multiplicative decomposition:

$$Y = T \times S \times E \tag{3.2}$$

Sometimes, one of the components of the time series might be multiplicative, while the others are additive:

$$Y = (T + S) \times E \tag{3.3}$$

3.2 Time series forecasting

Time series forecasting is to try to produce future values of a time series based on the previous. A model is built with the recorded data, and then we predict the next values with this model.
y represent the time series and y_t is a data at time t. The notation used to denote the predicted future value at time t+h when the value at time t is known is as follows:

$$\hat{y}_{t+h|t} \tag{3.4}$$

There exist many techniques to model and forecast a time series. Some of the most known are briefly introduced below [6].

Naive, **SNaïve**: The naïve model is relatively simple; any forecast value corresponds to the last observed value.

$$\hat{y}_{t+h|t} = \hat{y}_t \tag{3.5}$$

The **SNaïve** is an extension of the naïve model. We assume that we know a certain seasonal component of period T, and the forecasts are given by:

$$\hat{y}_{t+h|t} = \hat{y}_{t+(h\%T)-T} \tag{3.6}$$

These two models are often too simple, but easy to implement.

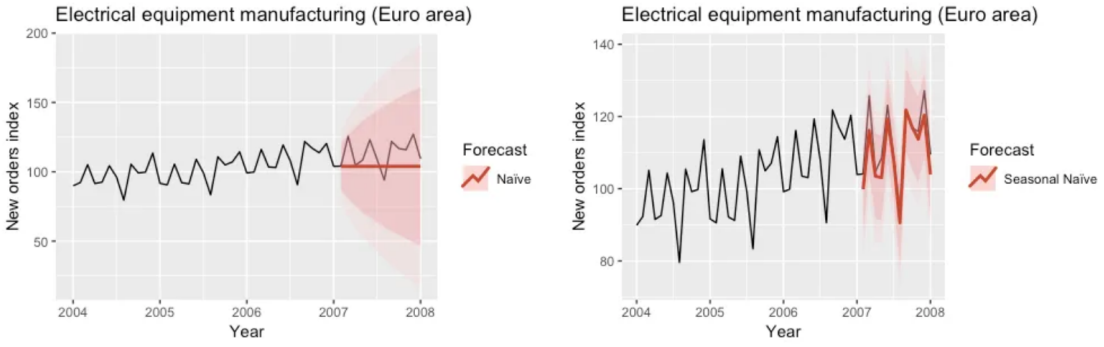


Figure 3.3: Naïve and SNaïve model forecast example [6]

ARIMA, **SARIMA**: ARIMA (AutoRegressive Integreted Moving Average) is among the most used approaches for time series forecasting. The model uses a linear combination of past values (AutoRegressive) and a linear combination of forecast error (Moving Average). This technique requires stationary time series (time series in which the mean remains constant, there is no trend component or seasonal component). It is possible to transform a non-stationary time series to a stationary one with the differencing operation (Integrating) (subtract the time series by the same time series shifted of one step).
The SARIMA model is the ARIMA extended by a linear combination of seasonal past values.

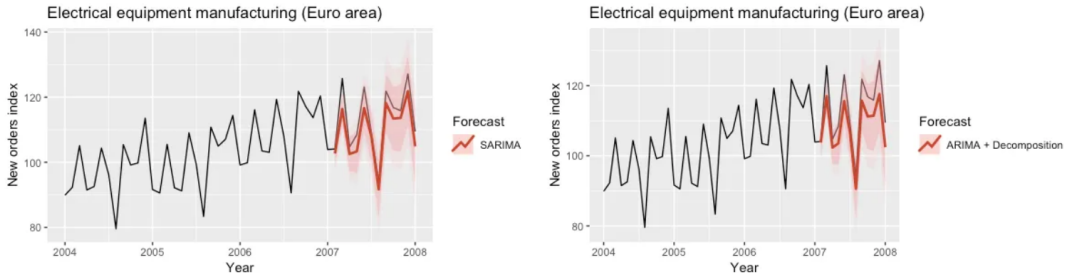


Figure 3.4: ARIMA and SARIMA model forecast example [6]

Exponential Smoothing (ES): this technique is largely used too; ES is a family of models which will be described in more detail in the next section. We can see ES as a weighted average of past observations, and the corresponding weights decrease exponentially as we go back in time [6].

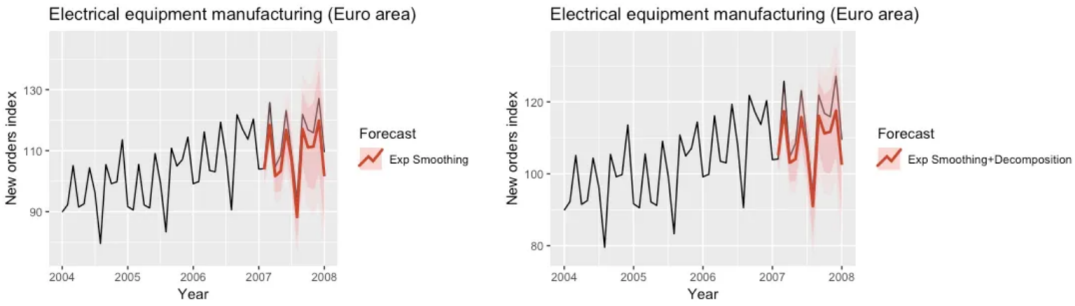


Figure 3.5: Exponential Smoothing model forecast example [6]

These three forecasting models are statistical models, but there also exist many models based on machine learning techniques. We have, for example, traditional machine learning methods such as kernel regression [7] and support vector regression [8] [9] or deep learning methods such as DeepAR [10], Transformer [11], WaveNet [12]. A very interesting and recent paper [13] attempts to compare forecasting methods. Indeed, time series forecasting has remained for a long time as one of the rare domains in which classical statistical models outperformed ML models. But according to [13], ML methods have just started to overcome the traditional methods, especially for energy, stock market and inventory demand, where both long series and explanatory variables are available to facilitate learning. However, this improvement is still not sufficient in other specific domains. One of the advantages of ML techniques is to be more generic (e.g. no need to identify the period of seasonality), but that becomes a disadvantage when there are few data available.

Given the high complexity and the necessary time to implement a single technique without external libraries, a selection of a specific technique has been made before the implementation. This choice is the **Exponential smoothing** method and can be explained for several reasons. Firstly, why a statistical model instead of an ML model? For this task, we need a robust model that can handle situations where there are few data (for example, when a product has recently been launched). The constraint to not use an external library is also a reason to choose a simpler model. Then, we need to choose between the (S)ARIMA model and the ES model. One of the main challenges of the task is to make the computation of the model automatic. The user should not have to define or select the model parameters. These parameters must be automatically estimated. Taking this into account, it is more difficult to implement an automatic forecasting algorithm for the ARIMA model than for the ES model [14]. Finally, another advantage of ES models is that they can be non-linear, unlike ARIMA models, which remain only linear. The next step is to understand in more detail the exponential smoothing family.

Chapter 4

Exponential Smoothing Models

This chapter is largely inspired by the book Forecasting with Exponential Smoothing, the state space approach [5].

4.1 Introduction to Exponential Smoothing

To explain ES, let us take the simplest form: **Simple Exponential Smoothing**.
Before forecasting data, the first step is to **model the time series**. This is done by smoothing the curve of the time series. Therefore, the random (or error) component is removed. The equation behind this smoothing is a recursive equation defined as:

$$\hat{y}_{t+1} = \hat{y}_t + \alpha(y_t - \hat{y}_t) \tag{4.1}$$

where $\hat{y}_t$ is the estimated (or modeled) value at time t, y_t the real value at time t and α a constant between 0 and 1.
The new estimated value is simply the old estimated value plus an adjustment for the error in the last step. The recursive equation can also be seen as:

$$\hat{y}_{t+1} = \alpha y_t + \alpha(1-\alpha)y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + ... + \alpha(1-\alpha)^{t-1} y_1 + (1-\alpha)\hat{y}_1 \tag{4.2}$$

The more the data are far from time t, the more their influence decreases, and this decrease is controlled by α. In fact, the weights in front of the y_t terms decrease exponentially. We can see that at the end of the formula, the $\hat{y}_1$ term must be estimated. The problem of choosing the starting value will be discussed later. We can see in figure 4.1 how a curve is smoothed depending on the alpha factor.

The next step is to **forecast** future values based on the just built model. For the Simple Exponential Smoothing, the forecast formula is simple:

$$\hat{y}_{t+h|t} = \hat{y}_{t+1}, \qquad h = 2, 3, ... \tag{4.3}$$

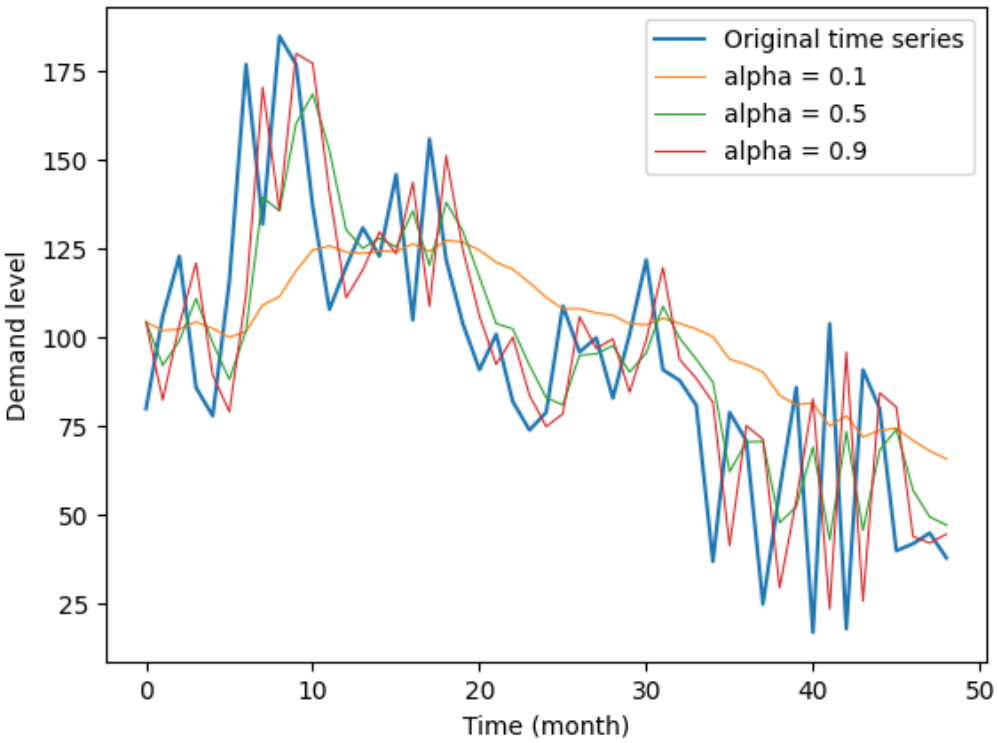


Figure 4.1: exponential smoothing modeling example for different alpha

The forecast is simply the estimate point just after the last real data available. Of course, this simple method cannot model all possible time series and does not take into account a possible seasonal pattern or global trend. That is why there are many other methods in the ES family.

Now, we must define several new different components in the ES equations. We have first a **level term** (l) and a **growth term** (b). Together, they express the trend component of the time series 3.1. Then we have the **seasonal term** (s) that expresses the current season component. Each term has its own parameter α (α for l, β for b, and γ for s). The terms can be combined in different ways, and it can be None (N) (simply not combined), Additive (A) or Multiplicative (M). The level term is always present. This gives nine different methods regrouped into Table 4.1.

Trend component	Seasonal component		
	N (None)	A (Additive)	M (Multiplicative)
N (None)	N,N	N,A	N,M
A (Additive)	A,N	A,A	A,M
M (Multiplicative)	M,N	M,A	M,M

Table 4.1: Table of the ES methods

Some methods in this table also have another historical name. As we have seen previously,

the cell (N,N) is originally called Simple Exponential Smoothing. Another well-known method is the cell (A,N) called Holt's linear method. We also have additive and multiplicative Holt-Winters methods given by cells (A, A) and (A, M), respectively. Each of the methods is described by a recursive equation for each component (to model the time series) and one forecast equation (to forecast the time series).

To illustrate a first method, we can choose the (A,N) method. It is defined by the following equations:

$$\text{Level:} \qquad l_t = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}), \tag{4.4}$$

$$\text{Growth:} \qquad b_t = \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1}, \tag{4.5}$$

$$\text{Forecast:} \qquad \hat{y}_{t+h|t} = l_t + b_t h. \tag{4.6}$$

where α and β^* are constant values between 0 and 1. *Note that* the reason for using β^* instead of β will be explained later. The first two equations are used to model the time series. Initial terms l_0 and b_0 must be estimated (as explained later). Once the time series is modeled and the terms l_t and b_t are calculated (where t is the time of the last available data), we can use the forecast equation to predict data h step later. This method can model time series with a trend component.

Then if we take the method (A,M) to express the seasonal component:

$$\text{Level:} \qquad l_t = \alpha \frac{y_t}{s_{t_m}} + (1-\alpha)(l_{t-1} + b_{t-1}), \tag{4.7}$$

$$\text{Growth:} \qquad b_t = \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1}, \tag{4.8}$$

$$\text{Seasonal:} \qquad s_t = \gamma \frac{y_t}{(l_{t-1} + b_{t-1})} + (1-\gamma)s_{t-m}, \tag{4.9}$$

$$\text{Forecast:} \qquad \hat{y}_{t+h|t} = (l_t + b_t h)s_{t-m+h_m^+}. \tag{4.10}$$

where m is the length of seasonality (e.g., the number of months or quarters in a year), and $h_m^+ = [(h-1) \bmod m] + 1$. This method is useful for modeling a time series with an additive trend component and a multiplicative seasonal component.
We can do the same for all methods. All formulas are shown in Table 4.2. Note that it also includes two other ways to combine the trend term, called additive damped (A_d)) and multiplicative damped (M_d)). These methods are used to express a decrease in trend slope in longer forecast horizons. It is especially useful for longer forecast horizons, which is not particularly the case in this context.

4.2 State Space Models

Many of the techniques presented in table 4.1 are known for a long time. The true added value of the book [5] is the definition of **state space models** that regroup all techniques into one equation system.

Trend	Seasonal		
	N	A	M
N	$\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}$ $\hat{y}_{t+h\|t} = \ell_t$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)\ell_{t-1}$ $s_t = \gamma(y_t - \ell_{t-1}) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t + s_{t-m+h_m^+}$	$\ell_t = \alpha(y_t / s_{t-m}) + (1-\alpha)\ell_{t-1}$ $s_t = \gamma(y_t / \ell_{t-1}) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t s_{t-m+h_m^+}$
A	$\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $\hat{y}_{t+h\|t} = \ell_t + hb_t$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $s_t = \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t + hb_t + s_{t-m+h_m^+}$	$\ell_t = \alpha(y_t / s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $s_t = \gamma(y_t / (\ell_{t-1} + b_{t-1})) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = (\ell_t + hb_t)s_{t-m+h_m^+}$
A_d	$\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$ $\hat{y}_{t+h\|t} = \ell_t + \phi_h b_t$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$ $s_t = \gamma(y_t - \ell_{t-1} - \phi b_{t-1}) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t + \phi_h b_t + s_{t-m+h_m^+}$	$\ell_t = \alpha(y_t / s_{t-m}) + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$ $s_t = \gamma(y_t / (\ell_{t-1} + \phi b_{t-1})) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = (\ell_t + \phi_h b_t)s_{t-m+h_m^+}$
M	$\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}b_{t-1}$ $b_t = \beta^*(\ell_t / \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $\hat{y}_{t+h\|t} = \ell_t b_t^h$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)\ell_{t-1}b_{t-1}$ $b_t = \beta^*(\ell_t / \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $s_t = \gamma(y_t - \ell_{t-1}b_{t-1}) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t b_t^h + s_{t-m+h_m^+}$	$\ell_t = \alpha(y_t / s_{t-m}) + (1-\alpha)\ell_{t-1}b_{t-1}$ $b_t = \beta^*(\ell_t / \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $s_t = \gamma(y_t / (\ell_{t-1}b_{t-1})) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t b_t^h s_{t-m+h_m^+}$
M_d	$\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}b_{t-1}^{\phi}$ $b_t = \beta^*(\ell_t / \ell_{t-1}) + (1-\beta^*)b_{t-1}^{\phi}$ $\hat{y}_{t+h\|t} = \ell_t b_t^{\phi_h}$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)\ell_{t-1}b_{t-1}^{\phi}$ $b_t = \beta^*(\ell_t / \ell_{t-1}) + (1-\beta^*)b_{t-1}^{\phi}$ $s_t = \gamma(y_t - \ell_{t-1}b_{t-1}^{\phi}) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t b_t^{\phi_h} + s_{t-m+h_m^+}$	$\ell_t = \alpha(y_t / s_{t-m}) + (1-\alpha)\ell_{t-1}b_{t-1}^{\phi}$ $b_t = \beta^*(\ell_t / \ell_{t-1}) + (1-\beta^*)b_{t-1}^{\phi}$ $s_t = \gamma(y_t / (\ell_{t-1}b_{t-1}^{\phi})) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t b_t^{\phi_h} s_{t-m+h_m^+}$

In each case, ℓ_t denotes the series level at time t, b_t denotes the slope at time t, s_t denotes the seasonal component of the series at time t, and m denotes the number of seasons in a year; α, β^*, γ and ϕ are constants, $\phi_h = \phi + \phi^2 + \cdots + \phi^h$ and $h_m^+ = \left[(h-1) \bmod m\right] + 1$.

