# University of Padua

Department of Mathematics

Master Thesis in Data Science

# Exponential smoothing with neural networks: a TBATS-NN hybrid model for hierarchical time series forecasting


*Supervisor*

Prof.ssa Mariangela Guidolin

Statistical Science Department

*Master candidate*

Francesco Biancucci

*Student ID*

2087620


*Academic year*

2023/2024

# ABSTRACT

Time series analysis is a key aspect in the manufacturing company field, as it is used to extract information from the previous sales of various products in order to forecast plausible future behaviours of the market. This allows the managers to elaborate proper marketing campaigns and organize the supply chain according to the predicted quantities they will sell in order to maximize profit.

In particular, hierarchical and grouped time series are a huge part of the sales-related data, as the data can be aggregated in various ways based on the type of product and where they were sold. Analysing these kinds of time series has the added difficulty given by the linear constraints the different levels of the hierarchy have with each other: the sum of the forecasted lower levels of the hierarchy needs to be equal to the predicted upper level they stem from.

In this thesis, a practical case of a manufacturing company sales data is used to outline how to handle hierarchical or grouped time series analysis using Python code, starting from the exploratory analysis and data cleaning, and then moving on to how to implement the most used forecasting models for time series. In particular, a new approach to the exponential smoothing and neural network hybrid, not documented in the current literature, is proposed: the TBATS-ANN hybrid. It is then shown how to reconcile the predictions in order to have results that satisfy the already mentioned linear constraint to which all the series are subject. In the end, different kind of performances are confronted in order to get the best model for the problem at hand.

# INDEX

# 1 INTRODUCTION

## 1.1 HIERARCHICAL AND GROUPED TIME SERIES

A time series is a list of numbers, the measurements, paired with some information about what times those numbers were recorded, the index (Hyndman & Athanasopoulos, 2021).

If a time series can be disaggregated in various attributes along a certain dimension, then it is called a hierarchical time series (HTS). These series types have been thoroughly studied given their importance in fields like business and economics.

The term hierarchical is used when the various categories in which the series can be split are nested within a larger group category, for example with a company's national sales being divided into the sales for each region. However, sometimes there is no natural way to disaggregate in a unique hierarchical fashion the various subclasses, as one can incur into such instances when the disaggregation dimensions are not nested but crossed: these are referred to as grouped time series (Hyndman & Athanasopoulos, 2021). The variables by which the data is split are called disaggregating factors.

There might be some cases where the disaggregation dimensions are both nested and crossed, resulting in a more complex scenario. Just to make a simple example, let us consider the sales of a company: they can be disaggregated into the geographical regions where its stores are located, each region's sales can then be divided into the single stores where its products are being sold, which can be further split into the various categories of product which are sold. In this scenario the highest hierarchy level would be the total profit of the company, which could then be followed in order by the region sales, the single store sales and the single product sales. But while the region and its stores are nested, the product split could be made earlier, so we could have the total sales split in the different products' sales first, and only then be further divided into the regions and their stores. This structure could be described as a "nested" geography hierarchy "crossed" with the product type.

Let us add some numbers to introduce the notation that will be then used in this work: let us now assume that the company sales its 2 products (X, Y) into 3 different regions (A, B, C), all of which have 2 stores (number from 1 to 6).

The first scenario mentioned had the geographical split before the product split, so a diagram of it at time $T$ would be the one shown in Figure 1.1.

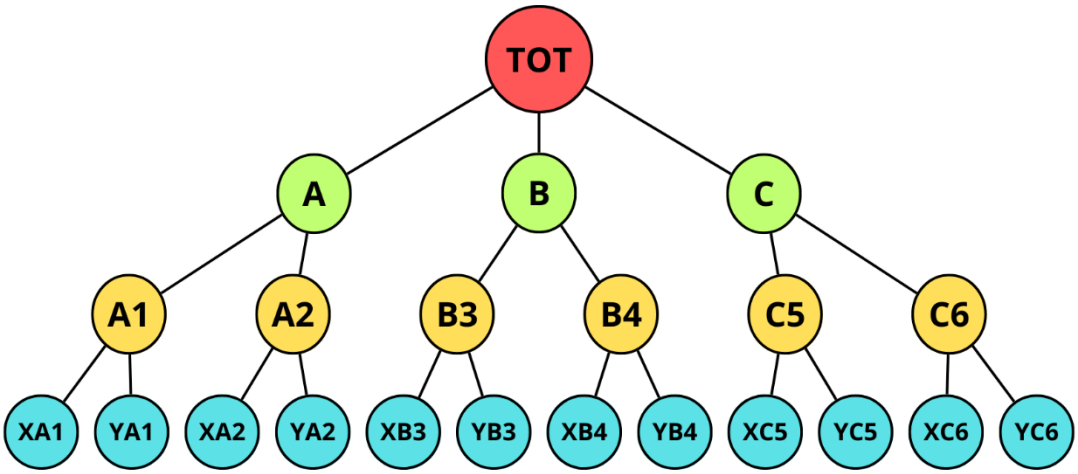


*Figure 1.1*

There are $n = 1 + 3 + 6 + 12 = 22$ series at all times, with $m = 12$ bottom level series. For any time $t$, the observation at any bottom level will sum up to the level above. To show some examples:

$$y_{t,tot} = y_{t,A} + y_{t,B} + y_{t,C}$$

$$y_{t,A} = y_{t,A1} + y_{t,A2}$$

$$y_{t,A1} = y_{t,XA1} + y_{t,YA1}$$

*Equation 1.1*

If the cross variable of the product is considered first, the diagram at time $T$ would be like the one in Figure 1.2.

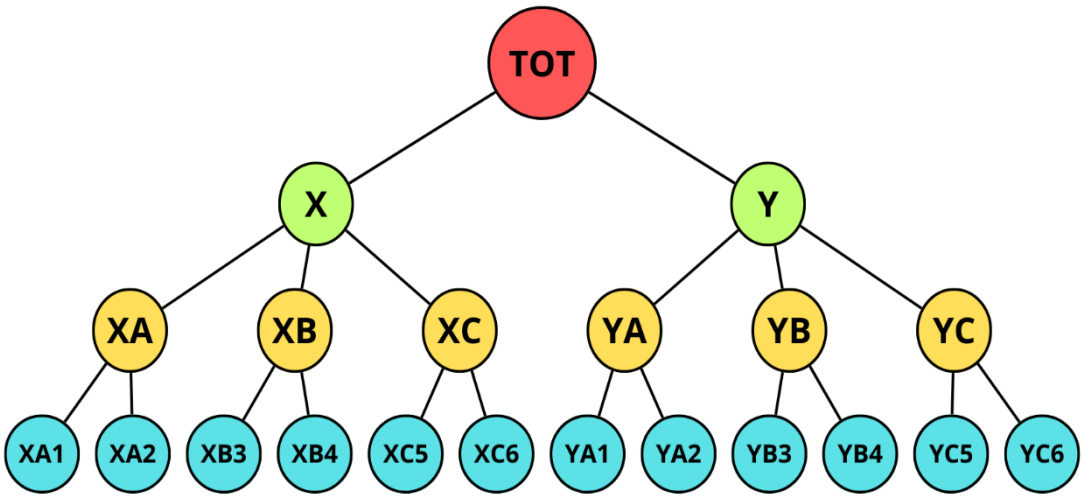


*Figure 1.2*

As one can see, the disaggregation is different and there are $n = 1 + 2 + 6 + 12 = 21$ series at each time, while keeping the $m = 12$ bottom level series. One can notice that, even though they are

listed in a different way, the bottom-level series are always the same, no matter when the split using the cross variable is introduced. Here an example of aggregation would be shown in Equation 1.2.

$$y_{t,tot} = y_{t,X} + y_{t,Y}$$

$$y_{t,X} = y_{t,XA} + y_{t,XB} + y_{t,XC}$$

$$y_{t,XA} = y_{t,XA1} + y_{t,XA2}$$

*Equation 1.2*

In situations like this, which disaggregation factors and when to use them while dividing the data is problem-dependent and should be thought through before starting any kind of analysis.

From now on the term HTS will be used for the sake of simplicity, even if the topic is grouped time series. After all, once a way to re-aggregate the data is chosen, one can identify a hierarchy even if some variables do not have a proper hierarchy themselves.

## 1.2 FORMAL DEFINITION

As previously discussed, the main characteristic an HTS has is the linear constraints it is subject to. For the next considerations, the HTS in Figure 1.3 will be used:

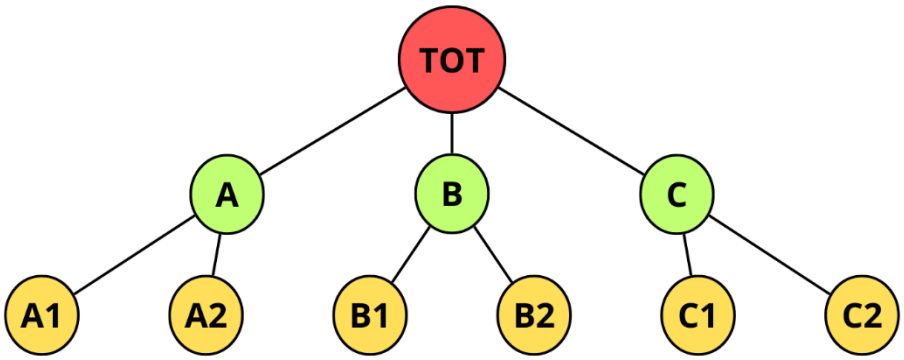


*Figure 1.3*

This simple example has 6 series as the bottom level, which can be written as a vector in $\mathbb{R}^6$, but has a total of 10 series describing the whole scenario. The notation used for HTS by, among others, Athanasopoulos et al. (2009) and the geometric interpretation introduced by Panagiotelis et al. (2021) will be used in the following considerations.

Let $n$ be the number of series in the HTS, $m$ be the number of series at the bottom level, $K$ be the number of levels and $Y_{i,t}$ be the vector containing the values of series $i$ at time $t$. If all the observation are stacked for the same time $t$, $Y_t \in \mathbb{R}^n$ is obtained, which is the vector with all the values of the HTS at that time step. Once a summing matrix $S$ of order $n \times m$ is defined, which aggregates the bottom level series all the way up to the hierarchy, and having the vector containing the observations of all bottom-level series at time $t$ be $b_t \in \mathbb{R}^m$, we have:

$$Y_t = Sb_t$$

*Equation 1.3*

Using the example in figure 1.3:

$$\begin{bmatrix} Y_{tot} \\ Y_A \\ Y_B \\ Y_C \\ Y_{A1} \\ Y_{A2} \\ Y_{B1} \\ Y_{B2} \\ Y_{C1} \\ Y_{C2} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Y_{A1} \\ Y_{A2} \\ Y_{B1} \\ Y_{B2} \\ Y_{C1} \\ Y_{C2} \end{bmatrix}$$

*Equation 1.4*

As already mentioned, the HTS data must adhere to linear constraints. To properly define this, the concept of coherent subspace is needed.

A coherent subspace is the $m$-dimensional linear subspace $\mathfrak{s} \subset \mathbb{R}^n$ for which some linear constraints hold for all $Y \in \mathfrak{s}$. Using this definition, a hierarchical time series can be formally defined an $n$-dimensional multivariate time series such that all the observed values $Y_1, \dots, Y_T$ and all future values $Y_{T+1}, Y_{T+2}, \dots$ all lie in the coherent subspace, so $Y_t \in \mathfrak{s}\ \forall\ t$.

Of course, the linear constraints that define the HTS are the aggregation: the sum of the values at any level must be equal to the level just above, and so on and so forth. With that being said, one can also notice that the above-given definition of HTS does not need data to follow any kind of hierarchy, and there is no reference to aggregation whatsoever. This more general definition is useful for tackling grouped TS, or structures that involve both the cross-sectional and temporal dimensions. One last thing, even if it refers to time series, the definition can be generalized to any vector-valued data in which some linear constraints hold for all their realisations.

## 1.3 FORECAST AND RECONCILIATION

The trickiest part of remaining in the coherent subspace when dealing with HTS is the forecast.

Let $\hat{\mathrm{Y}}_t(h) \in \mathbb{R}^n$ be the vector of point forecast estimate $h$ steps ahead for all the series in the HTS, using information available up to and including $t$: one has a coherent point forecast when $\hat{\mathrm{Y}}_t(h)$ belongs to the coherent subspace $\mathfrak{s}$.

So, the most critical step is the reconciliation, which transforms our predictions into coherent vectors. All the reconciliation techniques known can be represented through a matrix $P$, and so all the hierarchical forecast methods can be formulated like in Equation 1.5:

$$\tilde{Y}_t(h) = SP\hat{Y}_t(h)$$

*Equation 1.5*

$S$ is the $n \times m$ summing matrix, $P$ is a $m \times n$ matrix whose role changes depending on the reconciliation approach chosen, $\hat{Y}_t(h)$ is the forecasted vector of all the series of the HTS at times $t+1, \dots, t+h$, and finally $\tilde{Y}_t(h)$ is the reconciled vector that belongs to the coherent subspace $\mathfrak{s}$ of forecast of the HTS for times $t+1, \dots, t+h$. Notice that in some papers, the matrix $P$ is noted as $G$.

The $SP$ matrix is a linear mapping that has the $\mathfrak{s}$ coherent subspace as its image (Panagiotelis, et al., 2021). These two matrices can be interpreted as the two-step process of reconciliation: in the first one, the base forecasts $\hat{Y}_t(h)$ are combined to form a new set of bottom-level forecasts, which are then mapped to a full vector of coherent forecasts via $S$.

### 1.3.1 CLASSICAL METHODS

One the most applied method for hierarchical forecasting is the bottom-up approach, with some examples being Dangerfield & Morris, (1992), Shlifer & Wolff (1979), and Zellner & Tobias (2000). The idea is to simply forecast the bottom level and then aggregate the results using the summing matrix. With this technique the matrix $P$ is shown in Equation 1.6:

$$P = \left[0_{m\times(n-m)} \middle| I_m\right]$$

*Equation 1.6*

Where $0_{i\times j}$ is the null $i \times j$ matrix, and $I_m$ the identity matrix of order $m$. In this scenario $P$ just has the role to extract the bottom level forecast to then aggregate them for a coherent vector. The biggest advantage is that by modeling the entire bottom level, there is no information loss due to aggregation, and the dynamics of the individual series are captured. The biggest issue is the noisy data that usually are encountered in the bottom-level observation, becoming more challenging to model.

Another common approach is the top-down approach: the forecast of the total level (so the highest one) is disaggregated in the bottom-level series, based on the historical proportions of the data. Using the usual notation, $P$ is now the one in Equation 1.7:

$$P = \left[p \middle| 0_{m\times(n-1)}\right]$$

*Equation 1.7*

Where $p = [p_1, \dots, p_m]$ are a set of proportions for the bottom-level. Here the $P$'s role is to distribute the top-level forecast for the bottom-level series.

The proportion can be computed in different ways, and just like shown by Gross & Sohl (1990), a first one could be the ones shown in Equation 1.8

$$p_j = \sum_{t=1}^{T} \frac{Y_{j,t}}{Y_{tot,t}} \cdot \frac{1}{T}$$

*Equation 1.8*

Here each proportion $p_j$ is the average of the historical proportions of the bottom level series ($Y_{j,t}$) over the period $t = 1, \dots, T$ relative to the total aggregate ($Y_{tot,t}$), so the vector $p$ is the average historical proportion.

A second approach that we can use on the proportions is the one in Equation 1.9:

$$p_j = \frac{\sum_{t=1}^{T} \frac{Y_{j,t}}{T}}{\sum_{t=1}^{T} \frac{Y_{tot,t}}{T}}$$

*Equation 1.9*

In this case, the $p_j$ represent the average historical value of the bottom-level series relative to the average value of the total aggregate. The vector $p$ reflects the proportions of the historical averages.

The top-down approaches are easy to implement and return reliable forecast, but there is a great deal of information loss due to the aggregation. We do not take advantage of the individual series characteristics, such as time dynamics or special events.

Many businesses end up using a hybrid technique, the so-called middle-out approach (Hyndman, et al., 2011): the idea is to use the forecast for a middle-level of the hierarchy, and then aggregate it in the top levels with a bottom-up approach, and disaggregate them to the bottom levels using a top-down approach.

### 1.3.2 OPTIMAL RECONCILIATION

From the same paper of Hyndman, et al. (2011), another approach is proposed: it first forecasts all the levels of the series at all levels of aggregation independently, which will be referred to as base forecasts, and then uses a regression model to optimally combine these predictions to make them coherent. This technique is now known as optimal reconciliation.

One first consideration made in the paper is on the unbiasedness of the forecasts. Let us now assume that $\hat{Y}_T(h)$, the base (independent) forecasts are unbiased ($\mathbb{E}\left[\hat{Y}_T(h)\right] = \mathbb{E}[Y_T(h)]$), and that the reconciled forecasts need to be unbiased too, then it must hold, given $Y_{K,T}(h)$ the real values for $t = T + 1, \dots, T + h$ of the bottom level $K$ of the series, the Equation 1.10:

$$\mathbb{E}[\tilde{Y}_T(h)] = \mathbb{E}[Y_T(h)] = S\mathbb{E}[Y_{K,T}(h)]$$

*Equation 1.10*

Suppose that the mean of the future values of the bottom level $K$ is $\beta_T(h) = \mathbb{E}[Y_{K,T+h}|Y_1, \dots, Y_T]$, then one has $\mathbb{E}[\tilde{Y}_T(h)] = SP\mathbb{E}\left[\hat{Y}_T(h)\right] = SPS\beta_T(h)$. The only conclusion is that the unbiasedness for the reconciled forecasts will hold if Equation 1.11 holds:

$$SPS = S$$

*Equation 1.11*

This is true for the bottom-up approach with the aforementioned $P$, but it is not for the top-down one: no matter what $p$ vector is chosen, one always has $SPS \neq S$, so their revised forecast will always be biased even if the base forecasts are not.

Knowing the variance of the reconciled forecast one could also compute the prediction intervals: with $\Sigma_h$ being the variance of the base forecasts $\hat{Y}_T(h)$ we have Equation 1.12:

$$Var[\tilde{Y}_T(h)] = SP\Sigma_h P'S'$$

*Equation 1.12*

Moving on to the forecasts, one can write the base ones as Equation 1.13:

$$\hat{Y}_T(h) = S\beta_T(h) + \varepsilon_h$$

*Equation 1.13*

Here we have $\beta_T(h) = \mathbb{E}\left[Y_{K,T+h}\middle|Y_1, \dots, Y_T\right]$ which is the unknown mean of the bottom-level $K$, and $\varepsilon_h$ has 0 mean and covariance matrix $\Sigma_h$. All this suggests that one can estimate $\beta_T(h)$ by treating the above equation as a regression equation that can be used to forecast all the levels of the series.

Under the assumption that the errors $\varepsilon_h$ in the equation can be expressed as the aggregation of the bottom level $K$ errors, so $\varepsilon_h \approx S\varepsilon_{K,h}$, the computation of $\Sigma_h$ is simplified. Without this assumption, the difficulty of estimating $\Sigma_h$ can hinder the whole process. This way we are assuming that the errors follow the same linear constraints as the original data, which is reasonable if the forecasts also satisfy the same constraints.

Under these assumptions the most meaningful result of the paper is that one can use the ordinary least squares (OLS) when computing the reconciled forecast, without the need to for an estimate of the underlying covariance matrix. With these results, the final computation is in Equation 1.14:

$$\tilde{Y}_T(h) = S(S'S)^{-1}S'\hat{Y}_T(h)$$

*Equation 1.14*

So, using this technique we have $P = (S'S)^{-1}S'$.

Optimal reconciliation was once again tackled by Wickramasuriya et al. (2019). Their objective was to minimize the sum of variances of the reconciled forecast errors under the property of unbiasedness.

Considering the $h$-step ahead base forecast errors, $\hat{e}_T(h) = Y_T(h) - \hat{Y}_T(h)$, and the reconciled forecast errors, $\tilde{e}_T(h) = Y_T(h) - \tilde{Y}_T(h)$, then for any $P$ such that $SPS = S$ their covariance matrix is given by Equation 1.15:

$$var[Y_T(h) - \tilde{Y}_T(h)|Y_1, \dots, Y_T] = SPW_hP'S'$$

*Equation 1.15*

Where $\tilde{Y}_T(h) = SP\hat{Y}_T(h)$ and $Y_T(h)$ are the observed values for times $t = T+1, \dots, T+h$, and $W_h = \mathbb{E}[\hat{e}_T(h)\hat{e}'_T(h)|Y_1, \dots, Y_T]$ the variance-covariance matrix of the $h$-step-ahead forecast errors. This way we have an expression for the variance of the coherent forecast errors: this can be used to generate the confidence intervals for any hierarchical or grouped TS forecast that generate unbiased coherent forecasts.

The goal of the paper was to find the value of $P$ that minimizes the trace of $var[Y_T(h) - \tilde{Y}_T(h)|Y_1, \dots, Y_T]$ in order to have the best linear unbiased reconciled forecasts. This is why this approach is called the MinT (minimum trace) reconciliation.

The final result found is, given $W_h$ the positive definite covariance matrix of the $h$-step-ahed base forecast errors, in Equation 1.16:

$$P = (S'W_h^{-1}S)^{-1}S'W_h^{-1}$$

*Equation 1.16*

As a result, the reconciled forecasts become Equation 1.17:

$$\tilde{Y}_T(h) = S(S'W_h^{-1}S)^{-1}S'W_h^{-1}\hat{Y}_T(h)$$

*Equation 1.17*

Even though $W_h$ is computable, it is not easy to do so, that is why there is a series of alternatives that can be used as a substitute:

- $W_h = k_h I, \forall h$, with $k_h > 0$, is the most simplifying assumption and collapses the method the OLS by Hyndman et al (2011). This is optimal only in special situations, such as when the base forecast errors are uncorrelated and equivariant.
- $W_h = k_h diag(\widehat{W}_1), \forall h$ with $k_h > 0$ and

$$\widehat{W}_1 = \frac{1}{T}\sum_{t=1}^{T} \hat{e}_t(1)\hat{e}_t(1)'$$

*Equation 1.18*

  In this case MinT can be described as a WLS (weighted least squares) estimator.
- $W_h = k_h \Delta, \forall h$, with $k_h > 0$ and $\Delta = diag(S1)$ where $1$ is a unit column vector. This estimator depends only on the grouping structure of the collection. Its advantage over OLS is that it assumes equivariant forecast errors only on the bottom level of the series.
- $W_h = k_h \widehat{W}_1, \forall h$, with $k_h > 0$, the unrestricted sample covariance estimator for $h = 1$.
- $W_h = k_h \widehat{W}^*_{1,D}, \forall h$, with $k_h > 0$, $\widehat{W}^*_{1,D} = \lambda_D \widehat{W}_{1,D} + (1 - \lambda_D)\widehat{W}_1$ is a shrinkage estimator with diagonal target, $\widehat{W}_{1,D}$ is a diagonal matrix comprising the diagonal entries of $\widehat{W}_1$, and $\lambda_D$ is the shrinkage intensity parameter.

The glaring problem of these kinds of techniques is the necessity to compute the forecast for all the series of the HTS, and also requiring inverting matrices, both being computationally intensive tasks.

### 1.3.3 MACHINE LEARNING APPROACH

Another interesting approach was studied by Mancuso et al (2021) which uses machine learning techniques and neural networks that are capable of automatically extracting the relevant features of the hierarchy while enforcing coherence. It is also capable of using external variables at any level of the HTS. One of the biggest downsides is, with it being a ML technique, it requires a good amount of data points to be viable, i.e. a few years of daily observations.

The proposed model takes as input any aggregate series $y_{t,j}^{k,p}$ (so corresponding to node $j$ at level $k$ of the HTS and connected to the parent node $p$ at level $k-1$) and the vector of series $y_t^{k+1,j}$ (which contains the values of all the series at level $k+1$ connected to the previous one) and it is then divided into two steps:

- The training part, when the best forecasting method for the aggregated series, $F^*$, is chosen, while a NN is trained on the aggregated series plus the exogenous variables (training input) and the corresponding disaggregated series (training target output). This network is called by the authors NND, neural network disaggregation, and is composed by a multilayer perceptron, which handles the exogenous variables, and a CNN for the time series itself: their results are then concatenated and fed to a series of fully connected layers that then return the output.
- The test part, when the test set is used by the $F^*$ model to get some forecasts of the aggregated series which are then fed to the NND to get the coherent disaggregated series.

The network weights need to minimize the loss, but we also want this constraint to hold:

$$y_{t,j}^{k,p} = 1^T y_t^{k+1,j} = 1^T \widehat{y_t^{k+1,J}} = \widehat{y_{t,J}^{k,p}}$$

*Equation 1.19*

Where $1$ is the vector of all ones of size equal to the number of series the aggregated one is split into.

To be able to use back propagation in the training and also ensure the coherence in the results, a special loss is introduced to penalize the difference between the sum of the lower-level observation and the sum of the lower-level predictions:

$$L\left(y_t^{k+1,j}, \widehat{y_t^{k+1,J}}\right) = \frac{1}{T}\left[(1-\alpha)\sum_{t=1}^{T}\left\|y_t^{k+1,j} - \widehat{y_t^{k+1,J}}\right\|^2 + \alpha\sum_{t=1}^{T}\left(1^T y_t^{k+1,j} - 1^T \widehat{y_t^{k+1,J}}\right)^2\right]$$

*Equation 1.20*

Where $\alpha \in (0,1)$ is the parameter that controls on relevance of each contribution in this loss function. For the paper it was set to $0.5$ and other scenarios were not explored.

In the paper is shown how the ML approach is an improvement over all the other techniques and give better results. It needs to be said that there is anyway the need to find a good statistical model for the testing part which has to be chosen properly.

## 1.4 SCOPE OF THE THESIS

This thesis aims at comparing the performance, using a real dataset of the sales of a company, of all the methods mentioned before, and provide a step-by-step approach on how to forecast HTS and provide good results. It also aims at trying a different approach to forecasting which will be explored later: the TBATS-NN hybrid.

The next chapter is dedicated to giving the problem the proper background and show all the exploratory analysis that were performed.

Following that, the third chapter focuses on how the best forecasting statistical model was found, checking the results at all the levels of the HTS.

In the fourth one, the forecast results are reconciled using the various approaches which were listed, and the results are confronted to evaluate the goodness of each one.

In the fifth and final chapter, the conclusions are shown and some future work that might be of interest is proposed.

# 2 EXPLORATORY ANALYSIS AND DATA CLEANING

## 2.1 PROBLEM BACKGROUND

Before I started working for the Horsa Insight company for the internship, another business requested as a service a forecasting algorithm for their products' sales. This software would then become a part of their data pipeline architecture. The goal was to forecast a year of the company sales by using some exogenous variables that the company was already able to predict themselves.

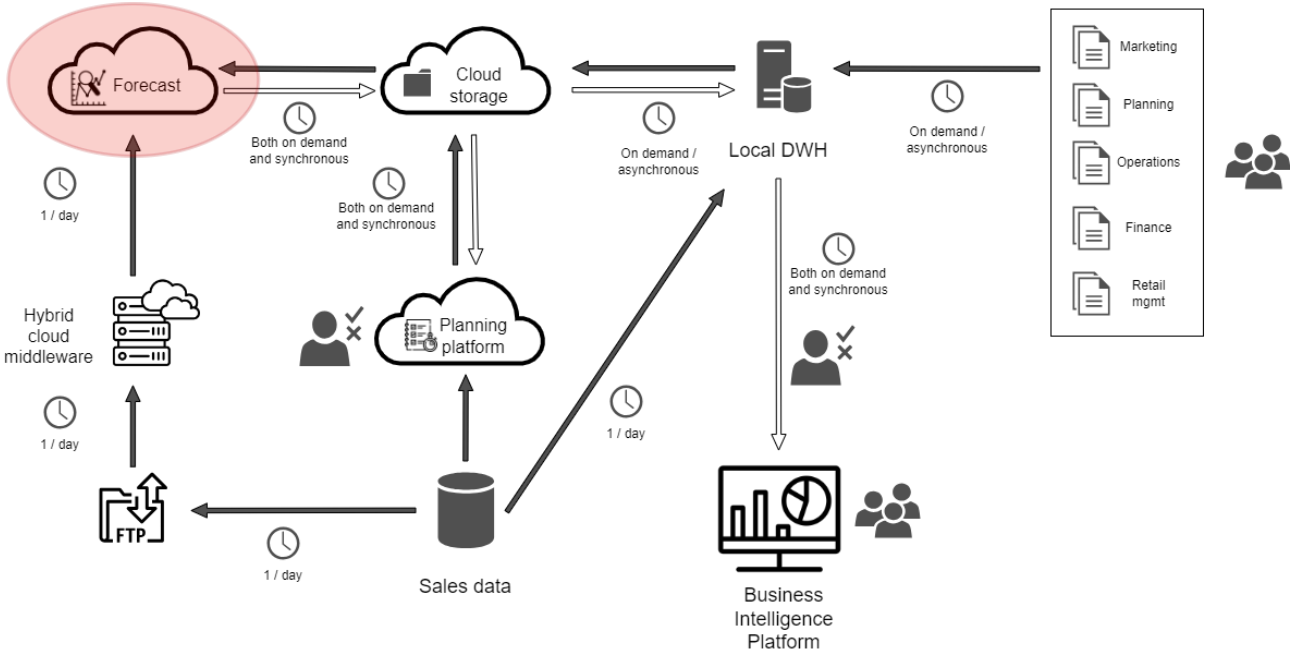


*Figure 2.1*

Following the black arrows, we can see that the forecasts are computed on the cloud using the sales data stored in a specific database and the information coming from the marketing and planning (besides others) department of the company. All that is stored on-premises and is linked to the cloud through a hybrid cloud middleware, which can be seen as a "frontier database".

The white arrows follow the flow of the forecasts once computed. They are used in the business intelligence processes (BI) and to enrich the planning platform. All the results are stored in both a cloud storage and a local DWH.

These processes and related data flows can be activated in two ways:

- On schedule, so every Sunday the forecasts are computed automatically.
- On demand by some manager, who through the planning platform can activate the pipeline and have the forecasts computed.

Having a clear idea on where, when and for what the forecasts are used, the objective was to try to find a better solution to the already implemented algorithm, which was a forecasting neural network. It was implemented in R using the Forecast package (Hyndman, et al., 2024), with the

function nnetar, so a special NN for time series analysis that uses previous values of the series in an autoregressive fashion and just one hidden layer. The client requested to work using Python.