Figure 4.2: Formulas for recursive calculations and forecast points [5]

Before describing this state space, we must talk about the error in a method. Indeed, each method actually gives two models, one with additive errors and the other with multiplicative errors. We can add a letter 'A' or 'M' to form a triplet (A,*,* or M,*,*) and identify a model. We will see that the two models will lead to the exact same prediction points but different confidence intervals.

We can now specify the general model with the state vector $\boldsymbol{x}_t = (l_t, b_t, s_t, s_{t-1}, ..., s_{t-m+1})$. The vector contains the level and trend terms of the current step, since the equations require only the previous step. It also contains all the m season terms, as we must keep them in the state to retrieve them m step later. The state space equations take the form:

$$y_t = w(\boldsymbol{x}_{t-1}) + r(\boldsymbol{x}_{t-1})\epsilon_t, \tag{4.11}$$

$$\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_{t-1}) + \boldsymbol{g}(\boldsymbol{x}_{t-1})\epsilon_t \tag{4.12}$$

where in the first equation, y_t denotes the observed value at time t, $r(\cdot)$ and $w(\cdot)$ are scalar functions. The term $w(\boldsymbol{x}_{t-1})$ represents the effect of the past on y_t and the $r(\boldsymbol{x}_{t-1})\epsilon_t$ term is the unpredictable part of y_t (ϵ_t is a white noise process with variance σ^2). The second equation, known as the transition equation, described how the state vector evolves over time. $\boldsymbol{f}(\cdot)$ and $\boldsymbol{g}(\cdot)$ are vector functions. The term $\boldsymbol{f}(\boldsymbol{x}_{t-1})$ shows the effect of the past on the current state $\boldsymbol{x}_t$ and the term $\boldsymbol{g}(\boldsymbol{x}_{t-1})\epsilon_t$ shows the unpredictable change in $\boldsymbol{x}_t$.

For the forecast equation, it can be defined in function of the vector $\boldsymbol{x}_t$ of the model,

$$\hat{y}_{t+h|t}(\boldsymbol{x}_t) = ... \tag{4.13}$$

All models can be expressed with the equations described above, and we will show one example of this transformation.

Let us take the equations 4.4 and 4.5 that define the (A,A,N) method and transform them in terms of w, r, $\boldsymbol{f}$, $\boldsymbol{g}$ and $\boldsymbol{x}$.
First, we can form y_t by combining the component (additive trend, no season and additive error).

$$y_t = l_{t-1} + b_{t-1} + \epsilon_t \tag{4.14}$$

Then by putting y_t in the l_t equation we get,

$$l_t = l_{t-1} + b_{t-1} + \alpha\epsilon_t \tag{4.15}$$

and by putting l_t into the b_t equation,

$$b_t = b_{t-1} + \beta\epsilon_t \tag{4.16}$$

where $\beta = \alpha\beta^*$ to simplify the notation.

We can now regroup these three equations into the state space equation by defining,

$$\boldsymbol{x}_t = \begin{pmatrix} l_t \\ b_t \end{pmatrix} \tag{4.17}$$

$$w(\boldsymbol{x}_{t-1}) = l_{t-1} + b_{t-1}, \tag{4.18}$$

$$r(\boldsymbol{x}_{t-1}) = 1, \tag{4.19}$$

$$\boldsymbol{f}(\boldsymbol{x}_{t-1}) = \begin{pmatrix} l_{t-1} + b_{t-1} \\ b_{t-1} \end{pmatrix} \tag{4.20}$$

$$\boldsymbol{g}(\boldsymbol{x}_{t-1}) = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{4.21}$$

$$\tag{4.22}$$

The forecast equation is taken from Table 4.2 and is,

$$\hat{y}_{t+h|t}(\boldsymbol{x}_t) = l_t + hb_t \tag{4.23}$$

The state space equations of all additive error models are written in Table 4.3. We can easily retrieve the four functions w, r, $\boldsymbol{f}$, $\boldsymbol{g}$ from them. For the multiplicative error, we can use the same equations with two small changes. Firstly, the function $r(\boldsymbol{x_t})$ that is 1 for additive error models is equal to $w(\boldsymbol{x_t})$ for multiplicative error models. And the function $\boldsymbol{g}$ returns a vector where each value is multiplied by $w(\boldsymbol{x_t})$. Changes are summarized in Table 4.2.

	Additive error	Multiplicative error
$r(\boldsymbol{x_t})$	1	$w(\boldsymbol{x_t})$
$\boldsymbol{g}(\boldsymbol{x_t})$	$\boldsymbol{g}(\boldsymbol{x_t})$	$\boldsymbol{g}(\boldsymbol{x_t})$. $w(\boldsymbol{x_t})$

Table 4.2: Changes between additive error models and multiplicative error models

For now, the steps to do forecasting are as follows.

- Model the time series with a chosen method and the corresponding vector $\boldsymbol{x}_t$ and the functions $w, r, \boldsymbol{f}, \boldsymbol{g}$.
- Forecast with the vector $\boldsymbol{x}_t$ obtained after the modeling.

However, several problems still need to be discussed, such as estimating the parameters and initial state, choosing the model, and calculating the confidence intervals.

4.3 Initialization of x_0 and Estimation of Parameters

Initialization of x_0

One of the remaining problems is the **initialisation of the state vector** $\boldsymbol{x}_t$ at time 0 ($\boldsymbol{x}_0$) that contains all the initials components ($l_0, b_0, s_0, s_1, ..., s_m$). [15] proposed a heuristic to initialize the

Trend	Seasonal		
	N	A	M
N	$y_t = l_{t-1} + \epsilon_t$ $\boldsymbol{x}_t = \begin{pmatrix} l_{t-1} + \alpha\epsilon_t \end{pmatrix}$	$y_t = l_{t-1} + s_{t-m} + \epsilon_t$ $\boldsymbol{x}_t = \begin{pmatrix} l_{t-1} + \alpha\epsilon_t \\ s_{t-m} + \gamma\epsilon_t \\ s_{t-m+1} \\ ... \\ s_{t-1} \end{pmatrix}$	$y_t = l_{t-1}s_{t-m} + \epsilon_t$ $\boldsymbol{x}_t = \begin{pmatrix} l_{t-1} + \alpha\epsilon_t/s_{t-m} \\ s_{t-m} + \gamma\epsilon_t/l_{t-1} \\ s_{t-m+1} \\ ... \\ s_{t-1} \end{pmatrix}$
A	$y_t = l_{t-1} + b_{t-1} + \epsilon_t$ $\boldsymbol{x}_t = \begin{pmatrix} l_{t-1} + b_{t-1} + \alpha\epsilon_t \\ b_{t-1} + \beta\epsilon_t \end{pmatrix}$	$y_t = l_{t-1} + b_{t-1} + s_{t-m} + \epsilon_t$ $\boldsymbol{x}_t = \begin{pmatrix} l_{t-1} + b_{t-1} + \alpha\epsilon_t \\ b_{t-1} + \beta\epsilon_t \\ s_{t-m} + \gamma\epsilon_t \\ s_{t-m+1} \\ ... \\ s_{t-1} \end{pmatrix}$	$y_t = (l_{t-1} + b_{t-1})s_{t-m} + \epsilon_t$ $\boldsymbol{x}_t = \begin{pmatrix} l_{t-1} + b_{t-1} + \alpha\epsilon_t/s_{t-m} \\ b_{t-1} + \beta\epsilon_t/s_{t-m} \\ s_{t-m} + \gamma\epsilon_t/l_{t-1} \\ s_{t-m+1} \\ ... \\ s_{t-1} \end{pmatrix}$
M	$y_t = l_{t-1}b_{t-1} + \epsilon_t$ $\boldsymbol{x}_t = \begin{pmatrix} l_{t-1}b_{t-1} + \alpha\epsilon_t \\ b_{t-1} + \beta\epsilon_t/l_{t-1} \end{pmatrix}$	$y_t = l_{t-1}b_{t-1} + s_{t-m} + \epsilon_t$ $\boldsymbol{x}_t = \begin{pmatrix} l_{t-1}b_{t-1} + \alpha\epsilon_t \\ b_{t-1} + \beta\epsilon_t/l_{t-1} \\ s_{t-m} + \gamma\epsilon_t \\ s_{t-m+1} \\ ... \\ s_{t-1} \end{pmatrix}$	$y_t = l_{t-1}b_{t-1}s_{t-m} + \epsilon_t$ $\boldsymbol{x}_t = \begin{pmatrix} l_{t-1}b_{t-1} + \alpha\epsilon_t/s_{t-m} \\ b_{t-1} + \beta\epsilon_t/(l_{t-1}s_{t-m}) \\ s_{t-m} + \gamma\epsilon_t/(l_{t-1}b_{t-1}) \\ s_{t-m+1} \\ ... \\ s_{t-1} \end{pmatrix}$

Table 4.3: State space for all additive error models

different components. There are three cases. When there is a season detected in the data and the data contain observations for at least 2 times all the periods of this season (e.g. 24 points for a 12 month in a year season (2 years)):

- **Initial seasonal component**: extracted seasonal components from the data (method explained just after)
- **Initial level component**: we compute a linear regression line with the first ten deseasonalized observations (deseasonalized means that the season component is removed). The initial level l_0 is the coefficient a of the linear regression $a + bt$.
- **Initial trend component**: for additive trend, the initial level b_0 is the coefficient b of the same linear regression $a + bt$ and for multiplicative trend, it is $1 + \frac{b}{a}$

The second case is where there is no season detected in the data, thus there is no need to estimate seasonal components:

- **Initial level component**: The initial level l_0 is the coefficient a from the linear regression $a + bt$ of the first ten observations
- **Initial trend component**: for additive trend, the initial level b_0 coefficient is b from the same linear regression $a + bt$ and for multiplicative trend, it is $1 + \frac{b}{a}$

Finally, when there is a seasonal component but not enough data, the seasonal terms are set to 0, and the other initials terms are the same as in the second case.

A classical method **to extract estimated seasonal components** from the initial values of a time series is explained in [16]. The first step is to calculate a moving average of order m, where m is the number of components of the season. This will give an estimate of the possible trend component. The moving average equation is:

$$\hat{T}_t = \begin{cases} \frac{1}{m} \sum_{j=-\frac{m}{2}}^{\frac{m}{2}} y_{t+j}, & \text{if } t >= \frac{m}{2} \text{ and } t < len(y) - \frac{m}{2} \\ 0, & \text{otherwise} \end{cases} \tag{4.24}$$

In our case where the season component is a month in a year, m = 12, we take only the first 24 months, the moving average equation is:

$$\hat{T}_t = \begin{cases} \frac{1}{12} \sum_{j=-6}^{6} y_{t+j}, & \text{if } t \geq 5 \text{ and } t < 18 \\ 0, & \text{otherwise} \end{cases} \tag{4.25}$$

This gives a vector with the estimated trend for the 24 first points (first 2 complete seasons). Then, we take the original time series and remove these trend components from it to obtain a detrended time series (dty_t).

$$dty_t = y_t - \hat{T}_t \tag{4.26}$$

Finally, we can obtain the estimated seasonal component with this detrended time series by averaging the detrended values corresponding to the same season component. For example, the estimated component of month m is the average of all the detrended values of month m. But in our case, it will be simply the detrended values as there is one estimate value for each season component m. Therefore, we obtain $\hat{S}_t$, which is the estimate season component at time t, by taking the corresponding values from dty_t.

Estimation of the parameters

The other difficulty is the **estimation of the parameters α, β and γ**. To resolve this, we use a **maximum likelihood estimator** obtained in [5]. We want to minimize

$$\mathcal{L}(\theta) = n \log \left(\sum_{t=1}^{n} \epsilon_t^2 \right) + 2 \sum_{t=1}^{2} \log |r(\boldsymbol{x}_{t-1})|. \tag{4.27}$$

where θ contains all the parameters of the model. The constraints imposed on the parameter values are $0 < \alpha < 1$, $0 < \beta < \alpha$ and $0 < \gamma < 1 - \alpha$. This choice is discussed in [5].
As we will test each combination of parameters one by one, the choice of how many combination to test is important. Table 4.4 summarizes the possible number of parameters to estimate that depend on the present components. In Table 4.5, we can see the approximate number of combinations possible as a function of the number of parameters and the precision between two values of the parameters to test. We can observe that the number of combinations can quickly increase if we take a step that is too small for a parameter and if there is more than one component in the time series. The way to calculate this is simply:

$$\text{nbCombinations} = \frac{1}{\text{stepSize}} + \text{has2Param}\frac{1}{2 * \text{stepSize}} + \text{has3Param}\frac{1}{2 * \text{stepSize}} \tag{4.28}$$

where has2Param is equal to 1 if there are two or three parameters and 0 otherwise, has3Param is equal to 1 if there are three parameters and 0 otherwise, and stepSize is the precision between two parameters. The number 2 added in the denominator of the two last fractions express the constraints on the parameters β and γ.

Trend component	Seasonal component		
	N (None)	A (Additive)	M (Multiplicative)
N (None)	1	2	2
A (Additive)	2	3	3
M (Multiplicative)	2	3	3

Table 4.4: number of parameters for each ES method

To optimize the process, we propose something that is not in the [5] book. We split the optimization of all models into two phases. For the first one, we take all the methods and choose

Number of parameters	Step size			
	0.1	0.05	0.03	0.02
1	10	20	30	50
2	50	200	450	1 250
3	250	2 000	6 750	31 250

Table 4.5: number of parameters combination for different step size (parameter precision)

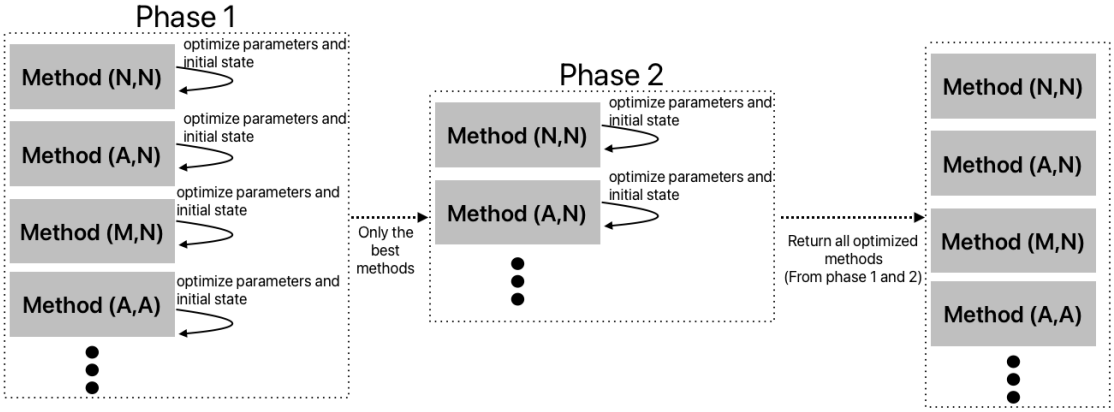


Figure 4.3: Optimization of methods' parameters schema

a relatively low-precision parameter to do a first optimization. We keep only the methods with the most promising results (the best half of the methods). Then, we launch a second phase with an increased precision parameter on these most promising methods to further optimize their parameters. By doing so, we avoid too much computation for methods that do not fit the time series. Obviously, if we choose a precision parameter of 0 for the first phase, all methods are kept in phase 2. Figure 4.3 shows the process. Now we have all our methods optimized and ready for the selection part.