Let us take a closer look at the data itself, which was properly anonymized using various techniques for the sake of this thesis. We are working with a sports clothing manufacturing company, which has weekly data on the sales of their products in different countries. The data is given in a CSV file which, when transformed into a data frame, results in a table with 1138100 rows and 43 columns. The different columns contain the following information:

- *Cal_date*: the date when the sales were computed, they are all Sundays, so the data refers to the last week's sales. From now on, when referring to a period of time using dates, this notation which uses the last day of the week will be kept.
- *Macroarea*: one of 3 regions the data is divided into, which are EMEA (Europe, Middle East, and Africa), AMER (America) and APAC (Asia Pacific).
- Area: all the macro areas have been further divided into other areas, which are Europe (EU) and Middle East (ME) for EMEA, Latin America (LA) and North America (NA) for AMER, and Greater China (GC) and Asia Pacific (AP) for APAC
- *Country*: one of the 62 specific countries where the products were sold.
- *Product_category*: the type of product sold, which can be clothes (CL), footwear (FW) or accessories (AC).
- Product: one of the 25 specific products the sales are related to.
- *Mark_down_code*: either Y or N (yes or no), so if the product sold was discounted or not.
- *Clientele_area*: if the product was sold to a fidelized (F) or non-fidelized (NF) customer. This data was collected through a loyalty card that a customer could use.
- *Amt_euro*: the actual money gained by the sales, it is our response variable.
- *Days_closed_covid_weighted*: number of days the shops were closed due to the COVID pandemic.
- *Impatto_covid*: impact of the COVID pandemic on the sales.
- *Days_location_open*: the number of days the shops were open. This measure is proportional to the number of shops in the related country, so it gives information on the selling potential of the company in that location.
- 31 *Event* variables related to various festivities the different countries might have. Since the data were anonymized, the events were masked too to avoid being able to deduce the original data, so they are just numbered from 1 to 31.

While the calendar dates give us information on the time stamp $t$, this is clearly a Grouped TS which could be aggregated in many ways, since there is a nested geographical hierarchy (macro-areas, areas, country) with a crossed product grouping (product category, product, markdown and clientele). An example represented through a diagram could be as in Figure 2.2:

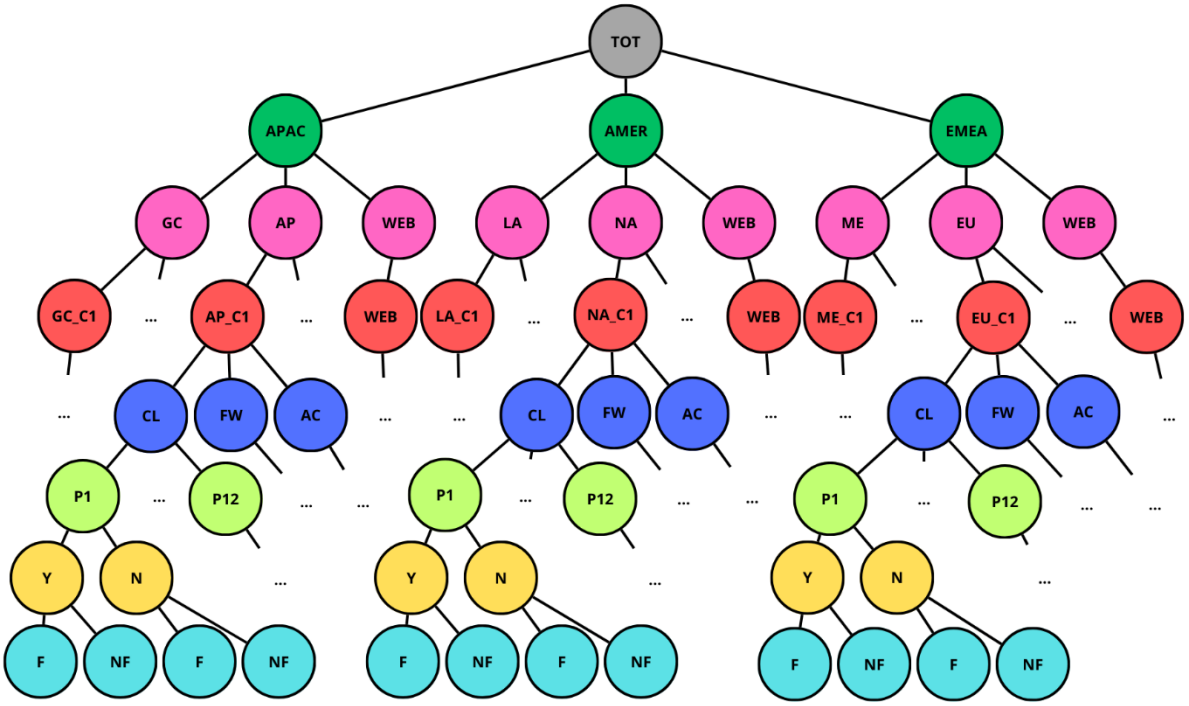


*Figure 2.2*

The diagram of the full HTS is big and complex and has a huge amount of series in the bottom level. Since the company asked to focus on forecasts of specific levels of the hierarchy, in particular the single countries sales disaggregated by single product, mark down code type, and clientele, the scenario is much simpler than the starting one The simplified splits are shown in the diagram in Figure 2.3:

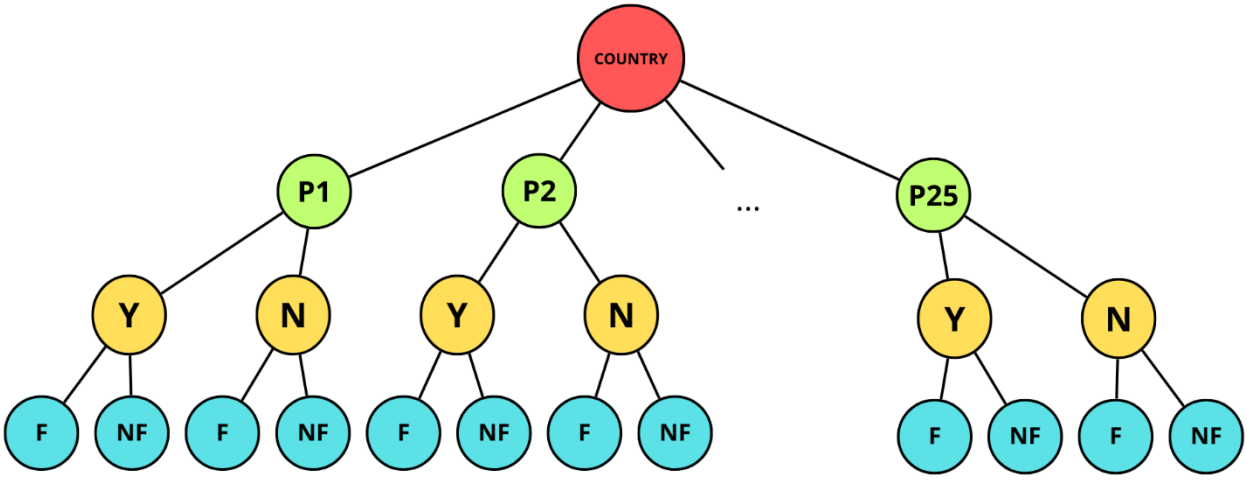


*Figure 2.3*

This way, if the client is ever interested in the product categories, area, macro-area, or total series, they can always use some bottom-up approach to aggregate the results of lower levels of the series and have unbiased estimates. Right now, with this split, there are as the bottom level for each week (so at each time step $t$) a vector of 100 elements for each country studied.

## 2.2 EXPLORATORY ANALYSIS

Before proceeding with the model selection for the forecasts, it is necessary to analyze the data at hand and properly clean it. All the graphs presented were created using the PLOTLY library from Python (Inc., 2015), and all the analyses were conducted using Python code.

### 2.2.1 RESPONSE VARIABLE

When dealing with the model selection, to assess the goodness of each option a simple train-test split will be used. The test set will be one year long, just like the client requested, so it is only needed to check if all the time series have enough data points to allow the analysis. Checking how many weeks each country's series has returns the graph in Figure 2.4:

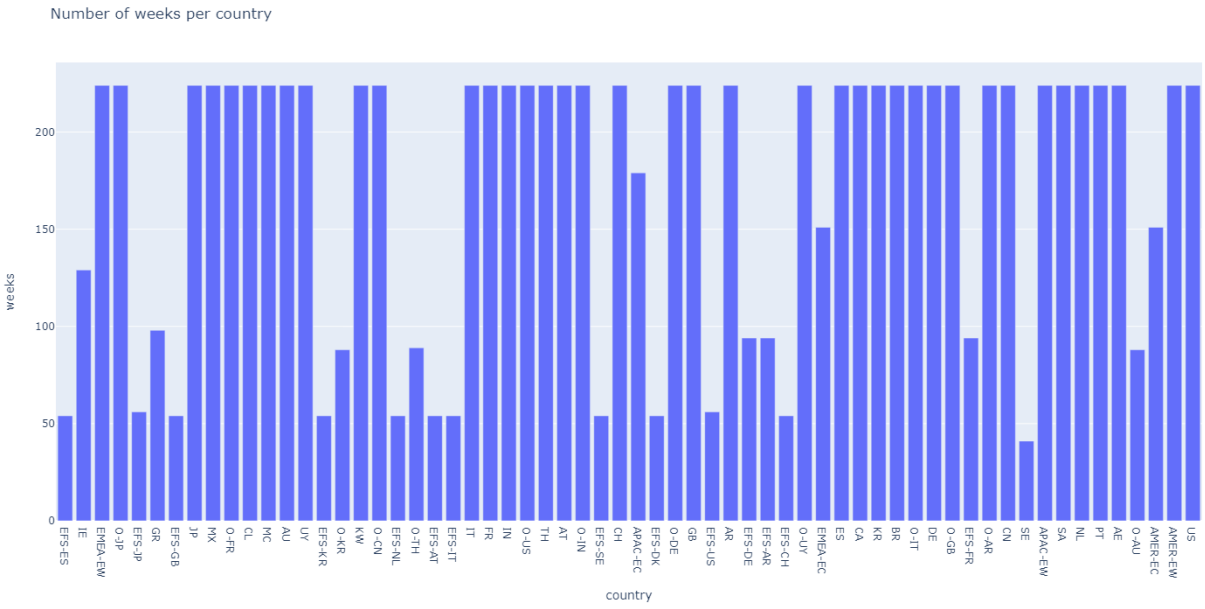


*Figure 2.4*

As one can see, of the 62 considered, many countries have less than 200 weeks of observation. This number is way too little since most of the models considered later have a seasonality component that requires data for at least two years to be evaluated properly. The decision to drop all the series with less than 200 observations was made, ending up with 39 series having 224 weeks to work with (Figure 2.5).

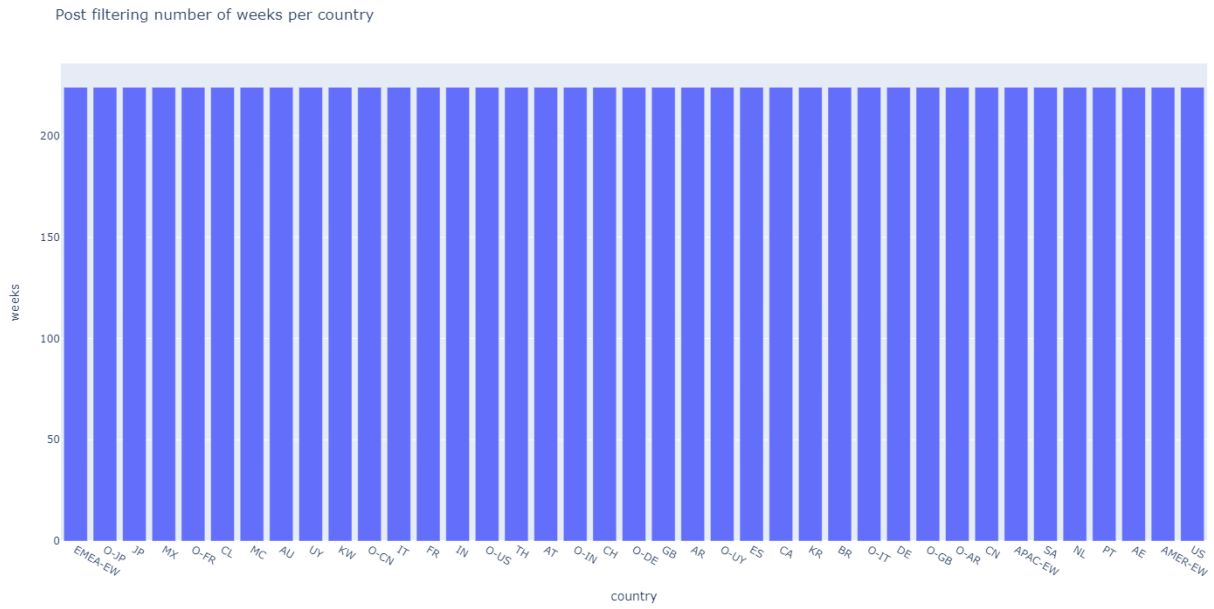


*Figure 2.5*

The next step is investigating how the various levels of interest of the series, once aggregated, behave.

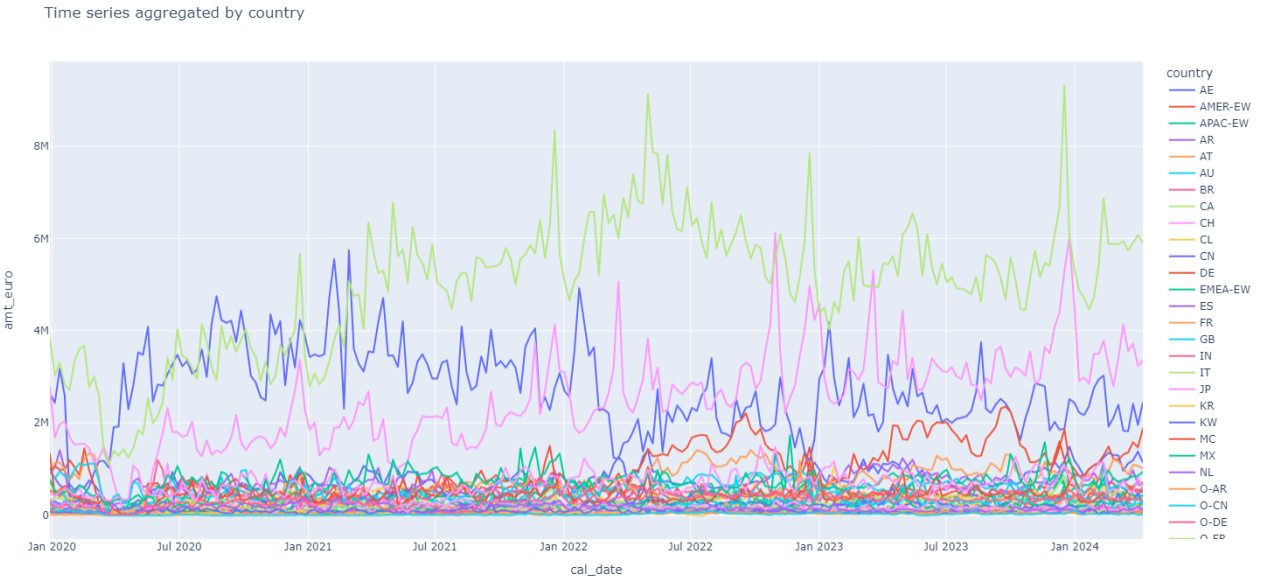


*Figure 2.6*

This procedure started by looking at the series for all the single countries, obtaining Figure 2.6 plot. Even though the graph is messy, some information can still be extracted: the periods start from December 29, 2019, and end with April 7, 2024. One can also see the effect of the COVID pandemic at the start 2020. The series were cropped to avoid considering it since it is a really special event, so all of them now start on 7 June 2020 (Figure 2.7).

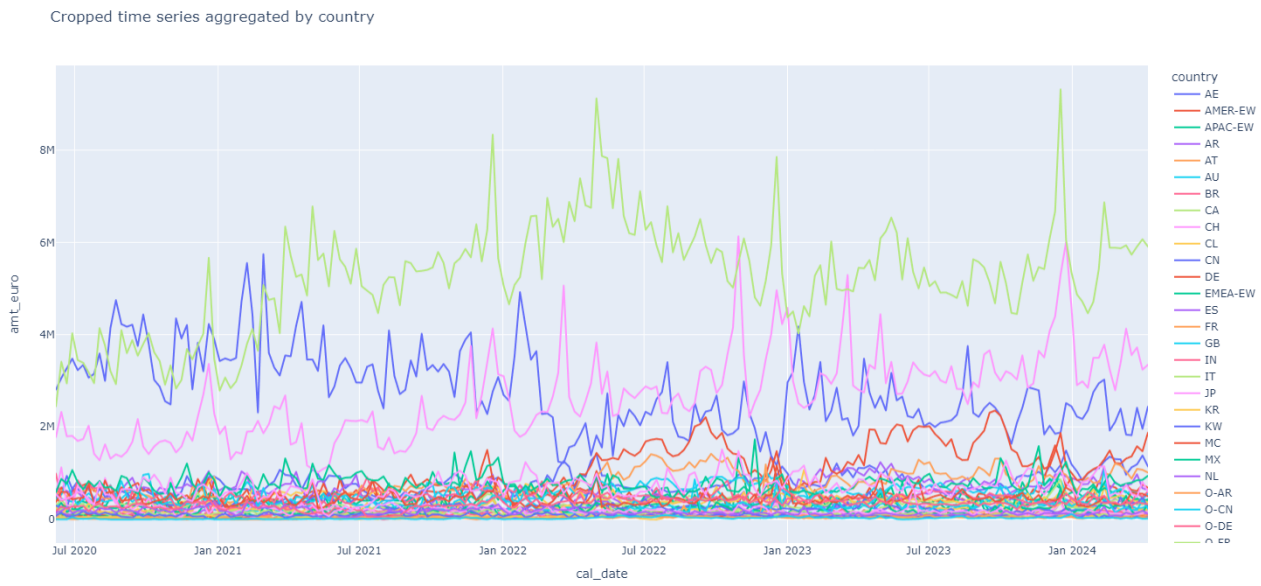


*Figure 2.7*

This change reduces the observation per data series from 224 to 201: while this might seem like a loss of information, considering that (hopefully) no other event like the pandemic should happen in the foreseeable future, there should not be any real change in the forecast eliminating those data points. Notice that the HTS will not be called cropped from now on.

All the series have different magnitude and behaviour, so looking at this graph there are no real other info that can be extracted.

The next level of interest of the HTS is the products, which immediately prove problematic: indeed, looking at the graphs related to their total sales worldwide, it has been noticed that some of them have really peculiar behaviours.

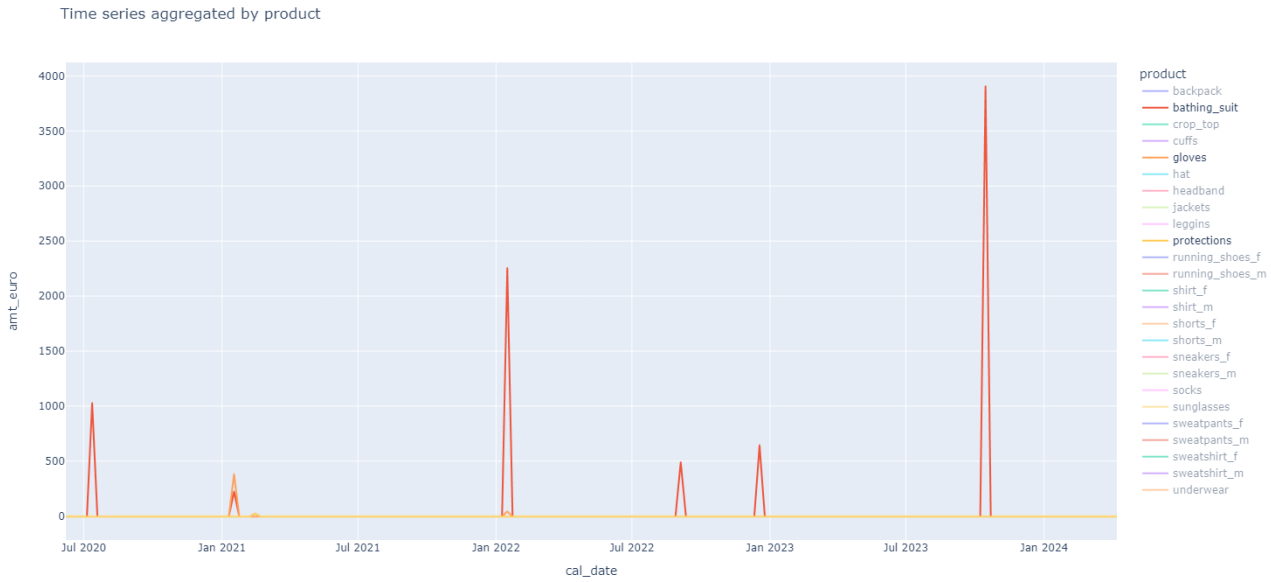


*Figure 2.8*

One can see in Figure 2.8 that the bathing suits, protections and gloves just have some sales spikes in a few dates: eliminating them from the model could only be beneficial, since in the most optimistic future, the model would just predict them as an almost-zero line. We can also see from the next graphs that their sales are negligible when compared to the other products. These are not the only merchandise that were eliminated:

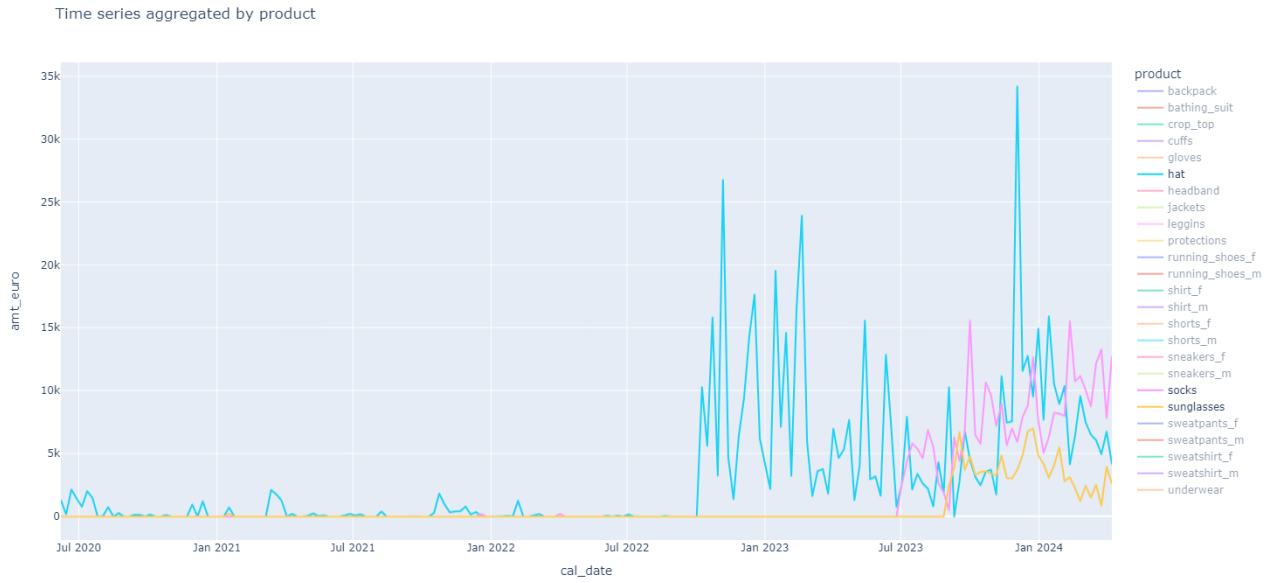


*Figure 2.9*

In Figure 2.9 it is clear that some products have been recently added to the repertoire of the company, like sunglasses and socks, or have been lately gaining popularity, like hats. The meaningful observations for this merchandise would be almost completely relegated to the test set of the series, so the training procedure would most likely not be able to model their behaviour properly. For these reasons, these products will also be excluded from the dataset. After eliminating these 6 products, the dataset is left with 19 merchandise and the time series for their worldwide sales is shown in Figure 2.10:

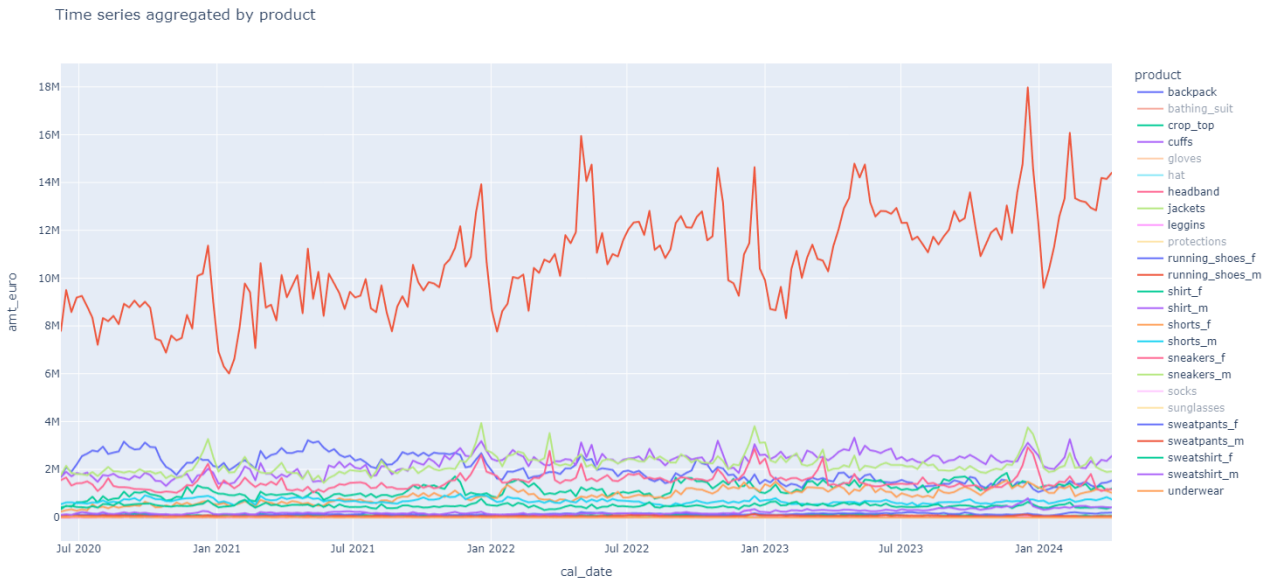


*Figure 2.10*

Further investigating the products' behaviour, it was checked which are the top selling products:

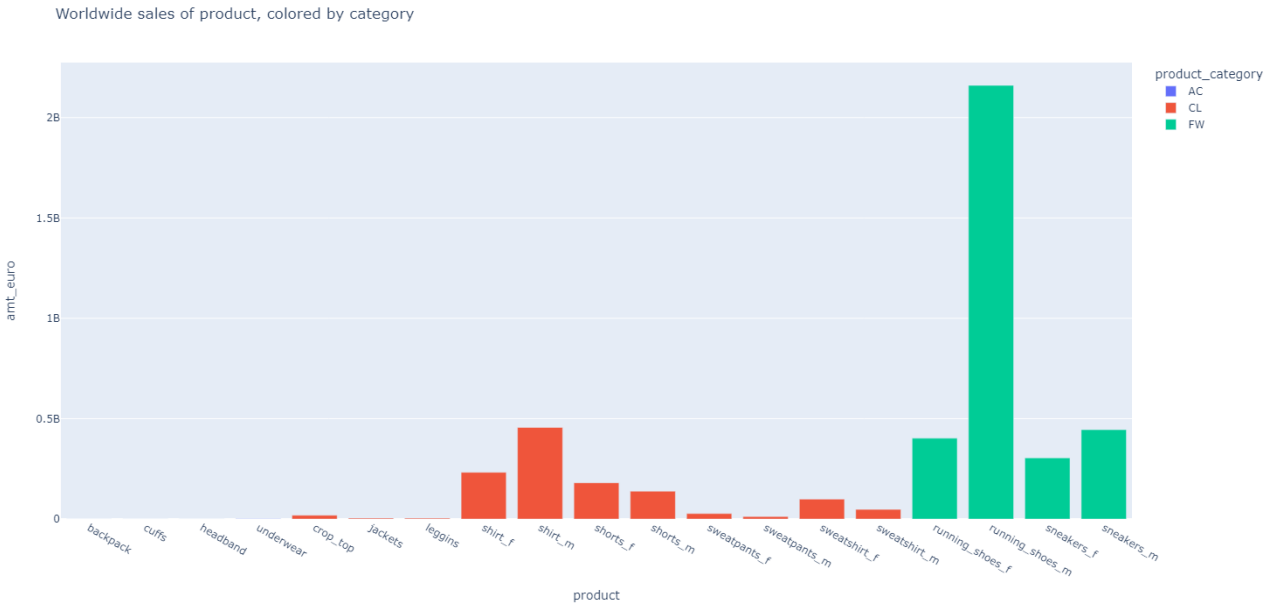


*Figure 2.11*

The conclusion one can deduct from Figure 2.11 is that the accessories do very poorly, while the sales of footwear is the leading income source of the company.