4.4 Model Selection

An important step in the forecasting process is the **selection of the model**. Our algorithm must automatically choose the most appropriate exponential smoothing model. We can divide this process into two steps:

- Choose the right method from the methods in Table 4.1
- Choose a model between additive error and multiplicative error

Method selection

The method selection is a key step in the process. We want to have the best forecast points possible, and the choice of the method used is crucial. To evaluate a method, we first use the maximum

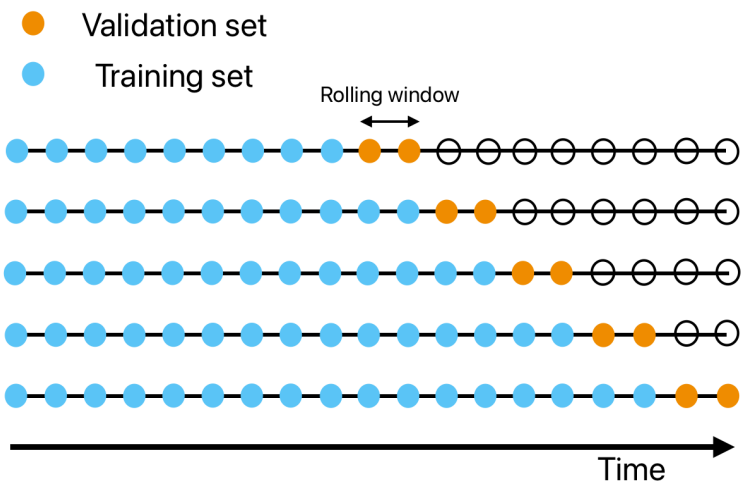


Figure 4.4: Cross-validation for time series

likelihood estimator. We reuse the equation 4.27, but here there is nothing to minimize; we simply obtain a modeling score. With that, we will try to find the method that best fits the time series. But then comes the problem of overfitting. Indeed, we want the best forecast point and not the best modeling. For this reason, we will use a second metric calculated with the well-known **cross-validation** technique.

General cross-validation randomly splits the original data set into a training set and a validation set. Then, it trains the model with the training set and evaluates it with the validation set that contains unseen data. The algorithm repeats the process multiple times and shuffles the data set between each operation. However, this approach does not work for a time series data set. First, the data must remain ordered to respect the time order. Then, the validation set must be subsequent to the training set because we want to forecast points in the future.
Therefore, the cross-validation technique must be adapted to time series data. The way to do it [17] is to not shuffle the data and to take a rolling window for the validation set. The length of this window has been set to 5, to focus the forecast on the closer point. The training set will contain all the data available before this window. The time constraint will be respected, but the negative point is that the size of the training set will differ between experiences. To decrease this effect, we can start the rolling window of the validation set at the middle of the time series to ensure a minimal size for the training set. The principle is represented in Figure 4.4.

The method chosen will be the one that combines good time series modeling and good forecast capability. The choice of the weight of the cross-validation score added to the modeling score is experimented and chosen in Section 6.1.

All methods do not fit for all time series. Time series that contains zeros or negative data

cannot be modeled by a multiplicative component. This could lead to zero division error and instability in the forecast. To prevent these effects, we remove the method with multiplicative component from the set of possible methods if the time series is not strictly positive. Thus, in this case, only four methods will be applicable instead of nine.

Model selection

Once the method is chosen, we must select the error model. It is impossible to do that with the cross-validation technique because we evaluate the forecasting point and these points are strictly equal for the two possible models. To make the choice, we only take the modeling score.

4.5 Summary of the model training

We can summarize the procedure for building the model. First, we optimize the parameters and initialize the vector state in all applicable models. Then we select the best method according to the modeling score and the cross-validation score. Finally, we choose the error model. After that, we have our model ready to forecast points and give the confidence intervals. The schema of this procedure is presented in Figure 4.5.

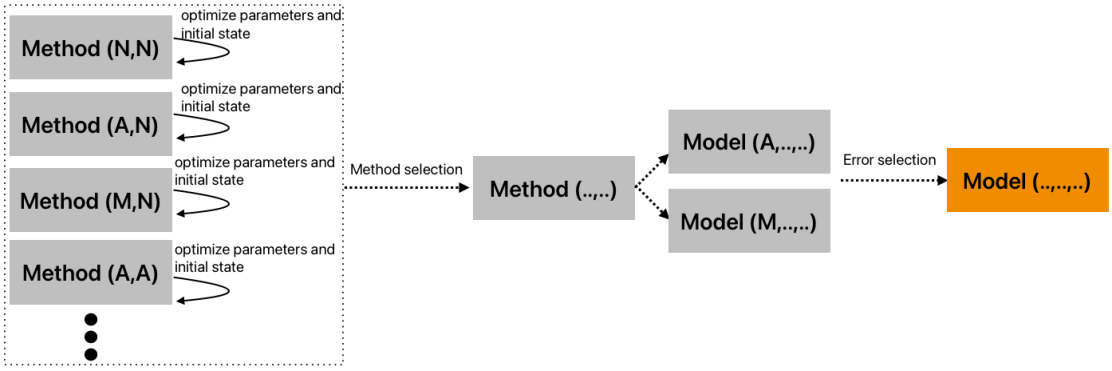


Figure 4.5: Model training procedure

4.6 Confidence intervals

Although we try to produce the best forecast points possible, there are several inevitably sources of uncertainty.

- The uncertainty in model choice
- The uncertainty in the estimates of the parameters (α, β, γ, $\boldsymbol{x_0}$)
- The uncertainty in future errors (ϵ_{t+1},..., ϵ_{t+h})

However, taking into account all these uncertainties is difficult. To simplify the task, we hypothesize a known model and known parameters. The confidence intervals will express the last

uncertainty. Historically, these intervals were computed with **statistical analysis** and each model had its own formulae. The other option possible is to use **simulation** to compute the intervals. We simulate many possible future paths with random error in each step. We generate observations $y_t^{(i)}$, for $t = n+1, ..., n+h$, starting with $\boldsymbol{x_n}$ from the fitted model. Thus, we continue the fitting step with the equation from the state space model, but since we do not have the correct y_t values, we randomly generate the error according to a chosen distribution. We repeat the procedure for i = 1,...,M, where M is typically a large integer (e.g., 5000). With all the simulated paths, we have collected a list of possible values for each forecast horizon: $\boldsymbol{y}_{n+h|n} = \{y_{n+h}^{(1)}, ..., y_{n+h}^{(M)}\}$. We can see in Figure 4.6 an example of a time series graph with some random future paths plotted. After these simulations, we can compute the desired quantiles at each forecast horizon step from all the simulated values. This technique has several advantages. It is very simple to implement, the results appear very close to the statistical ones, and the random error distribution can be chosen.

The error distribution is assumed to be Gaussian. The mean and variance of this Gaussian distribution are taken from the error recorded in the modeling part. During the computation of the chosen model, we simulate some forecast point at some point of the computation function. We compare the forecast obtained with the real points of the time series and save the difference. We regroup all these errors by their forecast horizon h. Therefore, we obtain h lists of all errors made for the forecast h in the future. Therewith, we computed the mean and variance of each list and used it to build the Gaussian distribution at the desired step. The schema of this process is drawn in Figure 4.7.

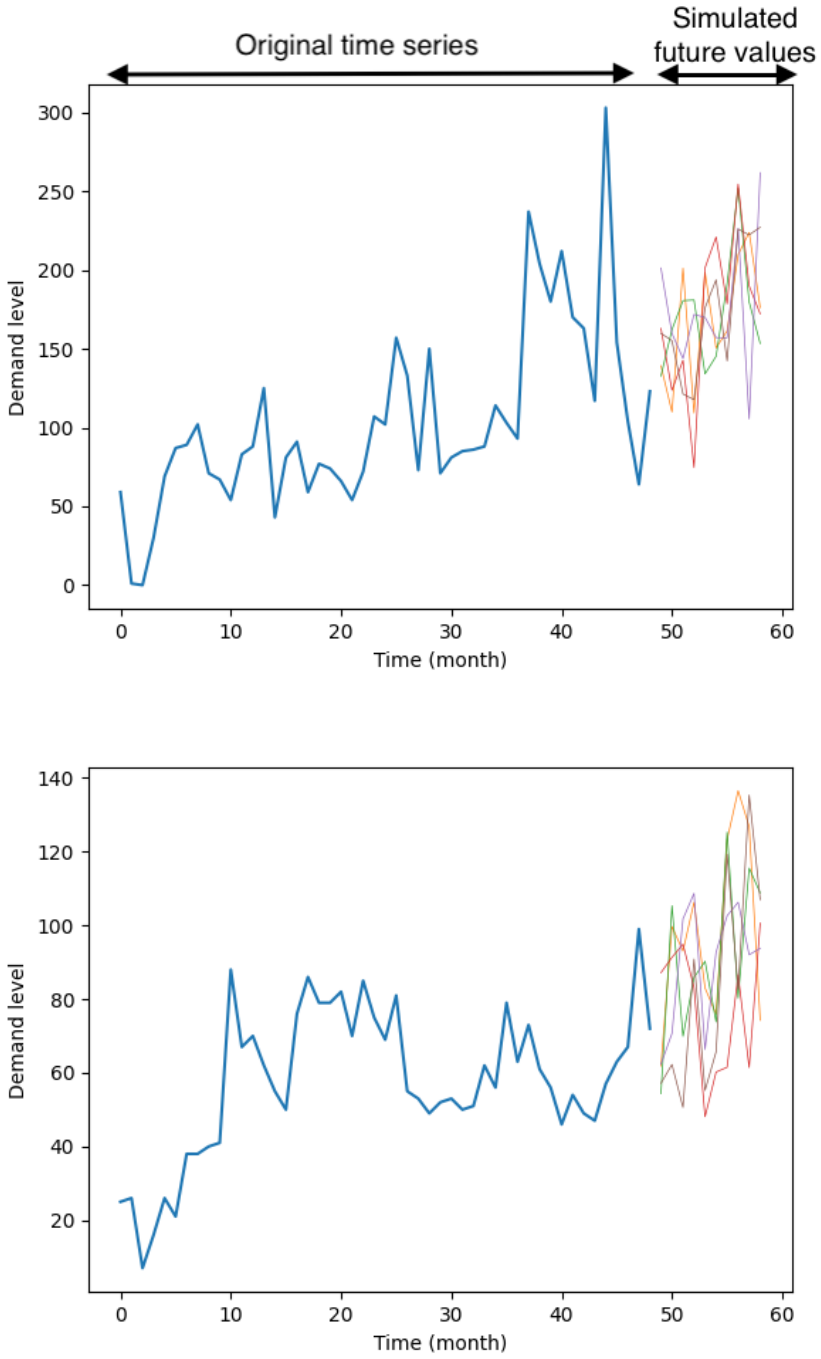


Figure 4.6: Time series with random future paths

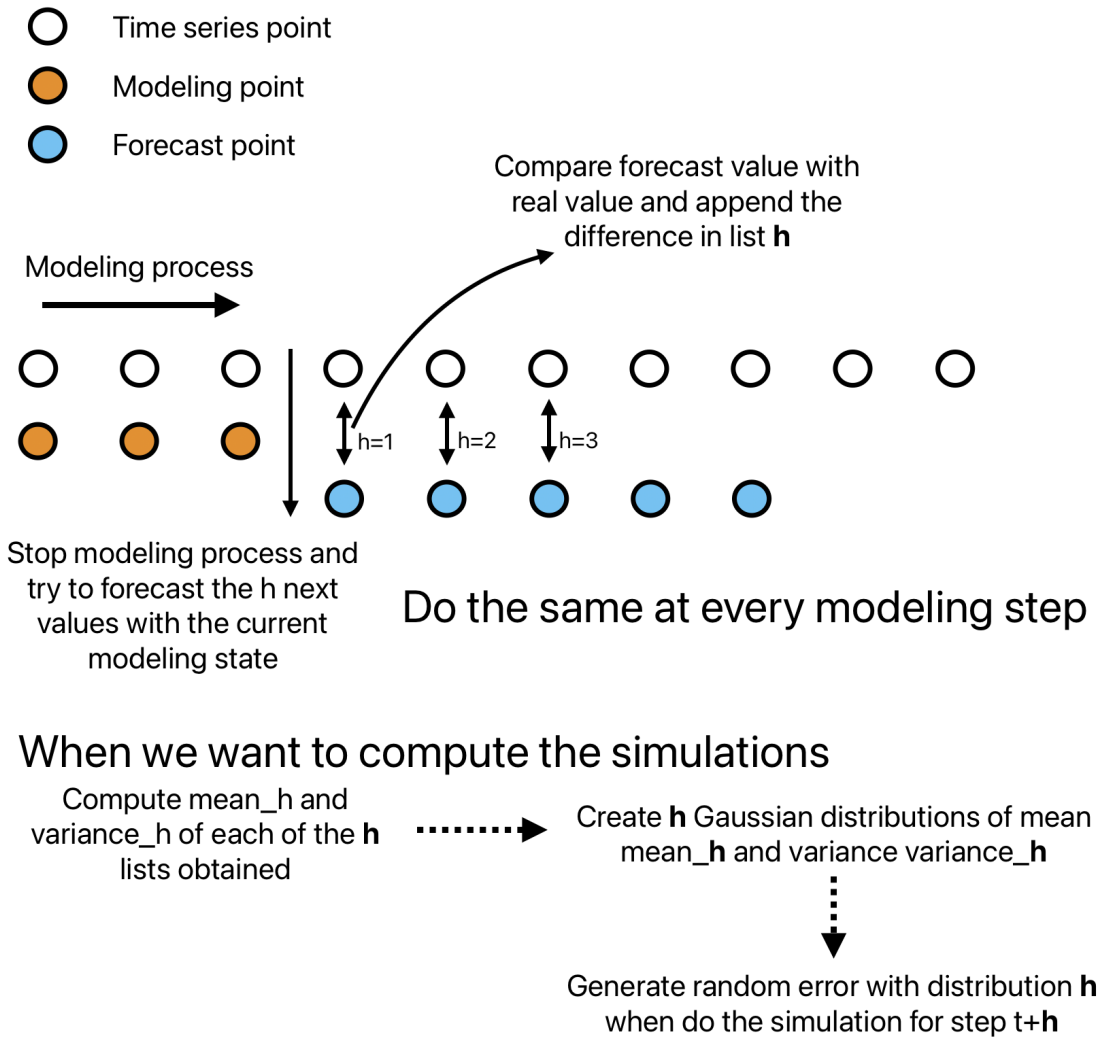


Figure 4.7: Schema for obtaining distribution to generate the errors during the simulation process

Chapter 5

Implementation in Odoo

After going through the theory, we will here explain how the implementation is made in Odoo software.

5.1 Odoo Architecture Overview

According to the Odoo documentation [18], the general architecture of Odoo software follows a multitier architecture. The presentation, the business logic, and the data storage are separated. As there are three levels, this is called a three-tier architecture. Each layer is written in a different programming language (Fig. 5.1).