Another factor of disaggregation is the markdown, which refers to the products being discounted or not when sold, so we can check with the bar plot in Figure 2.12 in every country how the series splits:

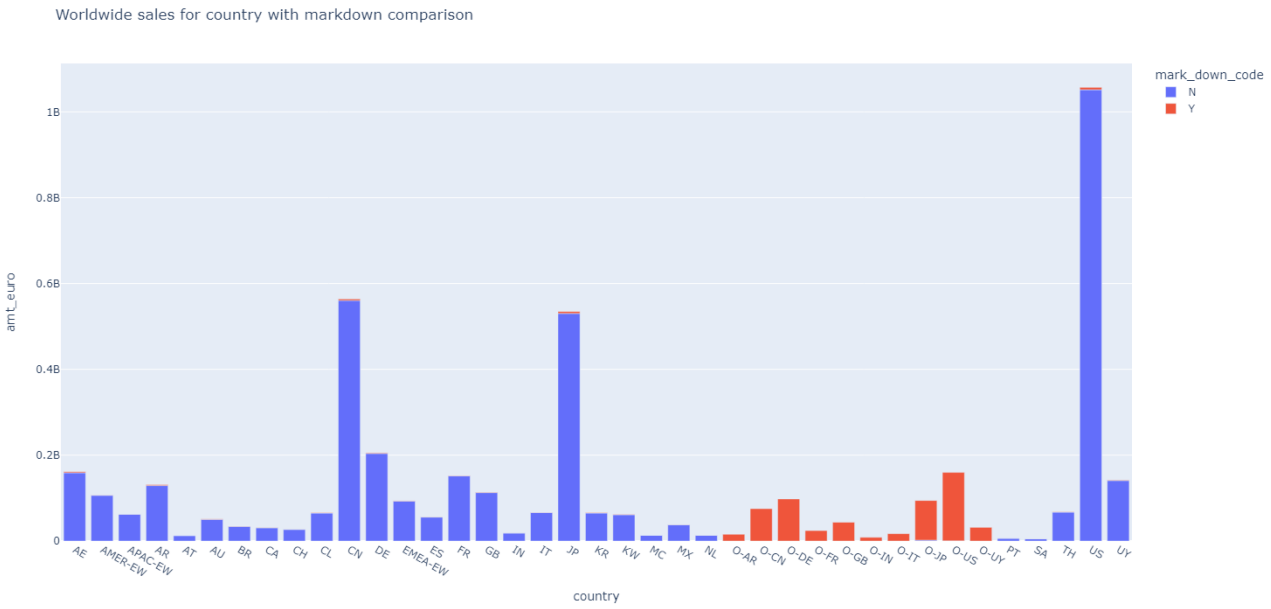


*Figure 2.12*

The products which are on sale are effectively sold almost entirely in the outlets of the country (the ones with the "O-"), so it might be a good idea to eliminate this aggregation level and split these sales just by looking at where the merchandise was sold. Indeed, upon further investigation, it was discovered that the minuscule portion of the discounted products that are not sold in the outlets are just a legacy of the company's old classification system: since they are really little and probably a mistake, we decided to incorporate them together with the non-sales ones. After this the markdown column was eliminated, as it is of no use.

The last disaggregation level is the clientele area, as in Figure 2.13:

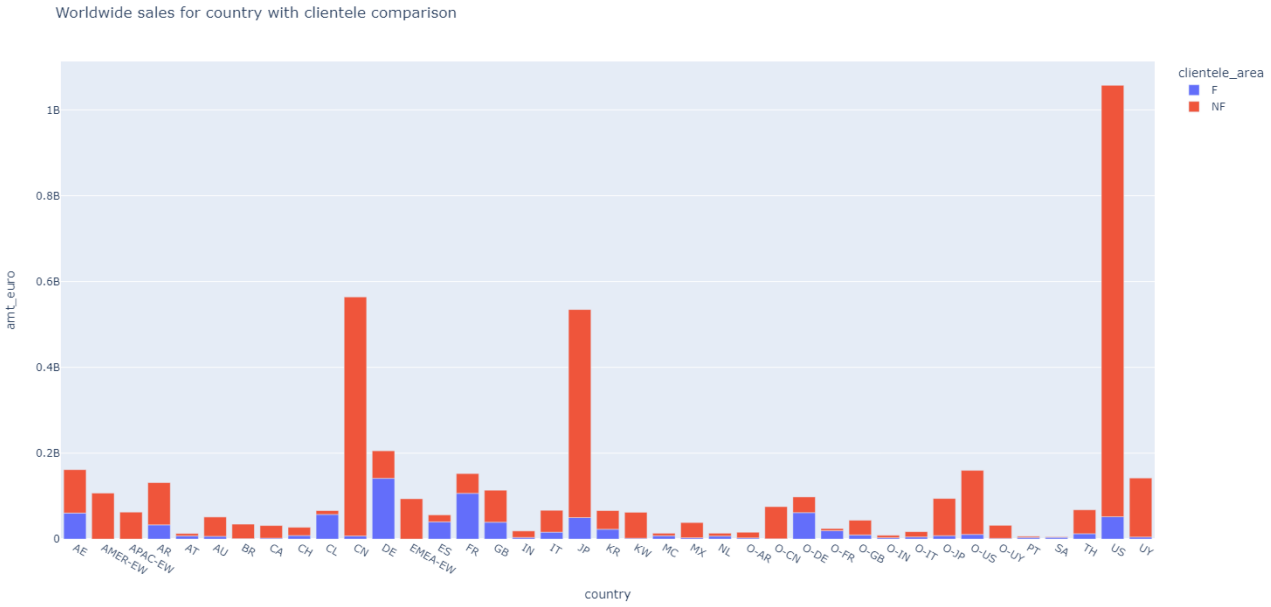


*Figure 2.13*

The splits here do not have a clear pattern, one can just see that there is a majority of sales by non-fidelized customers.

Now that the data was properly cleaned, it is interesting to identify the top selling countries, to have an idea of what are the most interesting ones to look out for.

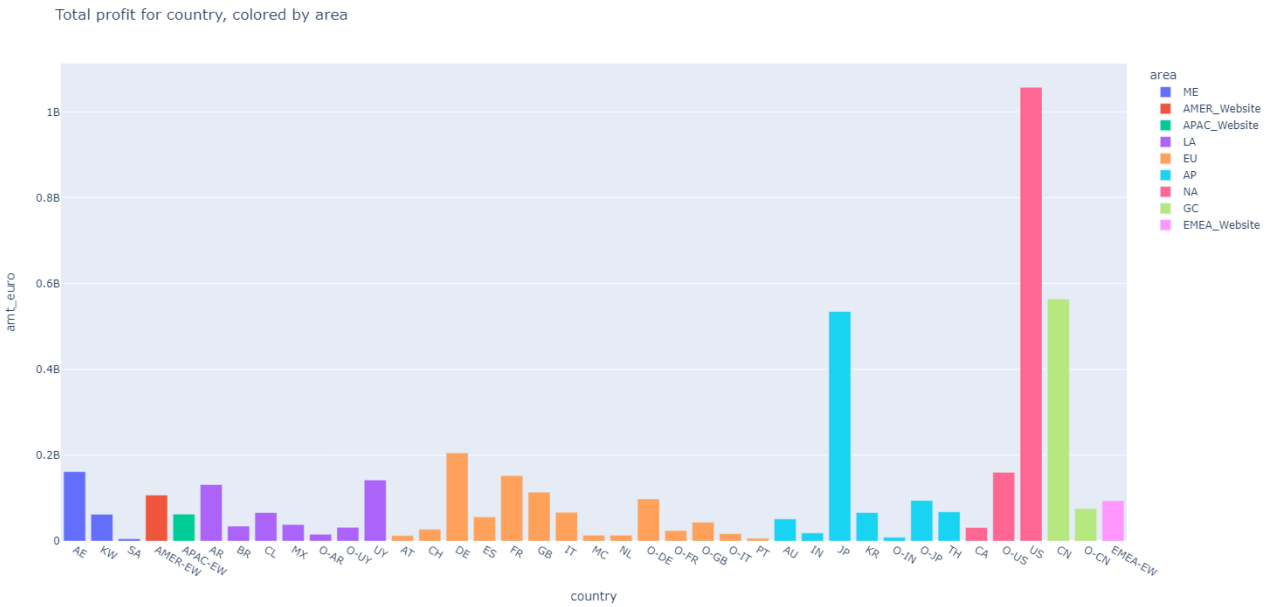


*Figure 2.14*

As we can see in Figure 2.14 the top 3 selling countries are US, CN and JP. Notice that even if this graph might suggest that the different macro areas have really different sales, confronting them stacked gives a clearer idea of how exactly they are distributed, just like shown in Figure 2.15:

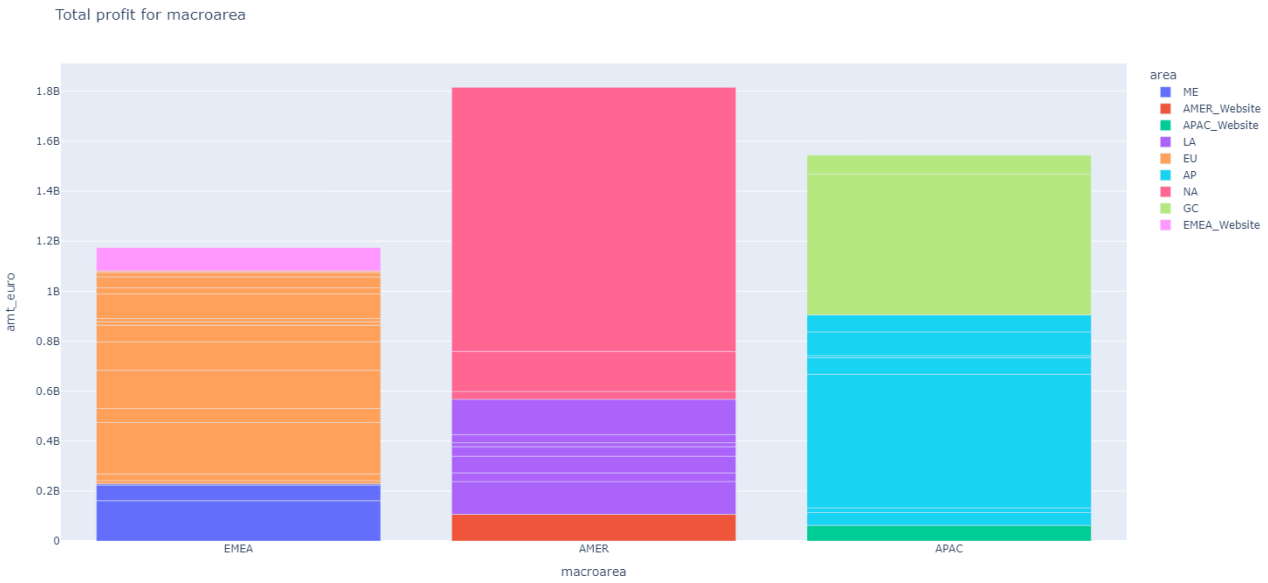


*Figure 2.15*

This confirms that the company's profit comes mainly from America, but the other two macro areas still make up a large slice of the income.

So, at the end of this preliminary procedure, the diagram of the HTS has changed significantly and turned into the graph in Figure 2.16:

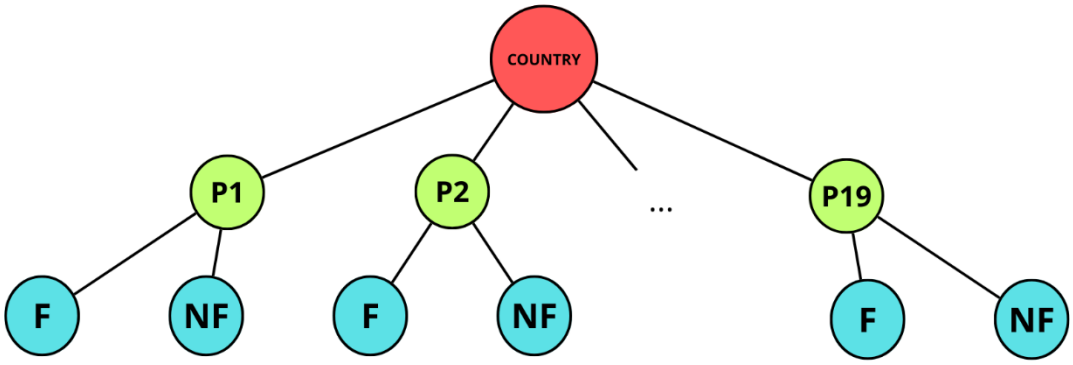


*Figure 2.16*

After the data cleaning done beforehand, for each timestamp $t$ and country we have a bottom level composed of 38 observations, so $b_t \in \mathbb{R}^{38}$. The data table has now 297882 rows and 42 columns.

### 2.2.2 EXPLANATORY VARIABLES

The explanatory variables need to be checked too to eliminate redundant ones and make sure there are no issues. There are a total of 34 exogenous variables. The first thing to notice is that these variables are all equal for the same timestamp $t$ and country, so all the bottom level series share the same vector $x_t \in \mathbb{R}^{34}$. Here in Figure 2.17 is shown an example of one of the top sellers, the US, but the next consideration can be extended to all of the other countries:

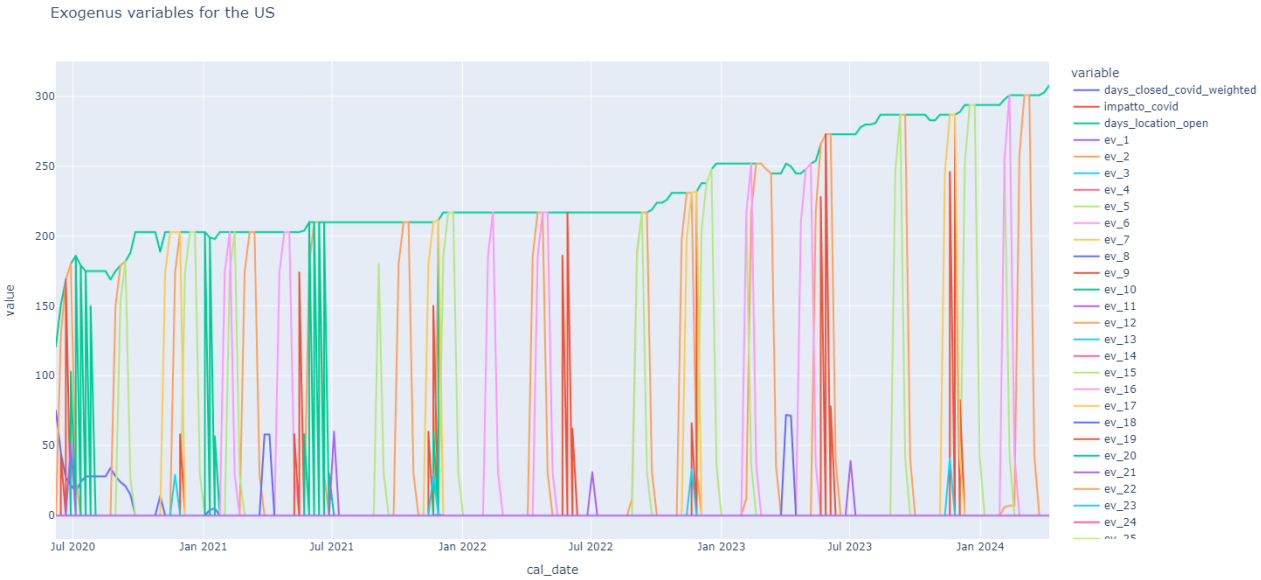


*Figure 2.17*

The first thing noticeable in Figure 2.17 is that the various events are, at most, equal to the *days_location_open* variable: this is because they are proportional to the number of shops the event

affects. All the events' variables have this spiking behavior because they are related to non-seasonal occurrences and festivities, or some special promotion launched by the company.

Another important aspect is clearer once all the event variables are not shown in the graph, as in Figure 2.18:

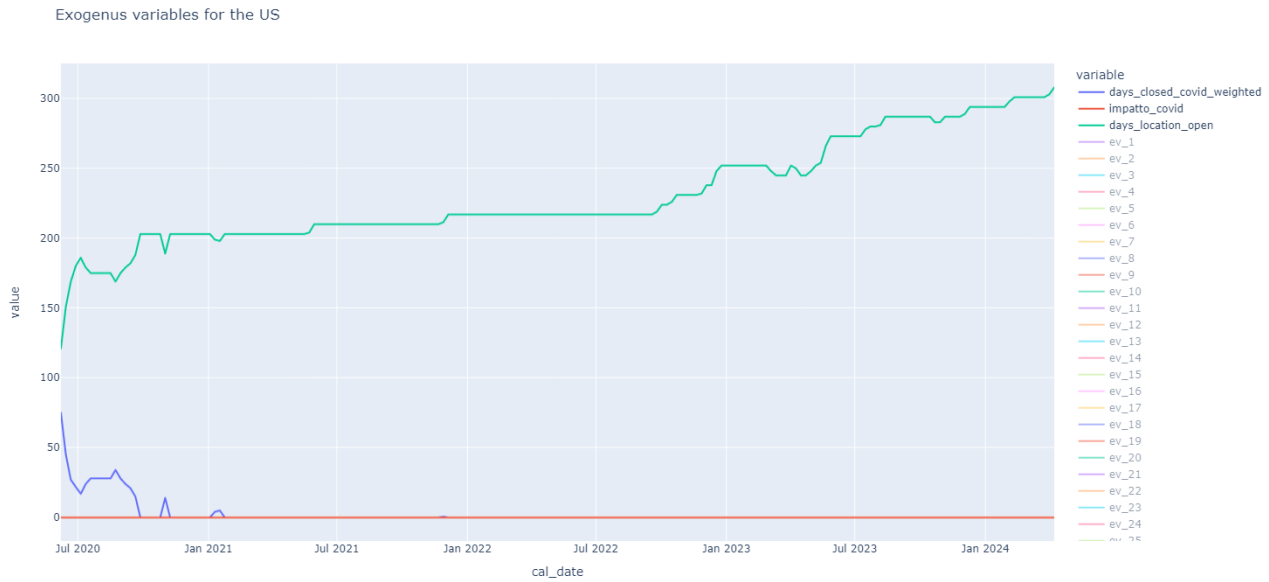


*Figure 2.18*

The *days_closed_covid_weighted* is in a way complementary to the *days_location_open* variable, while the *impatto_covid* variable is always zero. The latter info is more surprising, but it is most likely an error, and all the COVID impact information are stored in the *days_closed_covid_weighted*. With that being said, since the series was cropped to avoid part of the data that was the most affected by COVID, and since *days_closed_covid_weighted* seems a piece of redundant information already given by the *days_location_open*, both the pandemic-related regressors are dropped out of the data set.

With these last considerations, the preliminary steps of the analysis are over: the final data frame we are working with has 297882 rows and 40 columns, out of which 32 refer to the exogenous variables.

# 3 FORECASTING MODELS

The first step for the HTS forecasting is finding a suitable model that is able to produce good predictions using the external variables as requested by the client. As previously mentioned, after this it is necessary to use a reconciliation method to obtain coherent forecasts which are the key aspect of the HTS.

With data at hand, it is interesting to see an example of how the various splits contain different information and have different behaviors, to further motivate why such complex techniques are used to get coherent results and why all the levels of each country's disaggregation are forecasted independently. Suppose one is interested in the original hierarchy shown in Figure 3.1 (not the one asked by the client) and decides to check the top level of the HTS, the total aggregation over all the levels in the series:

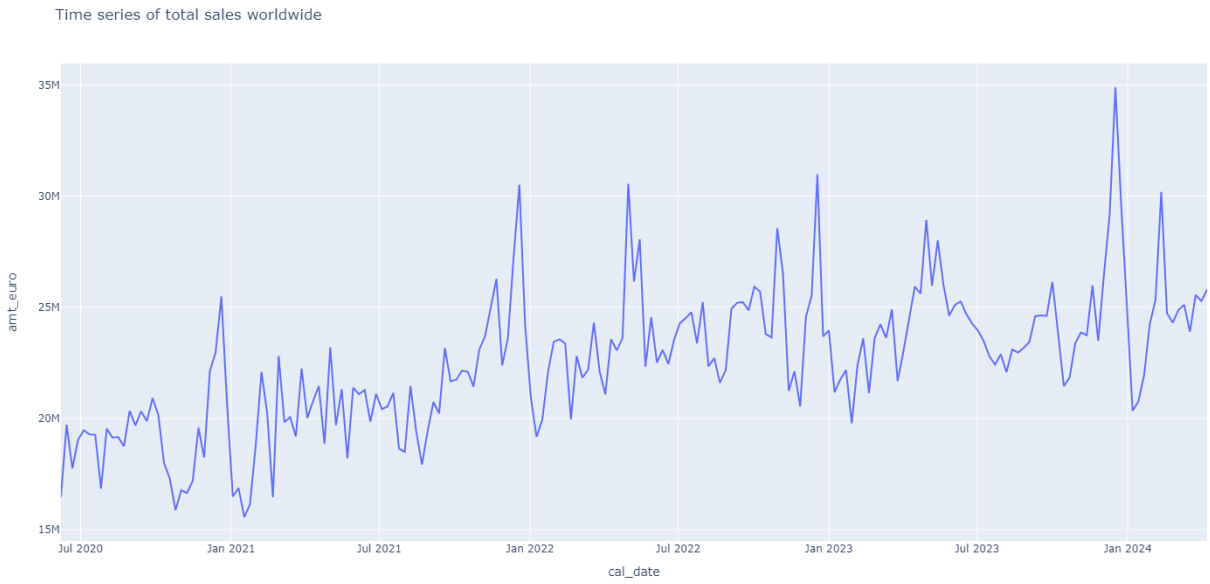


*Figure 3.1*

Looking at Figure 3.1 one can see a clear linear increasing trend, and the spikes in the periods of January seem to suggest there is a seasonality component. It might seem like there is no reason for further investigations, but the scenario is really different once the data is disaggregated per macro-area (Figure 3.2Figure 3.2):

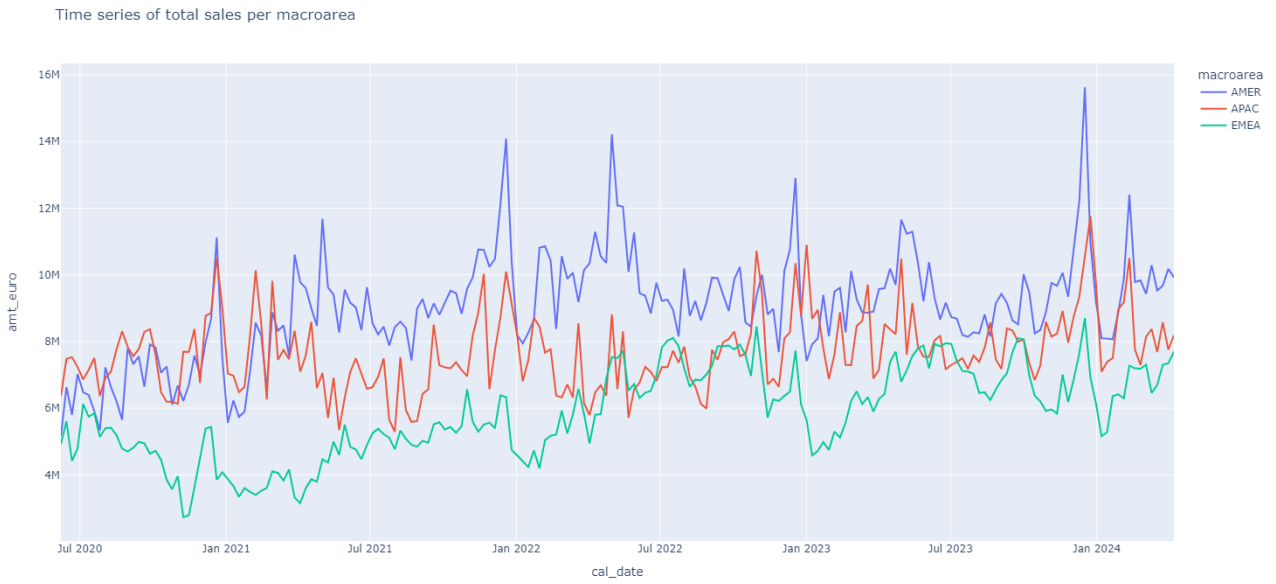


*Figure 3.2*

The trend of each macro-area is not as clear as before, and the same can be said for the seasonality. Disaggregating even further one of them, the APAC for example, Figure 3.3 is obtained:

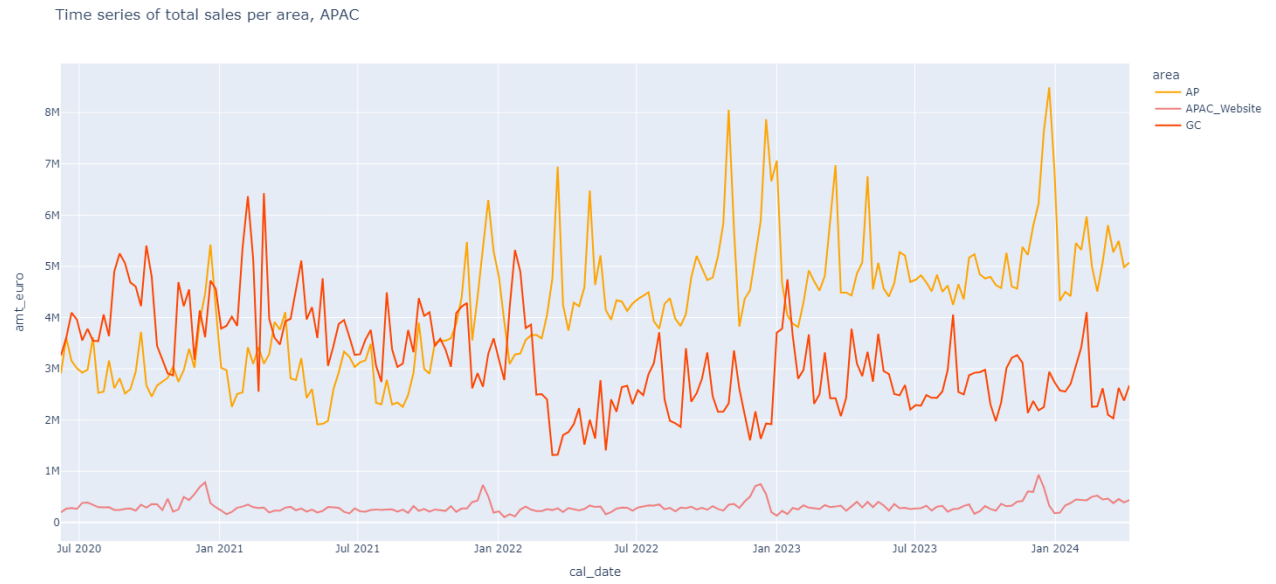


*Figure 3.3*

Here one of the series, the GC one, even seems to have a decreasing trend. This is further proof that each level should be forecasted independently.

To fit all the models, all the series have been split into a training and a test set: since the client is interested in the forecasts of a year of the sales, the test set is made up of 52 weeks, and consequently the train set has 149 observations.

To evaluate the models, three indicators to evaluate on the test predictions were chosen, so from time $T = 149$ to time $T + h = 149 + 52 = 201$. The first one is the Mean Absolute Error, in Equation 3.1

$$MAE = \frac{1}{h}\sum_{t=T}^{T+h}\left|Y_{i,t} - \hat{Y}_{i,t}\right|$$

*Equation 3.1*

For the series at level $i$, this metric gives information on how far the predictions are from the real values on an absolute scale. This might be useful to confront series on the same aggregation level (i.e. the products), but not so much when comparing different levels or countries that might have completely different magnitudes.

The second measure is the Mean Absolute Percentage Error (Equation 3.2):

$$MAPE = \frac{1}{h}\sum_{t=T}^{T+h}\left|\frac{Y_{i,t} - \hat{Y}_{i,t}}{Y_{i,t}}\right| * 100$$

*Equation 3.2*

Being a percentage error, it is better suited for comparing the results of the different countries, but there is the risk that it explodes when some of the actual values of the series are low. This is true for a lot of series in the dataset, as it has a lot of products with intermittent sales.

The last one is the Weighted Absolute Percentage Error (Equation 3.3):

$$WAPE = \frac{\sum_{t=T}^{T+h}|Y_{i,t} - \hat{Y}_{i,t}|}{\sum_{t=T}^{T+h}|Y_{i,t}|}$$

*Equation 3.3*

This metric is more solid to intermittent sales and to low actual values, so it will be used more as it is more suited for the data at hand.

Besides these metrics, for all the models it was also computed the training time and the number of parameters needed for them to work to have a complete overview of the comparison.

## 3.1 MODELS BUILDING BLOCKS

Before showing how all the proposed models work, an excursus on the various statistical and machine learning notions used in this thesis is needed.

### 3.1.1 MULTIPLE LINEAR REGRESSION

Since the client asked for the usage of the external variables, the first thing that comes to mind when dealing with regressors $x_i$ is the multiple linear regression model (Hyndman & Athanasopoulos, 2021).

$$Y_t = \beta_0 + \beta_1 x_{1,t} + \cdots + \beta_{k,t} x_{k,t} + \varepsilon_t$$

*Equation 3.4*

Each coefficient $\beta_i$ in Equation 3.4 measures the effect of each predictor $x_{i,t}$ on the response variable $Y_t$ after taking into account the effects of all the other predictors. This is a measure of the marginal effects of the predictor variables.

For this model to work, there are some assumptions on the error terms $\varepsilon_t$:

- They have mean zero, or the forecast would be biased.
- They are not autocorrelated, or the model would be inefficient and there is information in the data that can still be used.
- They are unrelated to the predictor variables.

This is just a matter of computing some estimates of the coefficients $\beta_i$, which are signed as $\hat{\beta}_i$, and using them alongside what we think the external variables will be to make forecasts. Considering the task this is possible, since it is an ex-ante forecast and the future values of the exogenous variables are computed by the company.

A good way to make this estimate is the least squares principle, which computes $\beta_i$ by minimizing the sum of the squared errors in Equation 3.5:

$$\sum_{t=1}^{T} \varepsilon_t^2 = \sum_{t=1}^{T} \left(Y_t - \beta_0 - \beta_1 x_{1,t} - \beta_2 x_{2,t} - \cdots - \beta_{k,t}\right)$$

*Equation 3.5*

This is called least squares estimation because it gives the least value for the sum of the squared values. Once the estimates are computed, the forecasts $\hat{Y}_t$ can simply be forecasted as shown in Equation 3.6:

$$\hat{Y}_t = \hat{\beta}_0 + \hat{\beta}_1 x_{1,t} + \cdots + \hat{\beta}_{k,t} x_{k,t}$$

*Equation 3.6*

These models allow for the inclusion of several relevant information given by the regressor, but they do not allow for the subtle time series dynamics to be modeled in any way.

### 3.1.2 ARIMA

When working with time series, another model widely used is the autoregressive integrated moving average model, ARIMA for short, and its seasonal counterpart, SARIMA. These models are capable of describing the behaviour of the series using its past values (Marcilio, et al., 2013).

To write down the actual formula of a SARIMA model we can introduce the backward shift operator $B$ (Hyndman & Athanasopoulos, 2021), which is a useful notational device:

$$BY_t = Y_{t-1}$$
$$B(BY_t) = B^2Y_t = Y_{t-2}$$

*Equation 3.7*

SARIMA models are usually represented as $(p, d, q)(P, D, Q)_s$.