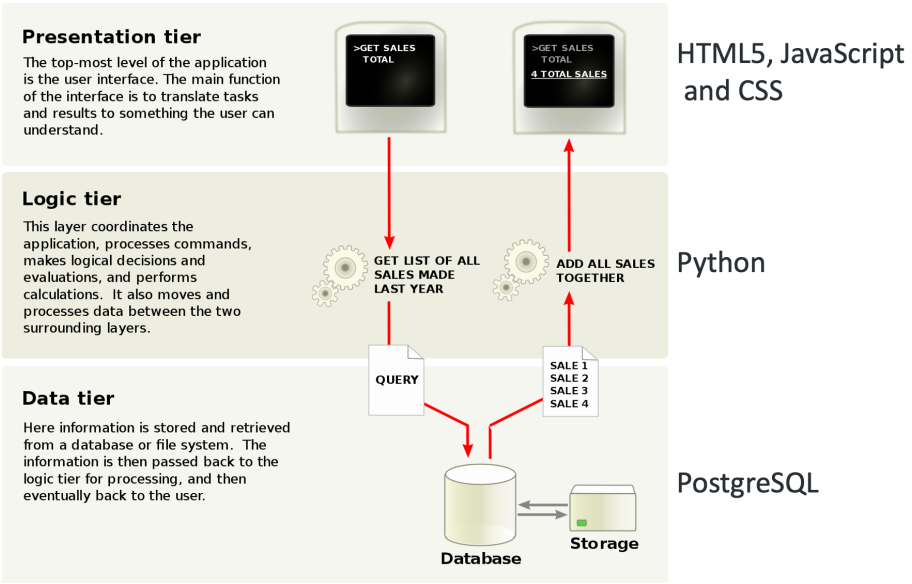


Figure 5.1: Three-tier architecture [19]

To complete the implementation step, the majority of the work will be realized in the logic tier, therefore in Python, while a small part will be accomplished in the presentation tier.
Actually, Odoo is a collection of modules which are optionally loaded with a database. There exists dependency between some modules. For example, a module can extend another module to add new

features to it. A module can contain logic level code as well as presentation level code. Another important point is that Odoo has two repositories. The main one is open source and regroups all the principal functions. The other one is private and contains additional features. This business model is called open-core, meaning that the core of an application is free for everyone, while Odoo keeps the possibility to sell additional features through a more complete enterprise version.

5.2 Implementation at Logic Level

5.2.1 Architecture overview

The modules that interest us are the MRP module (Manufacturing module) and an extension of it, the MPS module (Master Production Schedule module). Although it has limited impact, note that the MRP module is in the core repository, while the MPS is in the private enterprise repository. The relation is shown in figure 5.2.

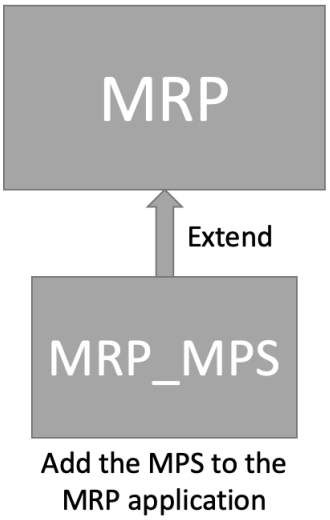


Figure 5.2: Architecture of MRP and MPS module

The target architecture after the implementation of the task is shown in Figure 5.3. A new module is created, named *ts_forecast* (time series forecast). To be able to use it, a module must add an adapter. This architecture allows reusing the forecast module in another task without duplicated code. As a result, the ts_forecast module is able to forecast any time series, not only a demand time series. This architecture can be reused if the *ts_forecast* module wants to be used in another module.

Let us then take a closer look at the modules. The more detailed architecture of the MPS and *ts_forecast* modules are shown in Figure 5.4.

We can see three basic Python classes in the *ts_forecast* module. These three models allow us to model and forecast a time series. Besides, there are three Odoo models. An Odoo model allows the storing of attribute class in the database automatically, and thus creates a link between the

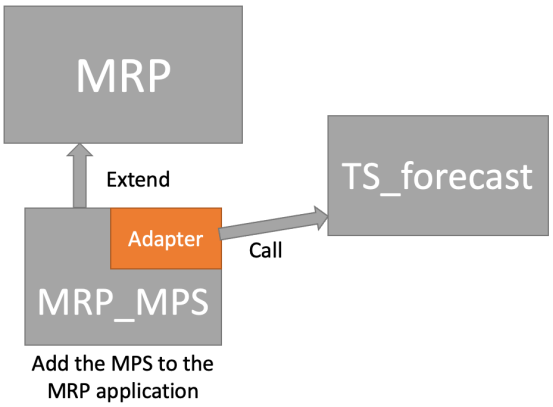


Figure 5.3: Architecture after added ts_forecast module

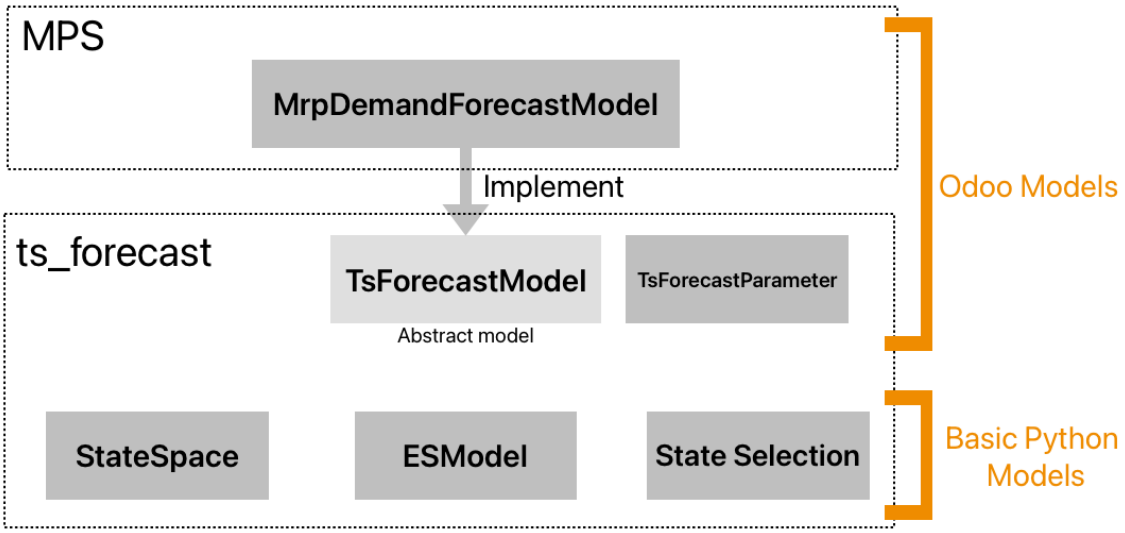


Figure 5.4: Architecture of the MPS and ts_forecast modules

logic level and the storage level. We will explain the function of each class below.

StateSpace is a basic Python class that represents a model from the state space model explained in Section 4.2. It takes as input the triplet that described the components of the time series (e.g. ('A', 'A', 'N') for the state error additive, trend additive and no season), the time series represented as a list of values, and the period of the time series (Figure 4.2). As we will try to obtain a regular time series, having as parameters the list of values ordered and the period is enough to represent the time series. The class will contain all the vectors and functions present in the state space equation 4.11. Therefore, this class describes entirely a model. The function *optimize_state_parameters* will try to estimate the parameters and initiate the vector $\boldsymbol{x_0}$ (4.3) for a specific time series.

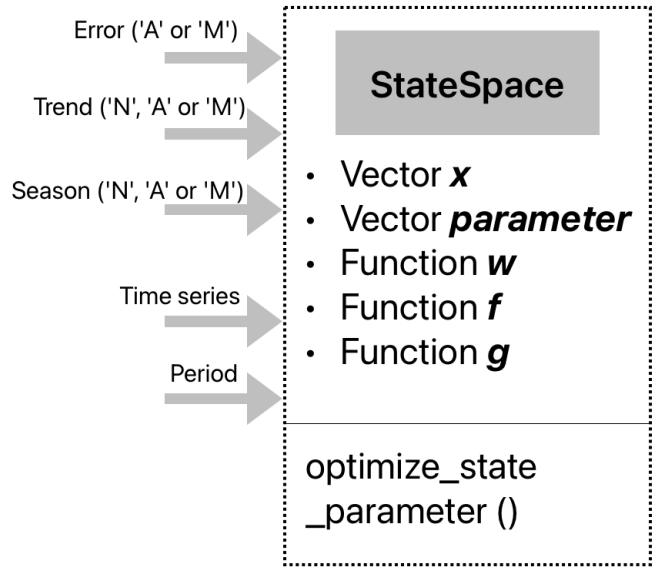


Figure 5.5: StateSpace class

ESModel (Exponential smoothing model) is also a basic Python class and is used to model and forecast time series with a provided state space model. It takes a stateSpace object as input and with that one can model the time series (*compute_model*()), compute the mean square error of a modeled time series (*compute_mse*()), forecast the modeled time series (*forecast*()) and compute the confidence intervals (*compute_simulation*()) (Figure 5.6). The mean square error is used to verify if a model fits the original time series correctly.

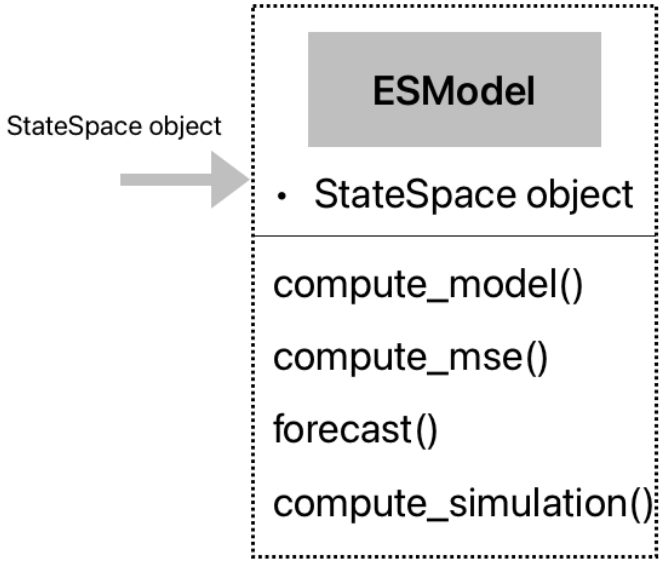


Figure 5.6: ESmodel class

StateSelection is the last basic Python class, and it is used to select the best state space model from a list of StateSpace object (*select_best_state*()) (Figure 5.7). The rules for selecting the best state are explained in Section 4.4.

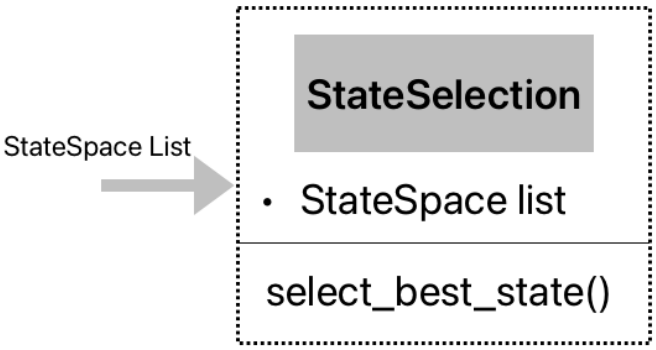


Figure 5.7: StateSelection class

TsForecastModel is an Abstract Odoo Model, which means that it must be extended by another Odoo model. This class will be responsible for doing the link between the basic Python models and the Odoo database. It saves all the data of the model in the Odoo database and uses the three previous classes to train the model. With the time series and the period, the function *train_model* is able to choose the best state with optimized parameters and save it in the database. The class **TsForecastParameter** represents a parameter that will be listed in the **TsForecastModel** model. The function *get_model()* simply returns an ESModel object built with the parameter saved in the database (Figure 5.8).

MrpDemandForecastModel is an extension of the **TsForecastModel**. This class is responsible for defining the meaning of the data. In this specific case, it will be the recorded past demand for a product. It also defines how the data are sent to the presentation level.

5.2.2 Execution Summary

A summary of a typical execution is shown in Figure 5.9. When the function *get_forecast_demands_data()* is called from the client side to obtain the forecast data, there is a first check to know if the model is trained or not. A model is considered trained if its stateSpace is defined, its parameters are optimized, and the date of the last training is the date of the last point in the time series. If this is the case, we directly get the model and, if not, the model is first trained. To summarize a training, we take all the applicable states from the stateSpace, and we optimize the parameters of each. Then, we select the best state and finally store it in the Odoo database. Getting the model is simpler; we get the previously saved stateSpace object from the Odoo database. We model the time series with this stateSpace object. Finally, we can get the forecast point and the confidence

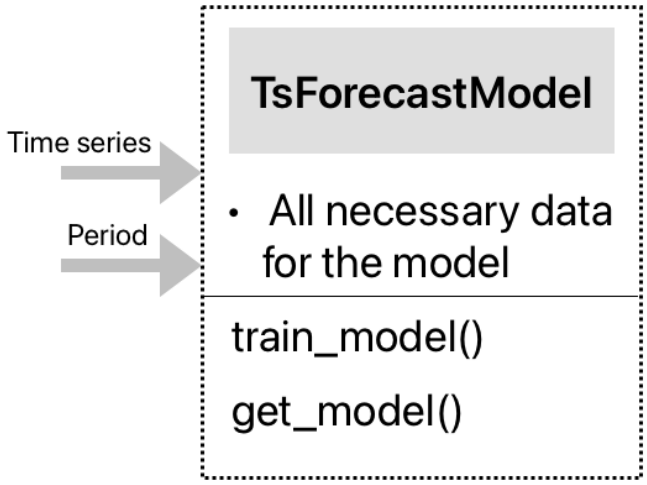


Figure 5.8: TsForecastModel abstract class

intervals and send them to the presentation level.

5.2.3 Time series Retrieval

One of the important steps in the process is the retrieval of time series data. As we have seen, this step takes place in the MrpDemandForecastModel. In this module, we define how to retrieve the time series and send it to the forecast module.

In our case, we want to recover from the database all sales history for a specific product. Fortunately, Odoo provides a powerful function to execute complex queries to the database without the need of writing the SQL query directly. It is the *read_group()* function [20]. We can ask to group, for a specific product, the sales quantity by a specific period with the sum aggregate to directly obtain the time series data. For the period without sales, we must add 0 values to the list in the good place.

To completely characterize the time series, we need the start date (the date of the first data), the period that space the data, and obviously the data list. We can see in Figure 5.10 an example of how a time series is recovered from a list of sales.