The letter $p$ refers to the autoregressive part of the model, so the one which uses the past $p$ values of the series to compute the next ones:

$$\begin{aligned} Y_t &= c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \varepsilon_t \\ &= c + \phi_1 B Y_t + \phi_2 B^2 Y_t + \cdots + \phi_p B^p Y_t + \varepsilon_t \end{aligned}$$

*Equation 3.8*

The $d$ is the order of differencing, which is a data transformation which is used to make the series stationary, that consists in calculating the differences among pairs of observations at a $d$ lag. For a $d = 1$ differencing we have:

$$Y'_t = Y_t - Y_{t-1} = Y_t - BY_t = (1 - B)Y_t$$

*Equation 3.9*

The last letter, $q$, is the moving average part, which uses as regressors the previous errors $\varepsilon$ between the model and the actual values to adjust the predictions.

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

*Equation 3.10*

The capital letters are the seasonal counterparts of the others, and $s$ is the seasonal lag. Using the backward shift operator, a SARIMA $(1,1,1)(1,1,1)_S$ model is:

$$(1 - \phi_1 B)(1 - \Phi_1 B^S)(1 - B)(1 - B^S)Y_t = (1 - \theta_1 B)(1 - \Theta_1 B^S)\varepsilon_t$$

*Equation 3.11*

As one can see, ARIMA and SARIMA models are only capable of fitting linear patterns in the data in an autoregressive fashion (Panigrahi & Behera, 2017), which might be limiting in some scenarios.

### 3.1.3 ETS

Similar to the ARIMA there is a whole other family of autoregressive models, the ETS, which is an exponential smoothing model.

Exponential smoothing techniques use a weighted average of the past observation to compute a forecast, with the later observation having a higher impact. These models can be classified in 30 different ways, based on how they handle the trend and the seaonality components of the data, and their error components. To distinguish them, the triplet $(E, T, S)$ is used, which stands for error, trend and seasonality components. The general model for all of the models involves a state vector

$x_t = (l_t, b_t, s_t, s_{t-1}, \dots, s_{t-m+1})$ and the state space equations (Hyndman & Khandakar, 2008) have the following form:

$$y_t = w(x_{t-1}) + r(x_{t-1})\varepsilon_t$$
$$x_t = f(x_{t-1}) + g(x_{t-1})\varepsilon_t$$

*Equation 3.12*

Where $\{\varepsilon_t\}$ is a Gaussian white noise process with mean 0 and variance $\sigma^2$ and $\mu_t = w(x_{t-1})$.

ETS can have both linear and nonlinear models. The linear ETS are actually special cases of the ARIMA family, but the nonlinear ones have no ARIMA counterpart, making them suitable for capturing nonlinear behaviors.

### 3.1.4 TBATS

One more exponential smoothing model is the TBATS (De Livera et al., 2011), which stands for Trigonometric exponential smoothing state space model with Box-Cox transformation, ARMA errors, Trend and Seasonal components. It allows for multiple, complex, and dynamic seasonalities in the TS (Karabiber & Xydis, 2019).

TBATS considers various alternatives and fits different models for then choosing the best one using the AIC:

- Models with and without Box-Cox transformation, which can transform non-normal response variables into normal, by trying different $\lambda$ values from -5 to 5 in the Equation 3.13:

$$y(\lambda) = \begin{cases} \dfrac{y^\lambda - 1}{\lambda} \; if \; \lambda \neq 0 \\ \log(y) \; if \; \lambda = 0 \end{cases}$$

*Equation 3.13*

- With and without trend and trend damping, the latter being the addition of a parameter that forces the trend, which increases or decreases indefinitely, to a flat line after a certain amount of time (Hyndman & Athanasopoulos, 2021).
- With and without the ARMA errors, with ARMA simply being an ARIMA$(p, 0, q)$.
- Non seasonal model.
- With harmonics used in seasonal models.

One assumption this model has is that the error distribution fits the Gaussian distribution.

### 3.1.5 ANN

One other method which is widely used in the field of TS forecasting are ANNs, the Artificial Neural Networks (Medina Maçaira, et al., 2018).

The greatest strength of the ANNs is their capability of approximating a large class of non-linear problems to any desired level of accuracy. In particular, the single hidden layers multilayer perceptron (MLP) is used widely in the field of TS forecasting (Panigrahi & Behera, 2017). A generic

multilayer perceptron is just a mathematical function that maps a set of inputs into an output and is formed by composing many simpler functions (Goodfellow, et al., 2016). The single hidden layer ones consist of an input layer, a single hidden layer, and an output layer with adjacent interlayer nodes connected by acyclic links.

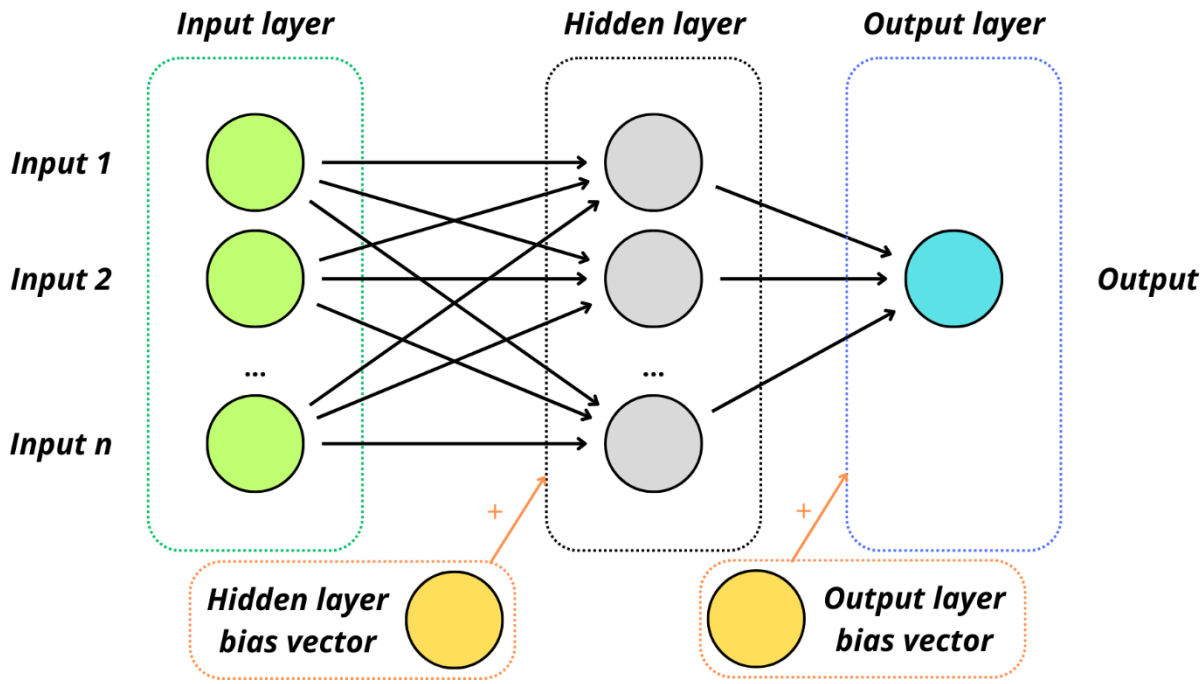


*Figure 3.4*

In Figure 3.4 is represented a simple MLP with $n$ inputs, $n$ hidden units in the only hidden layer, a single output and a bias vector for each layer beside the input. Each black arrow is a multiplication by some coefficient, so calling $X \in \mathbb{R}^n$ the input vector, $h \in \mathbb{R}^n$ the hidden layer, $B_h$ the bias vector of the hidden layer and $A \in \mathbb{R}^{n \times n}$ the matrix with all the coefficients, we get:

$$h = act(AX + B_h)$$

*Equation 3.14*

Where $act(x)$ is some activation function applied element-wise to the layer neurons. The most common activation functions for the hidden layers are the Rectified Linear Unit (RELU) and its slightly modified version the leaky RELU, the sigmoid, and the hyperbolic tangent. The main role of the activation function is to make the resulting output nonlinear.

These simple feedforward networks try to capture a nonlinear relationship between the inputs and the target output. To train these models to fit the data, the most common approach used is the gradient descent:

- The network parameters are initialized to small random values.
- A loss function that uses the network coefficients is chosen, usually the mean squared error:

$$MSE = \frac{1}{m}\sum_i \left(\hat{Y} - Y\right)_i^2$$

*Equation 3.15*

Here $m$ is the size of the of the input set at hand, $Y$ the target values of the network, $\hat{Y}$ the output predicted by the network.

- The goal is to minimize the loss function: to achieve an optimal minimum, the gradient of the function is computed, and the coefficients are moved in small steps in the opposite direction. The size of the steps is defined by the learning rate, an additional hyper parameter that must be set beforehand.

This part of the process is called training of the network, and uses the observation on the train set to evaluate the network coefficients.

## 3.2 IMPLEMENTED FORECASTING MODELS

The models which have been used are some of the most renowned ones which employ external variables (Medina Maçaira, et al., 2018) or a combination of them. All the results and error measures will be shown in chapters 3.3, 4.2, and 6.1

It needs to be pointed out that all the predictions of the models must be non-negative: some series, especially the ones being really close to zero, risk to be forecasted to negative values. To avoid this, an ex-post fix is done turning all of them to 0 before computing the metrics.

### 3.2.1 SARIMAX

The biggest issue with the regression model is that it completely ignores the autocorrelation that might be present in the time series. This is where the Seasonal Auto Regressive Integrated Moving Average (SARIMA) model steps in.

By combining the SARIMA with the linear regression, we get the SARIMAX (where the X stands for the external variables), which is practically a linear model with SARIMA errors $\eta$. An example with a SARIMA $(1,1,1)(1,1,1)_S$ errors is shown in Equation 3.16:

$$Y_t = \beta_0 + \beta_1 x_{1,t} + \cdots + \beta_k x_{k,t} + \eta_t$$

$$(1 - \phi_1 B)(1 - \Phi_1 B^S)(1 - B)(1 - B^S)\eta_t = (1 - \theta_1 B)(1 - \Theta_1 B^S)e_t$$

*Equation 3.16*

Where $e_t$ is white noise.

To handle these models in Python, the pmdarima (Smith & al., 2017) and the sklearn (Pedregosa & al., 2011) libraries were used; as for the approach the following was used for each country and each series:

1) Estimate the parameters of the linear model using all the non-constant-zero explanatory variables related to it, using the sklearn LinearRegression, on the training set.
2) Using the Akaike's Information Criterion (AIC), do a backward elimination of the non-statistically significant variables, by using the SequentialFeatureSelector function of sklearn.

$$AIC = T \log\left(\frac{SSE}{T}\right) + 2(k+2)$$

*Equation 3.17*

In Equation 3.17 is shown the AIC formula: $T$ is the length of the time series, $SSE$ is the sum of squared estimate of errors and $k$ is the number of variables in the model.
The lower the AIC, the better is the model. What SequentialFeatureSelector does is, starting from the full model created in step 1, eliminating the variable with the greatest $p$-value, then the model is re-fitted and the AIC re-evaluated again. If there is a significant improvement in the AIC, the variable is definitely removed from the model and the process in repeated with another variable; else the regressor is reinserted. The model fitted with the remaining variables is the best one for linear regression.

3) Using the estimated coefficients of the best linear model and the train set exogenous variables compute the training fit.
4) Compute the residuals by subtracting the best model training fit from the actual values of the training set.
5) Fit on the residuals a SARIMA model using the pmdarima auto_arima: this model, using the AIC, finds the best option for the $(p, d, q)(P, D, Q)_{52}$ parameters.
6) Using the test set external variables, compute the forecasts of the next year using the test variables, obtaining the test fit for the linear model.
7) Compute the 52 steps ahead forecast of the SARIMA model found and sumit to the test linear fit: this is the test SARIMAX prediction.
8) Compute on the test forecasts the MAE, MAPE and WAPE for both the linear fit and the SARIMAX fit.

The program implemented (Code 6.2), in a local machine, employed 3 hours, 24 minutes and 46 second to fit all the models. This model's results are noted as ARIMAX in all the tables that will follow.

### 3.2.2 HYBRID NEURAL NETWORK

The next model proposed is the hybrid NN approach, which combines exponential smoothing techniques with the ANN structure. Panigrahi & Behera (2017) proposed a hybrid model that combined ETS and MLP for univariate time series forecasting.

Panigrahi & Behera explain that other authors' works on hybrid statistical and NN models all work well when the time series is composed of a linear and a nonlinear pattern. The problem is that in real world scenarios, a TS may be purely linear or nonlinear, or contain a moltitude of linear and nonlinear patterns. For this reason, unlike the previous work which all hybridized with ARIMA models, the authors use ETS models alongside the ANN. The idea is that while ARIMA only captures linear patterns, ETS can capture both linear and nonlinear ones. The proposed algorithm is rather simple:

$$Y_t = C_t^1 + C_t^2$$

$$e_t = Y_t - \hat{C}_t^1$$

$$\hat{Y}_t = \hat{C}_t^1 + \hat{C}_t^2$$

*Equation 3.18*

The true values of the series $Y_t$ are a sum of two components $C_t$, which might be either linear or nonlinear. The ETS model is fitted on the training data to get the first components of the series, $\hat{C}_t^1$. The residuals $e_t$ of the ETS model are computed and are then used to fit an ANN which will in turn return $\hat{C}_t^2$. The main reasons to divide this way the algorithm are:

1) ETS contains both linear and nonlinear models.
2) Even though ANN can handle linear patterns, it can not do so as well as it can handle nonlinear behaviors.
3) If the TS has both linear and nonlinear patterns, it is more likely that the ETS will capture the linear part, returning some nonlinear residuals that are handled well by the ANN.

In the time series field, the inputs for MLP are usually the last $k$ observations of the time series $y_{t-1}, \dots, y_{t-k}$ and the output is the single step ahead forecast $y_t$. Panigrahi's work focuses on univariate time series forecasting, while the problem at hand is a multivariate one, thanks to the external regressors. For this reason, the algorithm is adapted in this way:

1) For each series in the HTS, an ETS model is fitted on the training data.
2) The residuals of each series training set are computed by subtracting from the original data the ETS fit.
3) The residuals are fed to an ANN alongside the external variables, and it is trained to return the whole vector of one-step ahead residuals.

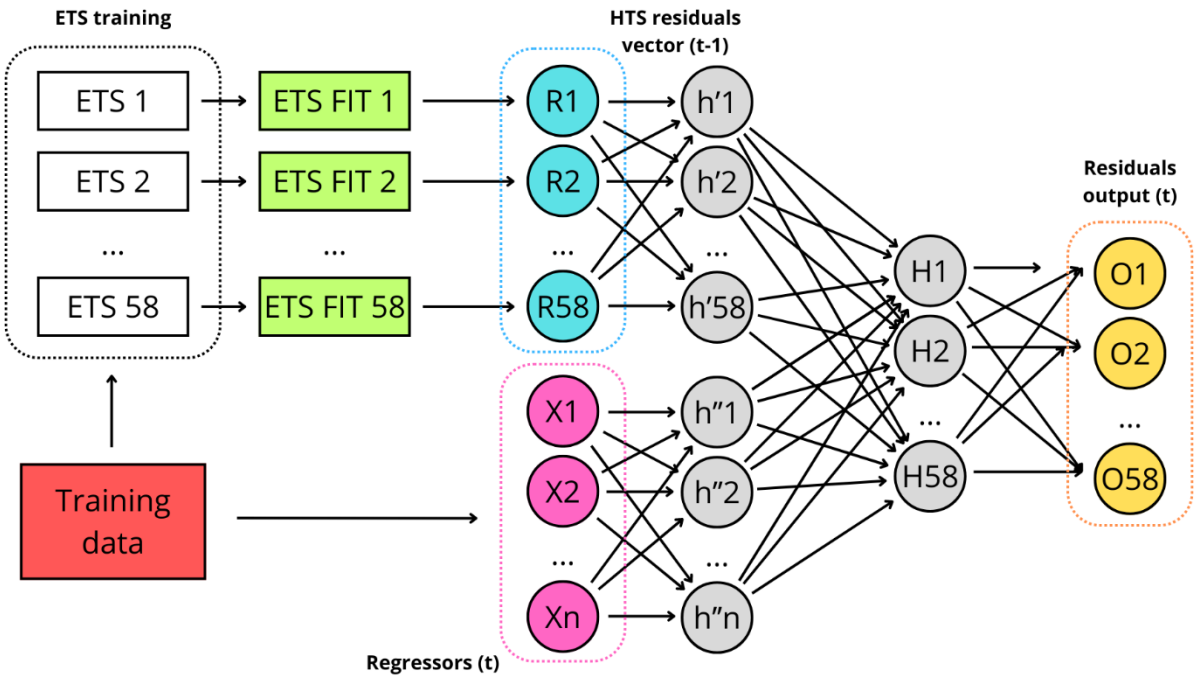


*Figure 3.5*

The architecture shown in Figure 3.5 uses just one past observation to predict the next one. This implementation has the benefit to incorporate the external variables in a nonlinear fashion and it

does not require to fit a whole NN for each level of the HTS for every country, reducing the computational burden. Another highlight is that it models the influence of different levels of the series on the others.