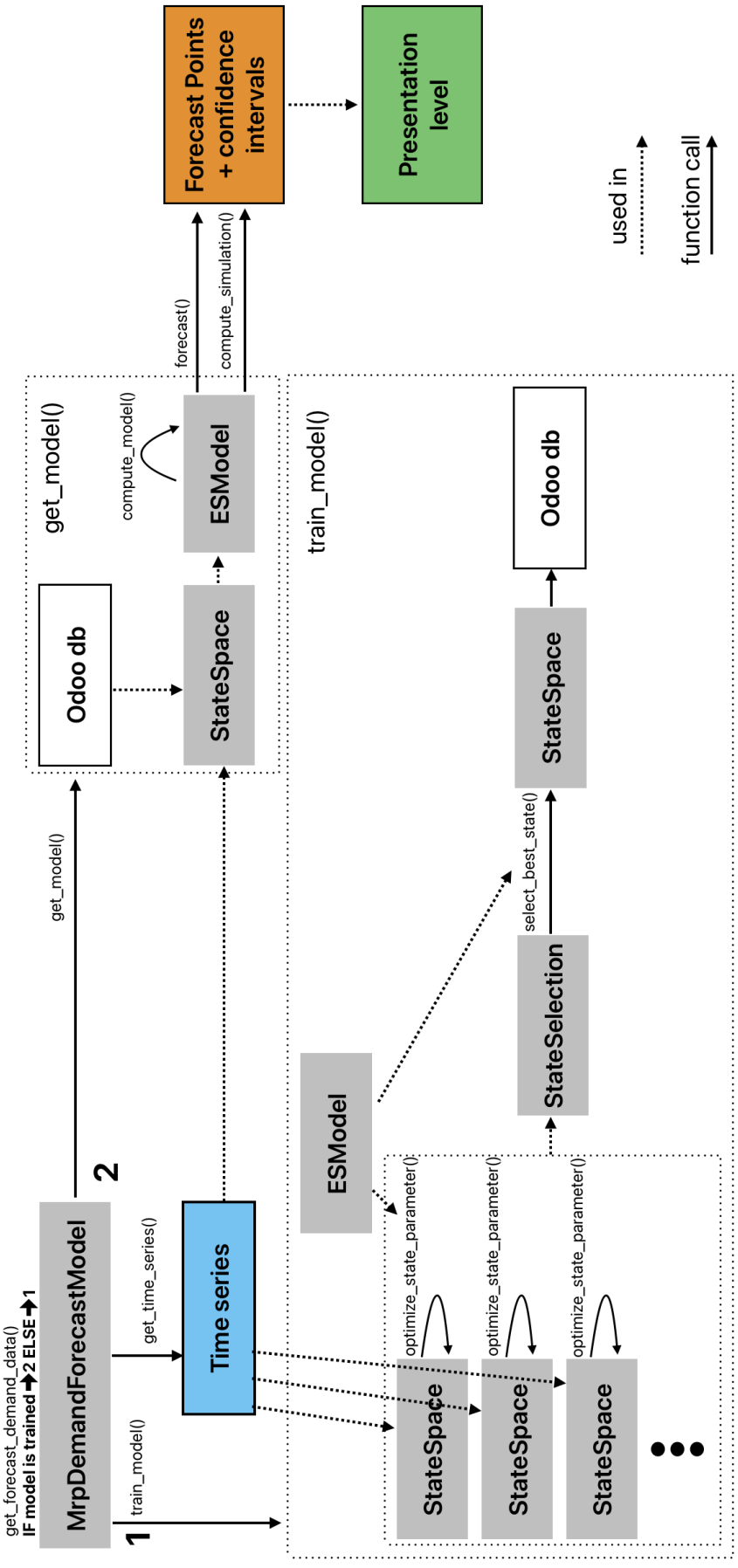


Figure 5.9: Execution summary

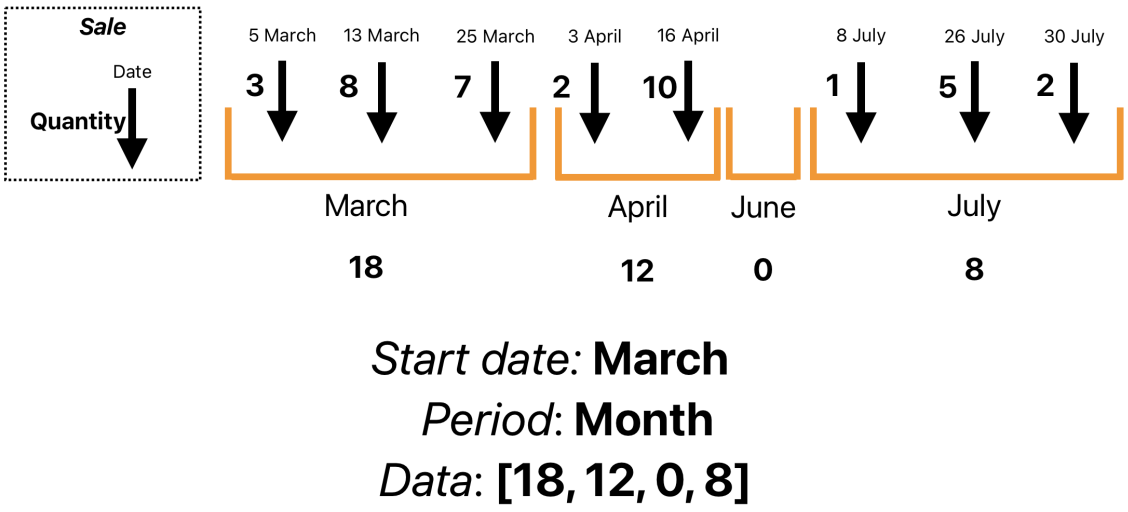


Figure 5.10: Example of time series retrieval from list of sales

5.2.4 Model Computation Algorithm

In this subsection we will explain in more detail the algorithm responsible for computing the model. This algorithm is based on the equations of the state space model explained in section 4.2.

$y \leftarrow$ time series

$\boldsymbol{x} \leftarrow \boldsymbol{x}_0$

$\boldsymbol{param} \leftarrow \alpha, \beta, \gamma$

while $i \leq len(y)$ **do**

$\hat{y} \leftarrow w(\boldsymbol{x})$

$error \leftarrow \frac{y_i - \hat{y}}{r(\boldsymbol{x})}$

$\boldsymbol{x} \leftarrow \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{param}) \times error$

$i \leftarrow i + 1$

end while

The vector $\boldsymbol{y}$ is a list that contains the time series data and $\boldsymbol{x}$ the vector state that is initialized by estimate values (subsection 4.3). For each point in the time series, we compute the estimation with the $w()$ function, then the error at step i is found and, finally, the vector $\boldsymbol{x}$ is updated. The functions w, $\boldsymbol{f}$ and $\boldsymbol{g}$ depend of the method, thus to switch between methods, only these functions change. They are defined by Python lambda functions and take the vector $\boldsymbol{x}$ (and the vector $\boldsymbol{param}$ that contains the parameters for the function $\boldsymbol{g}$) as input and can return simple numbers (w, r) or vector ($\boldsymbol{f}$ and $\boldsymbol{g}$). An example of these lambda functions is shown in Figure 5.11. They belong to the method with additive trend and no season. The r function is added afterward and depends on the error model. The parameters and initial vector $\boldsymbol{x}$ values are default values and are optimized later. The time complexity of this function is obviously $\mathcal{O}(n)$, where n is the length of the time series y.

```
self.state = {
    'x': [0, 0],
    'param': [0.1, 0.1],
    'w': lambda x: x[0] + x[1],
    'f': lambda x, t: [x[0] + x[1], x[1]],
    'g': lambda x, param: [param[0], param[1]],
    'forecast': lambda x, h: x[0] + h * x[1]
}
```

Figure 5.11: Example of stateSpace (A,A,N) defined by the Python lambda functions

5.2.5 Time Complexity Theoretical Analysis

One important specification is that the whole algorithm remains in $\mathcal{O}(n)$ where n is the number of points in the time series. Indeed, the length of the demand time series for a product can vary significantly. If a product is sold for five years, it will have five times more data than a product sold for one year. A time complexity greater than $\mathcal{O}(n)$ can quickly become problematic. We begin with a theoretical analysis and then attempt to confirm the results with some experiments in the next section.

The main function is the *compute_model*() function from the ESModel class, and as previously explained, it will model the time series point after point.Therefore, the time complexity of this function is $\mathcal{O}(n)$ where n is the number of points in the time series (number of periods recorded in this case). We can deduce the complexity of the *get_model*() function, as it just performs a *compute_model*() call.

For the training of a model we must also take into account the *optimize_state_parameter*() function. This function also uses the *compute_model*() function, but more than once. Indeed, each combination of parameters is tested, and each time the model is computed. Therefore, the time complexity is $\mathcal{O}(m.p.n)$ where m is the number of combinations of parameters to test and is a constant. p is the number of models that are optimized, it is a constant too. Then there is the state selection, which is approximately in $\mathcal{O}(k.n)$ where k is the number of different states in the state space. Thus, it is much smaller than m ($k << m$) and always smaller than 9. As m and k are constant, we can conclude that the time complexity remains linear with respect to the number of points in the time series.

5.3 Implementation at presentation level

After implementing the calculation at the logic level (in Python), we want to show the results to the user. Therefore, we must add code at the presentation level.

To see the time series graph and the forecast of the demand for a product, a button is added to the MPS. It is located in the subrow *Forecast demand* of every product. The button is shown in Figure 5.12.

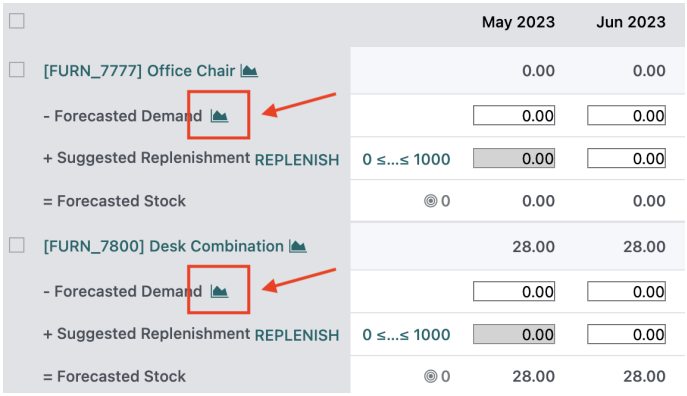


Figure 5.12: Location of the new button in the MPS

To draw the graph, there are three data sent to the user:

- The time series that contains the data and the start date (the period is known because it is defined by the user)
- The forecast points (a list of values)
- The confidence intervals (we choose to send only the bound point for the 50% and 80% confidence intervals) represented as one upper bound list and one lower bound list for each interval.

The x-axis is the time and the y-axis is the level of demand. We can see examples of results in Figures 5.13, 5.14, 5.15 and 5.16. The graph is done in JavaScript with the library chart.js already imported [21]. There is also a training button if the user wants to force the training of a product, and a date to show when the last training has been completed. Training is automatic when the user arrives on the graph if there is not yet a training or the last training was done without all data available now.

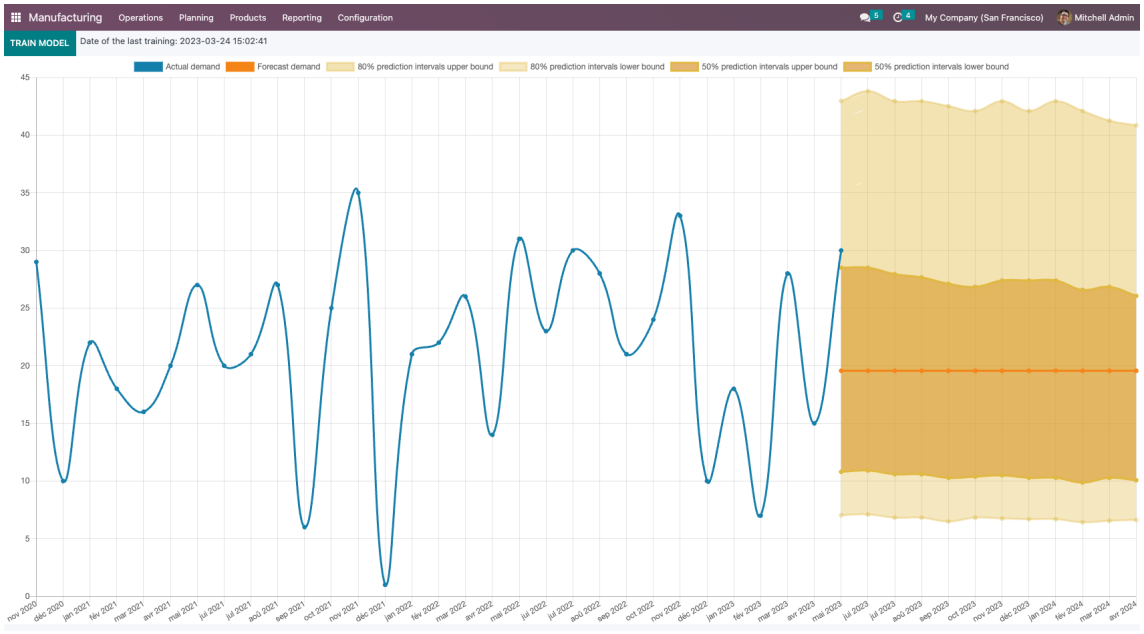


Figure 5.13: Example of result with no trend and no season detected

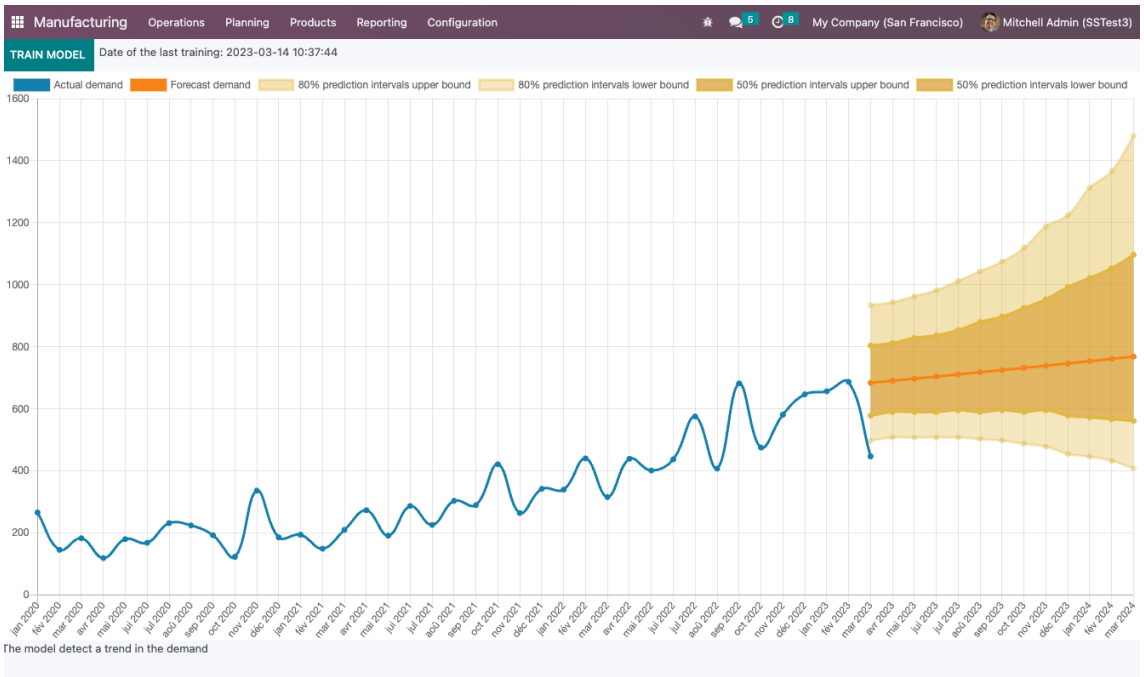


Figure 5.14: Example of result with a trend detected

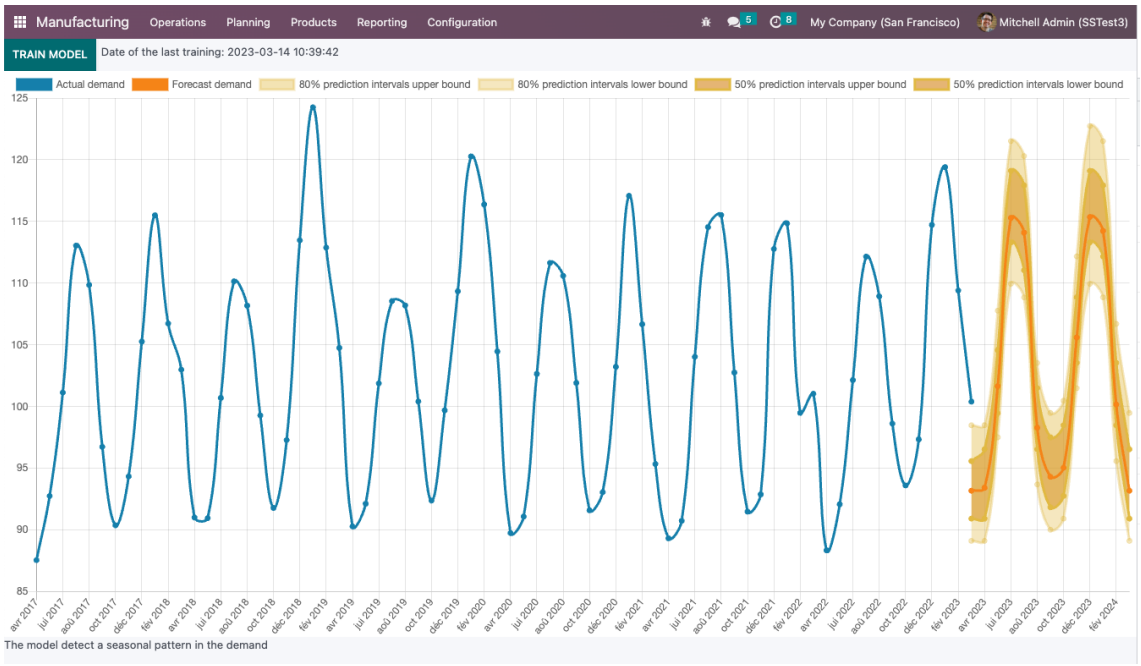


Figure 5.15: Example of result with a season detected

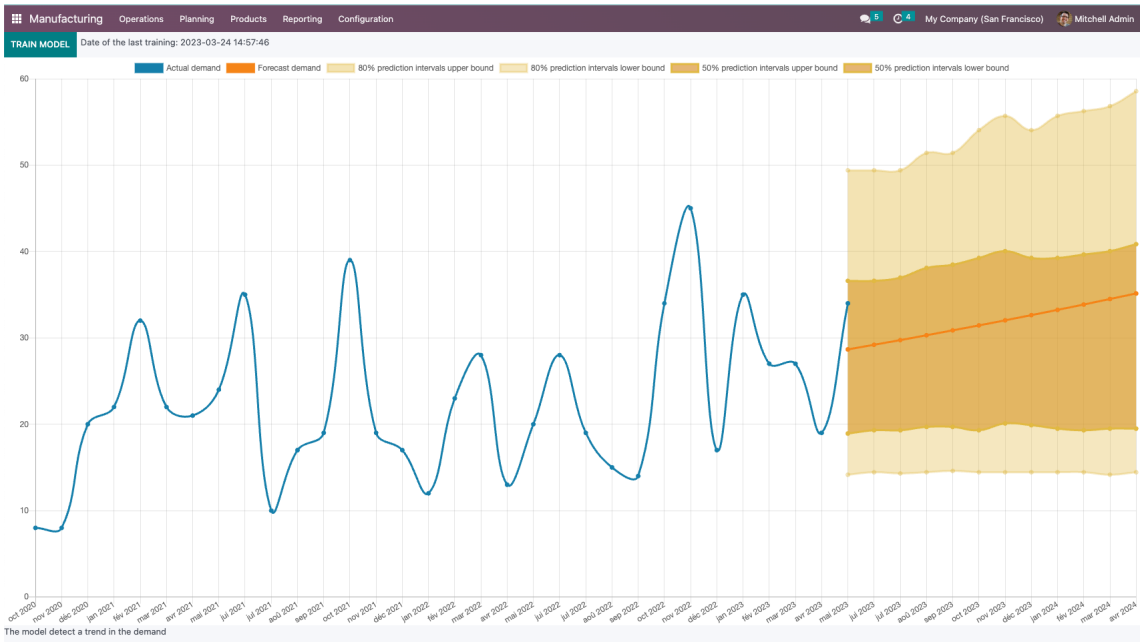


Figure 5.16: Example of result with a trend detected

Chapter 6

Validation Tests

In this section, the objective is to perform validation tests on our forecast methodology. The tests are divided in three parts:

- Test of the model selection procedure: verify that the model is well chosen
- Test of the accuracy of forecast points and confidence intervals
- Test of the time complexity of the training algorithm

6.1 Model Selection

With these tests, we want to verify that the algorithm chooses the right model. We generate a fake time series, in which we know the components, pass it to the model selection algorithm, and finally check if each component is well detected. For example, when we create a time series with an additive trend and no season, we expect that the model chosen is the model with an additive trend and no season.

To generate a time series, we first randomly select a model. Initial parameters and state vector are randomly estimated. Then we simulate the time series step by step with a random error in each step. This error follows a normal distribution, in which we will vary the standard deviation in the tests. This is exactly the same process as the simulation function to compute the confidence intervals.

For all tests, we use a prediction model error score defined in Figure 6.1. The green arrow corresponds to a correct component predicted, the orange arrow indicates when the component is not correctly predicted but the error is between additive and multiplicative, and the red arrow indicates when the model predicted None instead of additive or multiplicative, or the opposite. We obtain a score on seven that we can convert as a percentage. The higher this percentage, the better the model selection is. We made this choice because the most important goal is to

detect the presence or absence of a component, and having a wrong prediction between additive or multiplicative is relatively less important. Each set of tests is performed on 500 generated different time series.

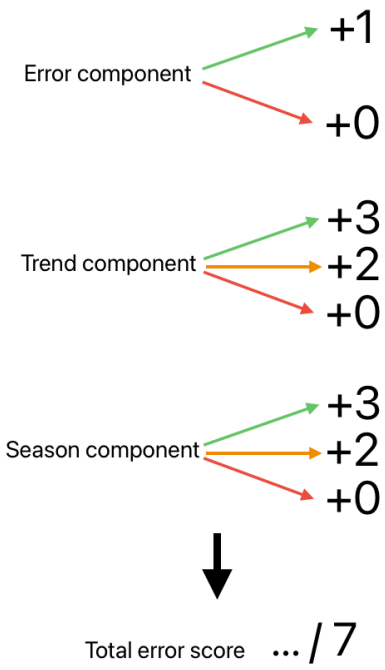
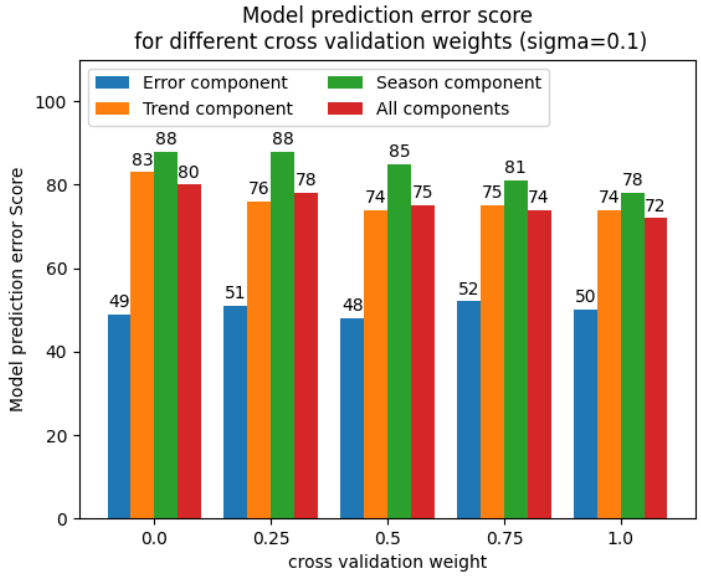
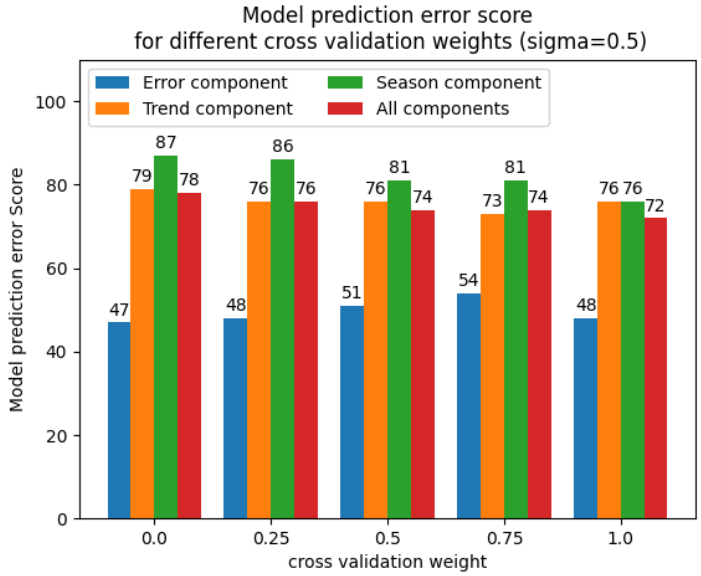


Figure 6.1: Model selection error score

For the first test, our aim was to modify the cross-validation weight to see the impact on the selection model error score. We also tried different standard deviations for the random error distribution. The results can be observed in the three graphs in Figure 6.2.

We can infer several comments from these graphs; firstly, the scores are good, especially the season component. Secondly, the error component is not well predicted. As expected if the standard deviation (sigma) is higher, the results are less effective but not drastically worse, and this is a good thing. Finally, when we look at the cross-validation weight, the error score decreases more as the weight increases. This is logical because the cross-validation aims at avoiding overfitting and deals with the prediction capabilities. Thus, the more we take into account the cross-validation score in the model selection process, the more the fitting model might walk away from the perfect fitting model. We still need to see how cross-validation performs on prediction accuracy.





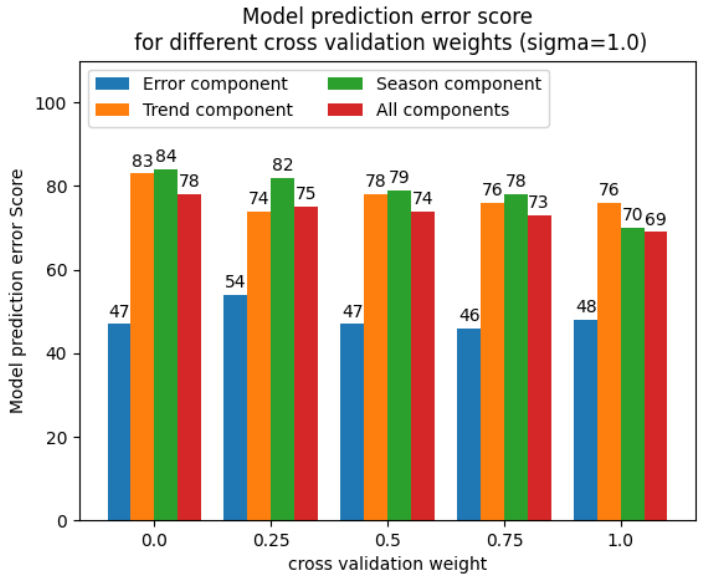


Figure 6.2: Comparison of the model selection error score for different cross-validation weight and standard deviation

The second thing we want to look at is what the precision of the parameters α, β, γ must be to obtain a good model selection result. Because, as we explained in section 4.3, there must be a trade-off between the precision of the parameters and the computation time.

Therefore, we tested several possible precisions for a parameter in phases one and two to see how the error score evolves. Results are shown in Figure 6.3. We can observe that the performance of the model selection procedure is not affected by the number of parameters that were tested in the optimize parameter function.

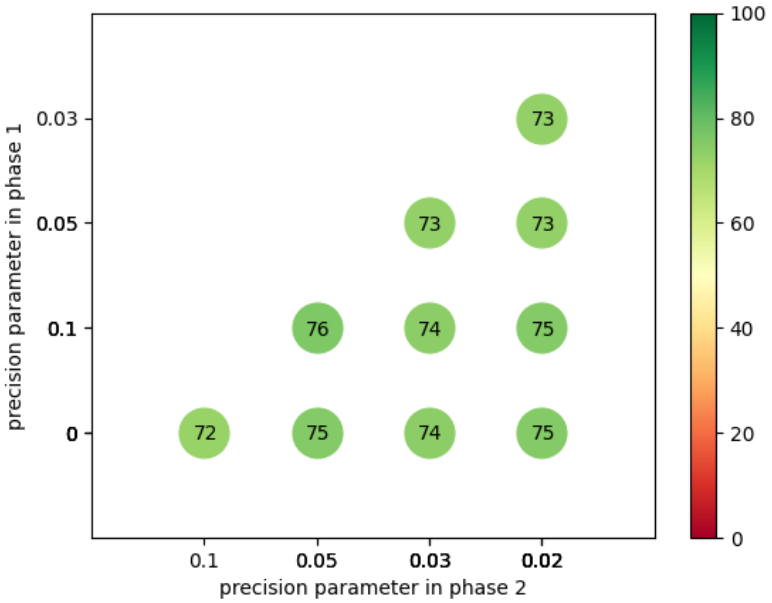


Figure 6.3: Model selection error score for different parameter precision in phase 1 and 2

Finally, for this section, we tried to generate time series of different lengths to see how the model selection error evolves. The graph of the results is shown in Figure 6.4.

Taking into account these results, three main conclusions can be drawn. One is that the global error score stays stable even when the time series lengths differ. Then, the trend component is easier to detect when the time series is longer, which is logical since we have a better global picture of the trend. And finally, the component season can be more difficult to detect when the size of the time series increases as there are more years to analyze.

6.2 Model Evaluation

In this part, the objective is to assert the accuracy of the forecast points and confidence intervals produced. Here, we need realistic time series taken from real life. We employ the large database used for an M5 competition [22]. The data comes from sales data from Walmart company. We hypothesize that sales are equal to demand. The data set contains more than 30 000 time series. These times series are the daily sales for a specific product. The duration of each time series is

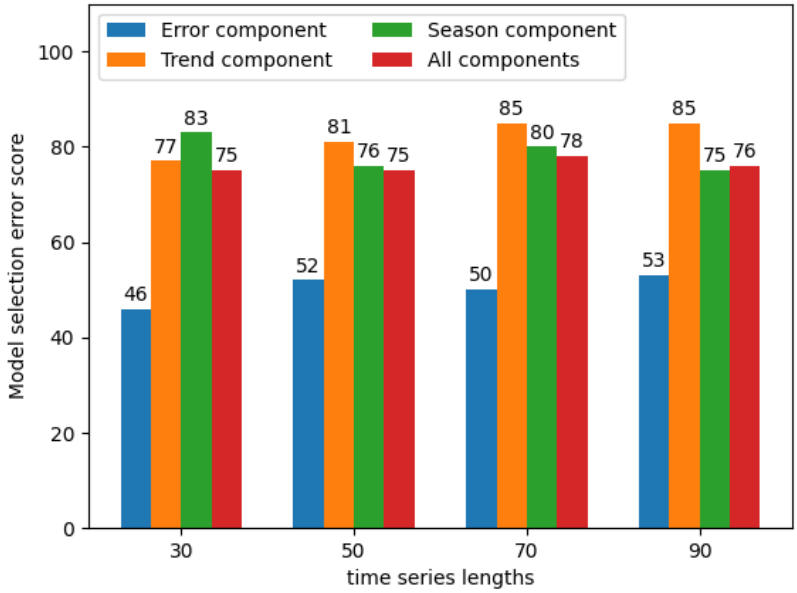


Figure 6.4: Model selection error score for different time series lengths

1940 days (54 months).

The error metric used to compare the accuracy of the forecast point is called *mean absolute scaled error* MASE [5]. With this metric we can compare the results obtained on two different time series, as it is scale independent, and handle non-strictly positive time series. The intuition behind this error metric is to scale the errors obtained at the forecast point based on the in-sample mean absolute error (MAE) with the naïve forecast method. Thus, we compute the $MAE = mean(|e_t|)$ where e_t is the error at time t between the original time series and the forecast point at time t obtained using the naïve forecast method. We use this error to scale the error obtained at each future step using our forecasting model.

$$q_t = \frac{e_t}{\frac{1}{n-1}\sum_{i=2}^{n}|y_i - y_{i-1}|}, \tag{6.1}$$

$$MASE = mean(|q_t|) \tag{6.2}$$

The metric used to evaluate the confidence intervals is simpler; we only record if the future point is in or out of the interval, and then we mean all the time series results. All tests are performed on a sample of 1000 different time series taken from the original data set.

In the first test, we try to compare the results using different cross-validation weights in the model selection test. We obtain Table 6.1 and the equivalent graph 6.5.

We can see that the best results are generally obtained with the 0.75 weight (except for the two first steps, where the 1.0 is better).

cross-validation weight	number of steps in the future									
	1	2	3	4	5	6	7	8	9	10
0.0	2.95	4.83	11.95	13.95	14.82	17.72	19.83	19.32	16.1	15.77
0.25	4.28	7.81	17.84	14.27	13.37	25.84	31.07	30.22	28.85	35.94
0.5	6.76	3.76	11.72	14.69	19.92	23.62	22.98	21.11	19.6	18.35
0.75	2.24	4.58	7.93	9.82	8.4	10.24	13.92	12.34	13.35	16.07
1.0	0.8	2.61	15.87	12.83	18.25	15.51	20.17	25.1	22.46	26.8

Table 6.1: Table of MASE for each future step and different cross-validation weights

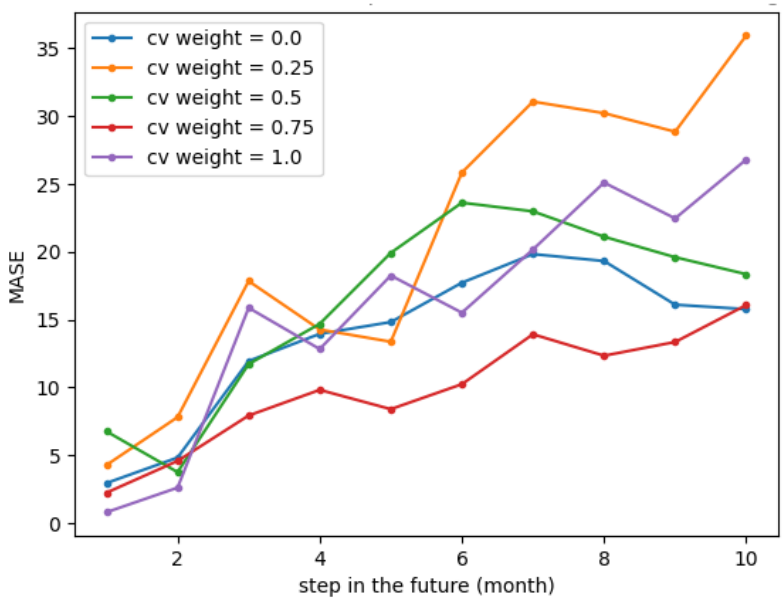


Figure 6.5: MASE for each step and different cross-validation weights

We also look at the confidence intervals and check the percentage of points that fall in the 50% and 80% intervals. The result should obviously be near 50% and 80% of IN. The results are shown in Tables 6.2 and 6.3

The results confirm the assumptions even if the percentage decreases slightly as we go far from the first step. We can observe once again that, the 0.75 weight shows the best results. Therefore, for the rest of the experiment, we will use the 0.75 cross-validation weight.

A second test aimed to look at the MASE for time series of different periods and lengths. Of course, the daily time series will be longer than the time series with a monthly period. The results are shown in Table 6.4

We can draw several conclusions from this experiment. Firstly, we can see that the scale of the

cross-validation weight	number of steps in the future									
	1	2	3	4	5	6	7	8	9	10
0.0	47.8	49.8	48.7	48.7	49.4	44.1	46.1	41.7	39.7	39.4
0.25	49.7	53.4	50.9	51.3	48.2	46.5	47.4	42.7	42.0	40.5
0.5	51.5	52.6	50.4	51.1	50.6	48.1	46.8	44.7	44.6	43.1
0.75	50.9	52.1	51.3	50.6	48.8	49.4	47.4	43.9	42.7	43.9
1.0	48.6	51.6	49.5	50.1	48.7	49.1	49.0	43.4	43.2	44.2

Table 6.2: Table of percentage of forecast point in the 50% confidence interval for different cross-validation weights

cross-validation weight	number of steps in the future									
	1	2	3	4	5	6	7	8	9	10
0.0	76.9	75.4	73.4	73.0	71.3	69.8	71.3	69.0	65.1	65.5
0.25	76.1	78.1	75.7	75.5	70.9	71.4	70.9	66.2	65.3	65.1
0.5	75.7	76.9	75.7	74.8	72.8	72.4	71.3	71.2	70.1	67.1
0.75	77.5	77.4	73.4	76.1	73.8	74.2	72.8	70.9	68.7	68.8
1.0	76.4	75.9	76.2	74.9	73.8	70.4	72.1	70.1	67.0	66.3

Table 6.3: Table of percentage of forecast point in the 80% confidence interval for different cross-validation weights

MASE is very different depending on the period of the time series. Despite the fact that the error metric is scale-free and allows us to compare forecast accuracy with different time series, in the demand context, it is difficult to compare different periods because the length of the time series and the level of demand differ a lot for two different periods. The second finding comes from the comparison of the MASE for different time series lengths. We clearly see that the longer the time series, the lower the MASE score is. This is quite logical because the modeling part can rely on more data and provide a better model.

The last feature to discuss is the optimization of the parameter, and in particular how many parameter combinations must be tested to obtain a good compromise between the quality of the results and the computational time. As explained in Section 4.3 the optimization of the parameters is done in two phases in order to avoid making the full optimization of the models that will not be chosen. The results of several possible combinations are shown in Table 6.5.

Obviously, the best results are obtained when the precision is the lowest (0.02). But with this precision, the computational time is high. Considering the objective of finding the best trade-off, we draw Table 6.6 to highlight the best choice.

To calculate the number of combinations tested, we use Table 6.5. We take the worst case,

length and period	number of steps in the future									
of the time series	1	2	3	4	5	6	7	8	9	10
1900 day	0.32	-0.09	-0.33	-0.19	-0.36	0.07	0.27	0.47	-0.07	0.07
1400 day	-0.46	-0.25	0.43	0.91	-0.22	0.37	0.15	-0.14	-0.26	0.2
720 day	-0.56	-0.65	-0.43	0.9	0.7	-0.69	0.11	0.03	-1.03	7.52
268 week	-0.12	0.11	0.4	0.57	0.07	0.19	0.53	0.59	0.62	-1.9
190 week	2.82	5.55	4.15	7.1	6.9	4.61	23.42	19.33	8.51	11.84
110 week	5.85	4.74	4.13	7.4	15.23	16.0	12.39	14.96	18.36	28.98
54 month	1.03	1.97	4.51	8.01	6.98	7.4	5.7	6.69	7.13	3.6
40 month	9.4	16.59	19.59	24.6	28.15	42.86	37.63	49.67	50.76	72.14
25 month	7.89	18.56	31.95	54.69	71.7	90.18	78.58	84.9	94.01	110.88

Table 6.4: MASE score for each step and different time series length and period

parameter's precision	number of steps in the future									
phase 1 and 2	1	2	3	4	5	6	7	8	9	10
0, 0.1	4.91	7.56	9.56	20.21	19.03	19.13	44.75	50.08	48.71	49.19
0, 0.05	3.45	5.08	15.11	15.62	18.87	20.23	21.54	19.71	17.6	19.68
0, 0.033	0.13	1.19	6.16	13.55	15.09	18.21	27.88	37.05	35.89	35.75
0, 0.02	5.44	0.69	7.59	8.95	12.49	9.38	12.18	13.83	12.27	17.6
0.1, 0.05	1.92	3.49	8.51	9.41	14.08	16.15	32.89	32.19	30.95	35.15
0.1, 0.033	3.14	7.31	23.67	23.45	25.43	30.46	37.07	36.15	36.25	37.8
0.1, 0.02	2.11	5.34	14.63	12.97	12.25	15.59	19.24	17.98	15.37	22.18
0.05, 0.033	3.94	11.13	27.06	32.73	31.07	31.61	50.99	51.9	48.04	47.73
0.05, 0.02	3.02	6.0	14.15	10.57	10.73	18.05	20.73	26.45	28.38	31.92
0.033, 0.02	1.22	4.78	10.97	12.91	14.94	14.04	13.69	12.92	12.13	16.16

Table 6.5: MASE score for different parameter precision in the optimization of the models

when the time series is strictly positive and all methods are applicable (9 methods with a different number of parameters for each). And when we are in phase 2, we choose to take 2 models with two parameters and two models with three parameters to average the best possible models selected after phase 1.

Though the final choice is not evident, taking the 0.02 precision for phase 2 seems mandatory to maintain a good MASE. For phase 1, we select the 0.1 precision to decrease the number of combinations, while maintaining a MASE close to the best one.

We can conclude the evaluation tests by performing a test with the final configuration chosen. It is recalled in Table 6.7. We tested this configuration on 10,000 different time series.

We show the results in Graph 6.6 and Table 6.8 for the forecast points and in Graph 6.7 and

parameter's precision			
phase 1	phase 2	average number of combinations to test	mean MASE over 10 first forecast
0	0.1	1210	27.31
0	0.05	8 820	15.68
0	0.033	28 830	19.10
0	0.02	375 000	10.04
0.1	0.05	5 610	18.47
0.1	0.033	15 610	26.07
0.1	0.02	66 210	13.76
0.05	0.033	23 220	33.62
0.05	0.02	73 820	17.00
0.033	0.02	93 830	11.37

Table 6.6: summary of the number of parameters combinations to test and MASE score

cross-validation weight	0.75
parameter precision phase 1	0.1
parameter precision phase 2	0.02

Table 6.7: Final configuration reminder

Table 6.9 for the confidence intervals.

6.3 Time Complexity Practical Analysis

After having analyzed the time complexity in a theoretical way in Section 5.2.5, we can confirm the results with practical tests. We measure the computational time of the training based on the time series length. All tests are performed on the same computer in the same configuration. The most important component is the relationship between measured time. That also allows us to get a first insight of how much time is approximately needed to train models. The measurement graph is shown in Figures 6.8 and 6.9.

We can observe that the theoretical results are confirmed, the evolution of the training computational time being well linear and function of the length of the time series.

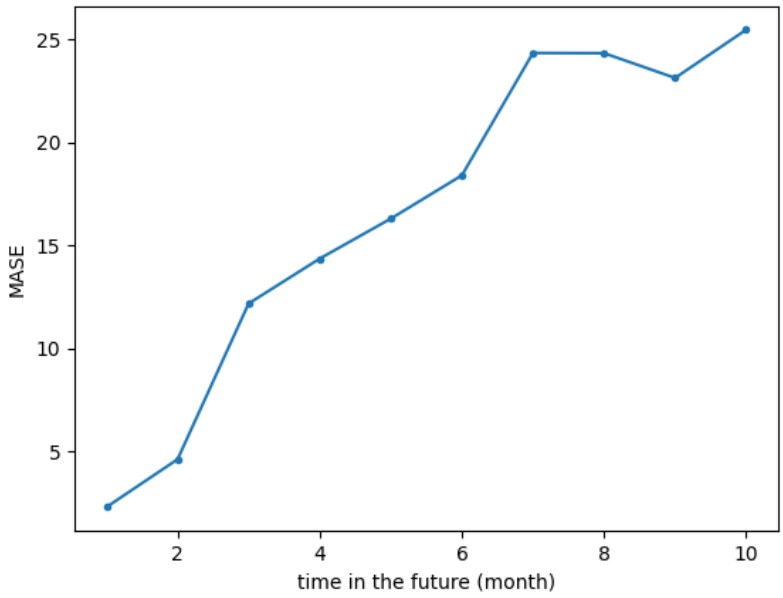


Figure 6.6: MASE for each future step with the final configuration

step in the future	MASE
1	2.31
2	4.63
3	12.19
4	14.37
5	16.31
6	18.41
7	24.35
8	24.34
9	23.14
10	25.48

Table 6.8: MASE for each future step with the final configuration

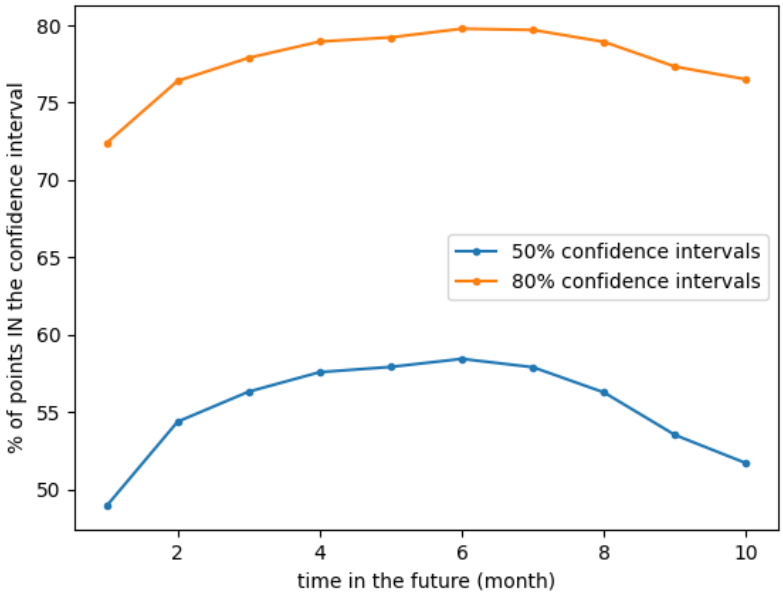


Figure 6.7: Percentage of points IN the 50% and 80% confidence intervals for each future step with the final configuration

step in the future	% of point IN the 50% confidence interval	% of point IN the 90% confidence interval
1	48.95	72.39
2	54.4	76.41
3	56.34	77.9
4	57.6	78.95
5	57.94	79.22
6	58.46	79.78
7	57.92	79.7
8	56.29	78.93
9	53.54	77.33
10	51.72	76.51

Table 6.9: Table of the percentage of points IN the 50% and 80% confidence intervals with the final configuration

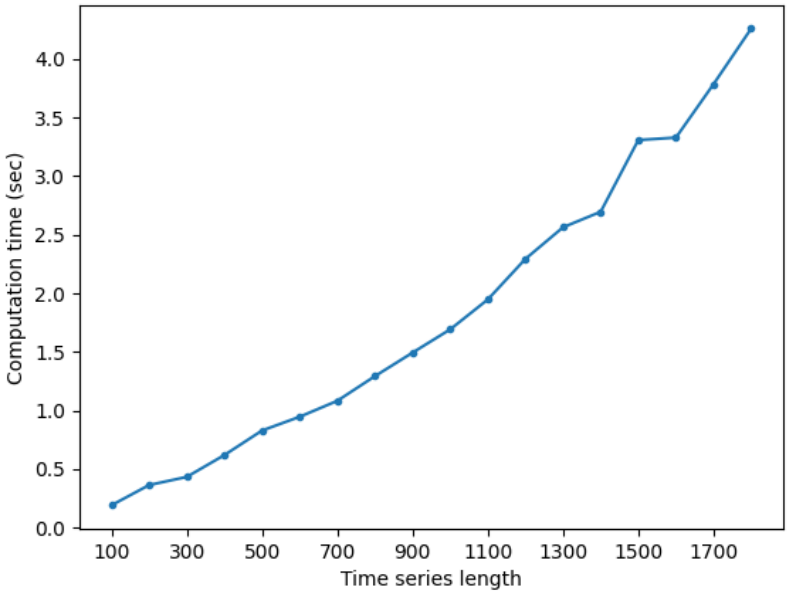


Figure 6.8: Computation time of model training for different time series lengths (with parameter precision 0.1 and 0.033 for phase 1 and 2)

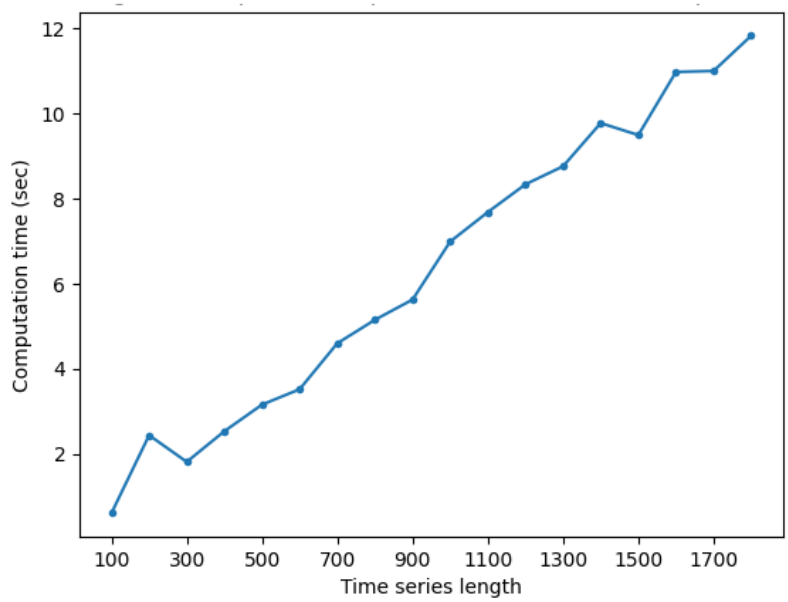


Figure 6.9: Computation time of model training for different time series lengths (with parameter precision 0.05 and 0.02 for phase 1 and 2)

Chapter 7

Conclusion and Future Work

7.1 Future Work

Before concluding this thesis, we can give some improvements for the future.

- The initial values for the term l_0, b_0 and all season term s_0,..., s_m are computed with heuristics, and due to the computational power available today, an optimization with the likelihood equation, as for the parameters, could be realistic.
- Season detection can be improved. Indeed, for the moment only the month season in a year is analyzed, but a season based on the day in a week or in a month could be possible.
- The multiplicative models are the most difficult to control; zero values can be problematic and cause possible instability in the forecast. We can cite, for example, the infinite variance problem or the convergence to zero problem. These problems and several methods to deal with them are covered in [5].

7.2 Conclusion

In conclusion, time series forecasting is a vital tool for making informed predictions about future events based on historical data. By leveraging the patterns and relationships within time series data, companies using Odoo software can gain valuable insights that enable better decision-making and planning for its production or replenishment.

We have implemented an algorithm capable of automatically building a model for a specific time series. Therefore, the user does not need to select or modify any parameters. All the tuning operations are done automatically. With this model, the forecast points and confidence intervals can be calculated and provided to the user.

By using the state space approach for implementing the exponential smoothing models, we have the possibility to easily extend the set of models with new exponential smoothing models,

and even improve the modeling of a time series.

Besides, the choice of architecture implementation allows the algorithm to be used in another place of the Odoo application. Indeed, as we have seen, time series forecasting can be used in many different subjects.

Finally, we have shown the good quality of the prediction results with the validation tests. This also allowed us to find the best configuration for the algorithm parameters.

The integration in Odoo is basic and visual, providing an initial glimpse into the potential of time series forecasting within the Odoo software.

Bibliography

[1] Odoo. *MPS documentation*. 2021. URL: https://www.odoo.com/documentation/16.0/applications/inventory_and_mrp/manufacturing/management/use_mps.html.

[2] Marco Peixeiro. *The Complete Guide to Time Series Analysis and Forecasting*. 2019. URL: https://towardsdatascience.com/the-complete-guide-to-time-series-analysis-and-forecasting-70d476bfe775 (visited on 02/14/2023).

[3] InfluxData. *What is time series data*. URL: https://www.influxdata.com/what-is-time-series-data/#:~:text=A%5C%20time%5C%20series%5C%20is%5C%20a,over%5C%20a%5C%20one-hour%5C%20procedure (visited on 02/14/2023).

[4] R.J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and practice (3rd edition)*. 2021. URL: https://otexts.com/fpp3/ (visited on 03/07/2023).

[5] David J. Hand. *Forecasting with Exponential Smoothing: The State Space Approach by Rob J. Hyndman, Anne B. Koehler, J. Keith Ord, Ralph D. Snyder*. 2009. DOI: https://doi.org/10.1111/j.1751-5823.2009.00085_17.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1751-5823.2009.00085_17.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1751-5823.2009.00085_17.x.

[6] Davide Burba. *An overview of time series forecasting models*. 2019. URL: https://towardsdatascience.com/an-overview-of-time-series-forecasting-models-a2fa7a358fcb (visited on 04/20/2023).

[7] E. A. Nadaraya. "On Estimating Regression". In: *Theory of Probability & Its Applications* 9.1 (1964), pp. 141–142. DOI: 10.1137/1109020. eprint: https://doi.org/10.1137/1109020. URL: https://doi.org/10.1137/1109020.

[8] A.J. Smola and B. Schölkopf. "A tutorial on support vector regression". In: *Statistics and Computing* 14 (2004), pp. 199–222. URL: https://doi.org/10.1023/B:STCO.0000035301.49549.88.

[9] Lim Bryan and Zohren Stefan. "Time-series forecasting with deep learning: a survey". In: *Phil. Trans. R. Soc. A.* 379 (2021). URL: https://doi.org/10.1098/rsta.2020.0209.

[10] David Salinas et al. "DeepAR: Probabilistic forecasting with autoregressive recurrent networks". In: *International Journal of Forecasting* 36.3 (2020), pp. 1181–1191. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2019.07.001. URL: https://www.sciencedirect.com/science/article/pii/S0169207019301888.

[11] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[12] Aaron van den Oord et al. *WaveNet: A Generative Model for Raw Audio*. 2016. arXiv: 1609.03499 [cs.SD].

[13] Spyros Makridakis et al. "Statistical, machine learning and deep learning forecasting methods: Comparisons and ways forward". In: *Journal of the Operational Research Society* 74.3 (2023), pp. 840–859. DOI: 10.1080/01605682.2022.2118629. eprint: https://doi.org/10.1080/01605682.2022.2118629. URL: https://doi.org/10.1080/01605682.2022.2118629.

[14] Rob J. Hyndman and Yeasmin Khandakar. "Automatic Time Series Forecasting: The forecast Package for R". In: *Journal of Statistical Software* 27.3 (2008), pp. 1–22. DOI: 10.18637/jss.v027.i03. URL: https://www.jstatsoft.org/index.php/jss/article/view/v027i03.

[15] Rob J Hyndman et al. "A state space framework for automatic forecasting using exponential smoothing methods". In: *International Journal of Forecasting* 18.3 (2002), pp. 439–454. ISSN: 0169-2070. DOI: https://doi.org/10.1016/S0169-2070(01)00110-8. URL: https://www.sciencedirect.com/science/article/pii/S0169207001001108.

[16] R.J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and practice (3rd edition)*. 2021. URL: https://otexts.com/fpp3/classical-decomposition.html (visited on 03/07/2023).

[17] Egor Howell. *How To Correctly Perform Cross-Validation For Time Series*. 2023. URL: https://towardsdatascience.com/how-to-correctly-perform-cross-validation-for-time-series-b083b869e42c (visited on 03/10/2023).

[18] Odoo. *Odoo tutorial 1 (Architecture Overview)*. 2022. URL: https://www.odoo.com/documentation/16.0/developer/tutorials/getting_started/01_architecture.html.

[19] Wikipedia. *Multitier architecture*. 2023. URL: https://en.wikipedia.org/wiki/Multitier_architecture (visited on 05/03/2023).

[20] Odoo. *Odoo $read_groupfunction$*. URL: https://www.odoo.com/documentation/16.0/developer/reference/backend/orm.html#odoo.models.Model.read_group.

[21] *Chart.js*. URL: https://www.chartjs.org/.

[22] University of Nicosia. *M5 Forecasting - Accuracy*. 2020. URL: https://www.kaggle.com/competitions/m5-forecasting-accuracy/overview.

[23] Edouard Thieuleux. *Safety Stock Formula Calculation: 6 best methods*. URL: https://abcsupplychain.com/safety-stock-formula-calculation/ (visited on 05/11/2022).

Appendix

Safety Stock

In addition to implementing a forecasting algorithm in Odoo software, another task about safety stock was assigned. We will present it briefly in this Appendix because this work has mostly enabled the discovery of Odoo software and has been a good first step into the code.

Firstly, the concept of safety stock is explained. Then, different methods to calculate it are presented. And finally, the result is shown.

What is the safety stock

The safety stock is a level of additional stock that the vendor maintains to minimize the risk of possible stock out.
This is useful for the vendor for many reasons:

- The stock is ready to face possible demand spikes
- If the lead time is huge, it is important for the vendor to not be out of stock otherwise the client could not purchase
- It can be used to prevent price fluctuation

How calculate the safety stock

Firstly, it is important to well estimate the safety stock. Indeed, a too high safety stock could lead to a too high increase of the holding costs, and if the safety stock is too small, the vendor could have stock out.

The formulas used are taken from [23]. Before listing them, the different quantities used are explained:

- SS = Safety Stock
- sales = demand per time's unit
- LT = Lead time (time to replenish a product in the stock)

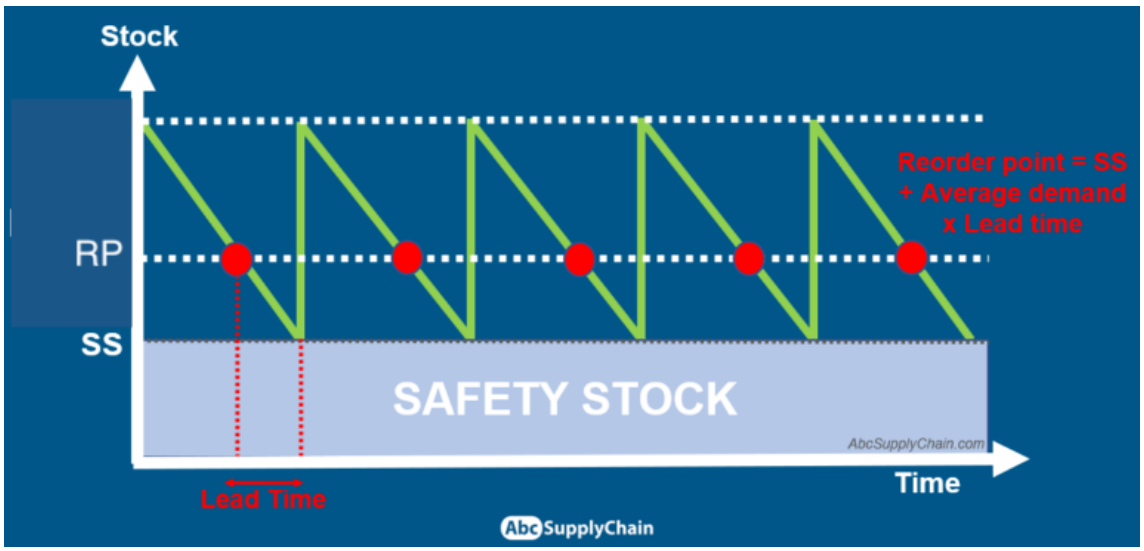


Figure 7.1: Safety Stock and Reorder point [23]

- μ = mean
- σ = standard deviation
- Z = service factor (link to the normal distribution)

For all formulas, all the quantities use the same unit of time.

Name	Equation
Average-Max	$SS1 = (max(LT) \times max(sales)) - (\mu_{LT} \times \mu_{sales})$
Uncertainty about demand	$SS2 = Z \times \sigma_{sales} \times \sqrt{LT}$
Uncertainty about lead time	$SS3 = Z \times \mu_{sales} \times \sigma_{LT}$
Uncertainty about demand and lead time (independent)	$SS4 = Z \times \sqrt{LT \times {\sigma_{sales}}^2 + {\mu_{sales}}^2 \times {\sigma_{LT}}^2}$
Uncertainty about demand and lead time (dependent)	$SS5 = Z \times \sigma_{sales} \times \sqrt{LT} + Z \times \mu_{sales} \times \sigma_{LT}$

Table 7.1: Safety stock formula

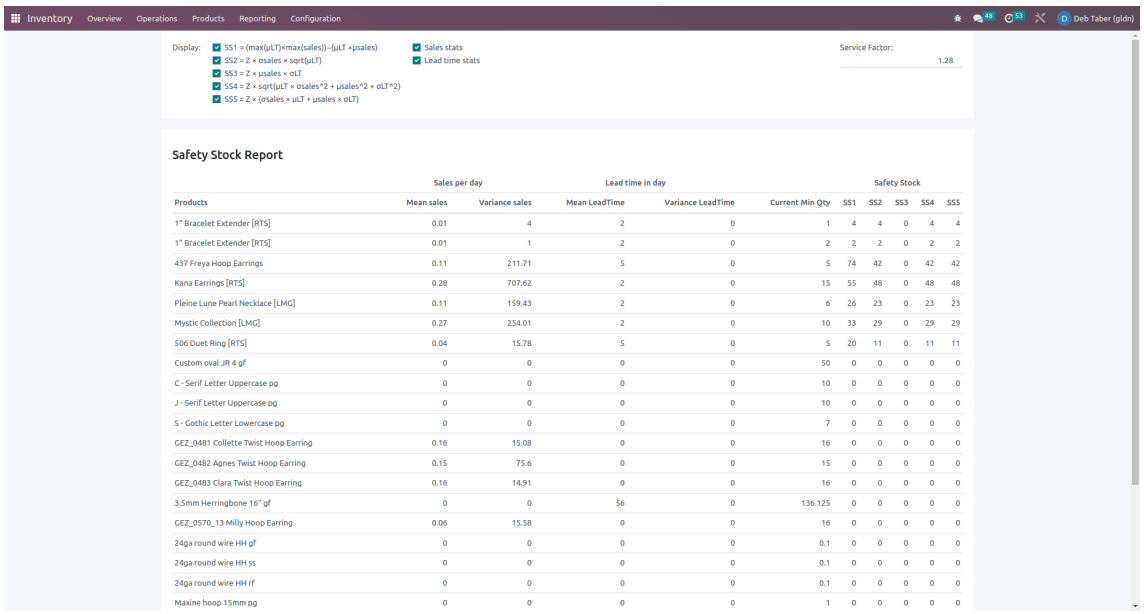
Figure 7.1 illustrates the principle of safety stock and the reorder point (when the stock level reaches the reorder point, we must replenish the product).

Implementation in Odoo

The implementation contains three part:

- To retrieve the quantities present in the formulas from the database
- To calculate the safety stock with the formulas
- To present the report for the user

For the first part, we have to compute the means and standard deviations for the sales history recorded in the database.



	Sales per day		Lead time in day			Safety Stock				
Products	Mean sales	Variance sales	Mean LeadTime	Variance LeadTime	Current Min Qty	SS1	SS2	SS3	SS4	SS5
1" Bracelet Extender [RTS]	0.01	4	2	0	1	4	4	0	4	4
1" Bracelet Extender [RTS]	0.01	1	2	0	2	2	2	0	2	2
437 Freya Hoop Earrings	0.11	211.71	5	0	5	74	42	0	42	42
Kana Earrings [RTS]	0.28	707.62	2	0	15	55	48	0	48	48
Pleine Lune Pearl Necklace [LMG]	0.11	159.43	2	0	6	26	23	0	23	23
Mystic Collection [LMG]	0.27	254.01	2	0	10	33	29	0	29	29
506 Duet Ring [RTS]	0.04	15.78	5	0	5	20	11	0	11	11
Custom oval JR 4 gf	0	0	0	0	50	0	0	0	0	0
C - Serif Letter Uppercase pg	0	0	0	0	10	0	0	0	0	0
J - Serif Letter Uppercase pg	0	0	0	0	10	0	0	0	0	0
S - Gothic Letter Lowercase pg	0	0	0	0	7	0	0	0	0	0
GEZ_0481 Collette Twist Hoop Earring	0.16	15.08	0	0	16	0	0	0	0	0
GEZ_0482 Agnes Twist Hoop Earring	0.15	75.6	0	0	15	0	0	0	0	0
GEZ_0483 Clara Twist Hoop Earring	0.16	14.91	0	0	16	0	0	0	0	0
3.5mm Herringbone 16" gf	0	0	56	0	136.125	0	0	0	0	0
GEZ_0570_13 Milly Hoop Earring	0.06	15.58	0	0	16	0	0	0	0	0
24ga round wire HH gf	0	0	0	0	0.1	0	0	0	0	0
24ga round wire HH ss	0	0	0	0	0.1	0	0	0	0	0
24ga round wire HH rf	0	0	0	0	0.1	0	0	0	0	0
Maxine hoop 15mm pg	0	0	0	0	1	0	0	0	0	0

Figure 7.2: Safety Stock Report in Odoo

The two last parts are done on the client side. The final result shows a report of the different safety stock for a list of selected products. It is shown in Figure 7.2. The user has the possibility to modify the service factor Z to increase the probability of facing the demand and to select what to display in the report.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**