Before fitting the model on the training set, its necessary to handle some hyperparameters that might undermine the network performance if chosen poorly. The ones studied are the number of epochs needed for the training, the activation function used in the different layers and the number of previous residuals that should be fed in the network. A grid search is done over these parameters: the epochs could either be 50, 100 or 200, the activations could be the sigmoid, leaky RELU or tanh, and finally the number of previous timestamps could be 1, 2 or 3.

To do this, a 4-fold cross-validation method using a moving window is used (Hyndman & Athanasopoulos, 2021): the training set is further divided into multiple training and validation sets. Both the sets, in each fold of the validation, change as one can see in Figure 3.6, according to a moving window.

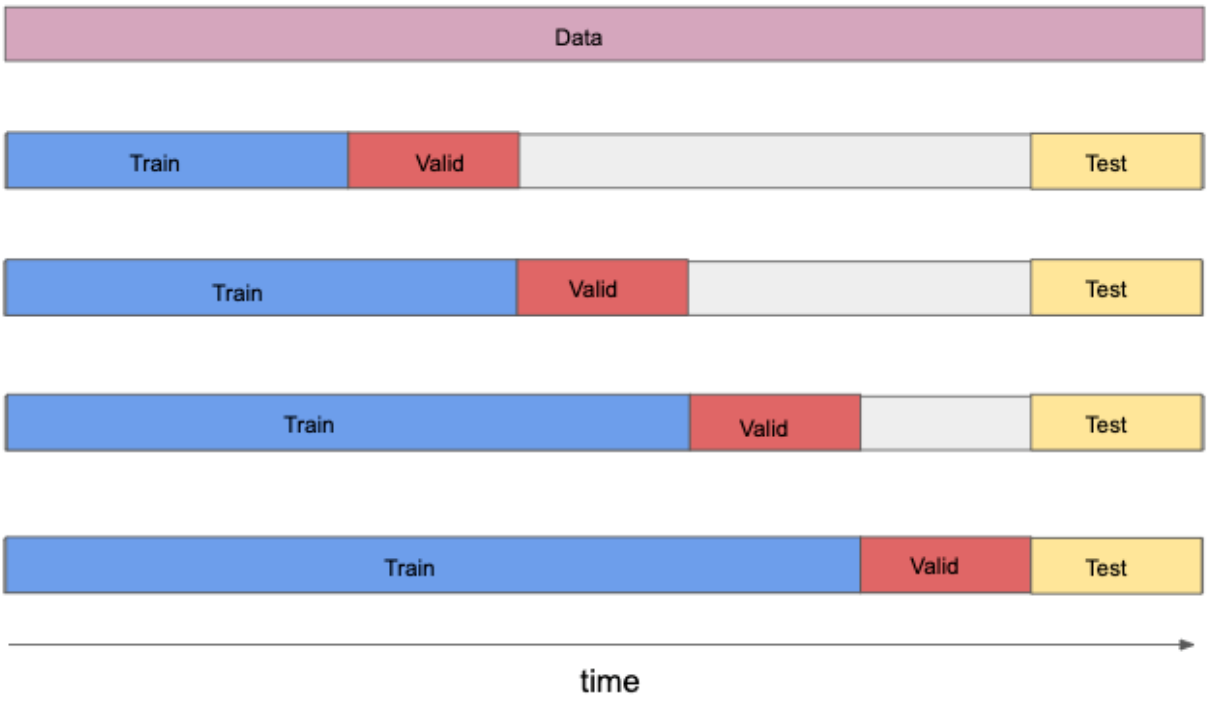


*Figure 3.6*

The parameters needed for this procedure are:

- The starting window, so the number of observations in the training set of the first fold of the validation.
- Ending window, which is the number of data points included in the last training iteration.
- Forecasting window, that is the number of data points included for the forecasting.
- Expanding step, which is the number of data points added to the training at each new fold.

For this procedure the starting window was set to 105, the ending window to 138, the forecasting window to 11 as well as the expanding step. Considering how long is the time series, this is the smallest amount possible of observations needed for the cross-validation: we need at least 100

observations to estimate the ETS properly with its seasonal components, and the remaining observations need to be divided. 11 is actually a suboptimal number of observations for the forecast, since it ignores the task at hand of forecasting an entire year of sales. Another issue is that the best approach would be to perform the cross-validation for each country and each level of the series but given the time constraint of the job this was not possible, so it was done only on the top 3 selling ones (US, CN, JP) and their results were averaged. The metrics used to compare the results of the grid search was the Mean Squared Error (MSE) computed on the validation set:

$$MSE = \frac{1}{q}\sum_{i=n+1}^{n+q}\left(\hat{Y}_t - Y_t\right)^2$$

*Equation 3.19*

In Equation 3.19 $q$ is the forecasting window length, in this case 11, $n$ is the number of data points used in training the model, which changes at each validation, $\hat{Y}_t$ and $Y_t$ are the forecasted vector and the real values of the series vector respectively.

When training the NN, both in the validation process and in the final version of it, the residuals which were fed into it were scaled between 0 and 1. The main reason why this was done is because of the way the MSE is computed: if the normal scales of the data were kept, the first component of the $Y_t$ vector, which is the aggregated total of the series, would have a magnitude which is much higher than all the other components, and this would also be true for its residuals. This would lead to a validation process which would just end up returning a Neural Network that models properly only the errors for the total aggregated series. By scaling, all the series levels have the same range of values, and the lowest MSE found in the cross-validation is the one of the NN that minimizes the residuals for all the series levels. The simple scaling formula is shown in Equation 3.20:

$$R_{k,T}^{(sc)} = \frac{R_{k,T} - \min\left(R_{k,t}\right)_{t=1}^{n}}{\max\left(R_{k,t}\right)_{t=1}^{n} - \min\left(R_{k,t}\right)_{t=1}^{n}}$$

*Equation 3.20*

Here the specific value of the series $k$ at time $T$, $R_{k,T}$, is scaled using the maximum and the minimum of the whole series $R_{k,t}$ used for the training part. Once each series training set residuals are scaled, they are all fed to the NN for the training part except for the last $l$, where $l$ is the number of previous observations used for the forecast. After that, since this network is autoregressive, for the validation set forecasts the first prediction is computed by using the last $l$ unused values, which are then fed to the network in a recurrent fashion to get all the needed predictions. Figure 3.7 shows an example for $l = 3$.

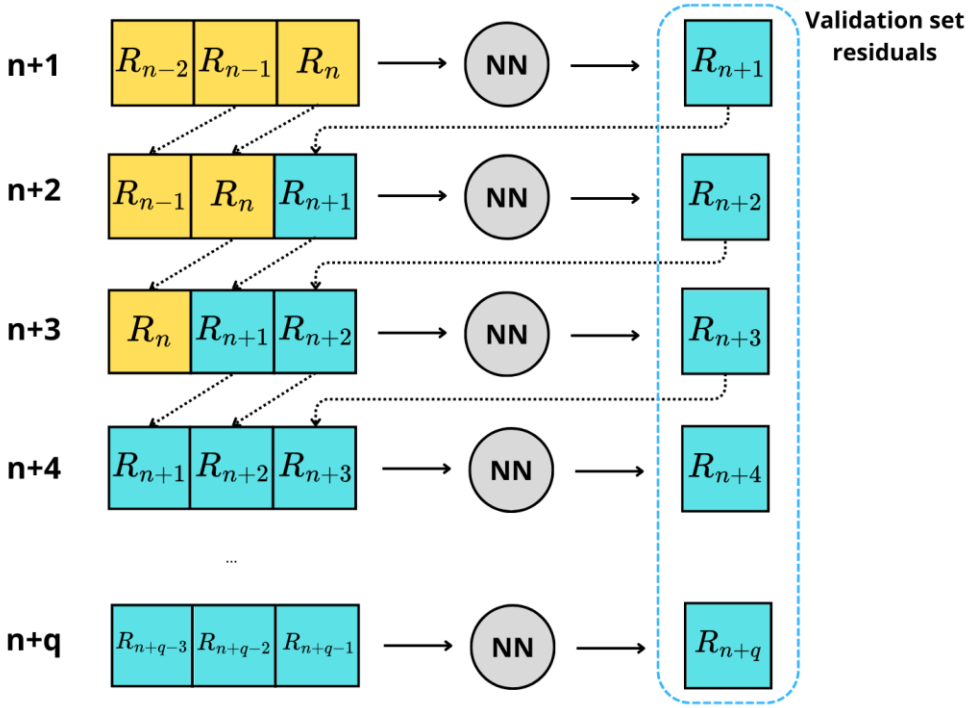


*Figure 3.7*

Once the validation process was finished, the results were averaged for the various folds and countries, and then sorted by MSE.

| Epoch | Act | Previous observations | MSE |
|---|---|---|---|
| 50 | sigmoid | 1 | 2.63 |
| 100 | sigmoid | 1 | 2.647 |
| 50 | tanh | 1 | 2.65 |
| 50 | sigmoid | 2 | 2.655 |
| 100 | sigmoid | 2 | 2.656 |
| 200 | sigmoid | 1 | 2.659 |
| 100 | tanh | 1 | 2.662 |
| 200 | tanh | 1 | 2.664 |
| 50 | sigmoid | 3 | 2.675 |
| 200 | leaky_relu | 3 | 2.704 |
| 100 | sigmoid | 3 | 2.708 |
| 200 | sigmoid | 2 | 2.718 |
| 200 | tanh | 3 | 2.721 |
| 200 | sigmoid | 3 | 2.734 |
| 100 | leaky_relu | 3 | 2.735 |
| 100 | tanh | 3 | 2.736 |
| 50 | leaky_relu | 3 | 2.736 |
| 50 | tanh | 3 | 2.75 |
| 50 | leaky_relu | 2 | 2.752 |
| 50 | tanh | 2 | 2.76 |
| 50 | leaky_relu | 1 | 2.771 |

| | | | |
|---|---|---|---|
| 200 | leaky_relu | 2 | 2.772 |
| 100 | tanh | 2 | 2.79 |
| 100 | leaky_relu | 2 | 2.799 |
| 200 | tanh | 2 | 2.821 |
| 100 | leaky_relu | 1 | 2.843 |
| 200 | leaky_relu | 1 | 3.095 |

*Table 3.1*

In Table 3.1 one can see the results: it seems like a higher number of training epochs, or more than one previous observation are not necessary for improving the model performance, while the best MSE are obtained by using the sigmoid as the various hidden layers activation function. In the end, the final residuals Neural Networks were trained using 50 epochs, the sigmoid activation function, and one previous observation.

### 3.2.3 TBATS-NN HYBRID

While the ETS proves to be a great model, it is rather simple in the available exponential smoothing available models.

The TBATS are also exponential smoothing technique, but they are more complex and consider many different scenarios, so their results could be an improvement over the ETS model. A nouvelle model was then implemented: the TBATS-NN hybrid.

The architecture in basically the same as the ETS hybrid model shown in Figure 3.5:

1) For each series in the HTS, a TBATS model is fitted on the training data.
2) The residuals of each series training set are computed by subtracting from the original data the TBATS fit.
3) The residuals are fed to an ANN alongside the external variables, and it is trained to return the whole vector of one-step ahead residuals.

As far as it concerns the NN hyperparameters, since the TBATS is a more sophisticated version than the ETS, no other cross validation procedure was done to choose the best NN parameters, and the same architectures of the ETS residuals NN were used.

The code for both methods is shown in Code 6.3 for the ETS fit, Code 6.4 for the TBATS fit and finally in Code 6.5 for the residuals modelled with the NN.

### 3.2.4 ENSEMBLE

In literature, it has been noted that a simple way to improve forecasts is to combine the results of different models (Timmermann, 2006). The simplest way to do so is with the simple average, where the results of the different forecasting models are simply averaged over the number of models.

To try this out, two ensembles were proposed: one with ARIMAX and the TBATS hybrid NN models, and one between ARIMAX, TBATS hybrid NN and ETS hybrid NN models.

## 3.3 THE MODELS RESULTS

When considering which model to choose, while the performance of the model itself is quintessential, there are other features that one must take into account. The other two that were considered are the time used for creating the model, since as highlighted when showing the pipeline, the models are built on the fly when requested by the client, and the number of parameters that need to be stored to use it.

Let us now see how each of the six models proposed performed in each of these features.

### 3.3.1 PERFORMANCE

As previously mentioned, the models were fitted on all the levels of the series for each country. The results can be confronted based on the series level in the hierarchy, or by country.

The first results shown are the ones using the different series levels. To show the computed metrics heatmaps have been used, one for each metric and one for each level of the hierarchy.

Since looking at all the metrics for each level and for each country would be too long and difficult, a weighted mean was used to compute an average metric

$$\widehat{WAPE} = \sum_{i \in country} p_i \cdot WAPE_i$$

*Equation 3.21*

$$p_i = \frac{\sum_{t=1}^{T} Y_{tot,t}^{i}}{\sum_{i \in country} \sum_{t=1}^{T} Y_{tot,t}^{i}}$$

*Equation 3.22*

In Equation 3.21 is shown how to compute the weighted mean of the WAPE, but the formula is identical for all the other metrics. As shown in Equation 3.22, the proportions are computed using the sum of the total sales for each country, which are divided for the total sales for all the countries, over a period of time $T$: in the following results as $T$ is the training time length, 149, was used.

Starting from the bottom level (the product and clientele type split) the maps in Figure 6.1, Figure 6.2, and Figure 6.3 of the appendix are computed.

The first issue is the MAPE: because of the intermittent sales of the products, the metric skyrockets to values which are way too high. This is a recurring problem in the next analysis of the bottom and middle levels, so it will be ignored in those levels. Another issue is that some values, especially in WAPE, are so high that the map does not return a clear indication on how the models are performing when compared to each. By applying a natural logarithm to both the WAPE and MAE's results they are more understandable.

By comparing the maps in Figure 3.8 and Figure 3.9 one can see that the series with the highest WAPE are also the series with the lowest MAE. This implies that they are the least selling series, which means that in an absolute total scale of all the sales these should not pose a big problem.

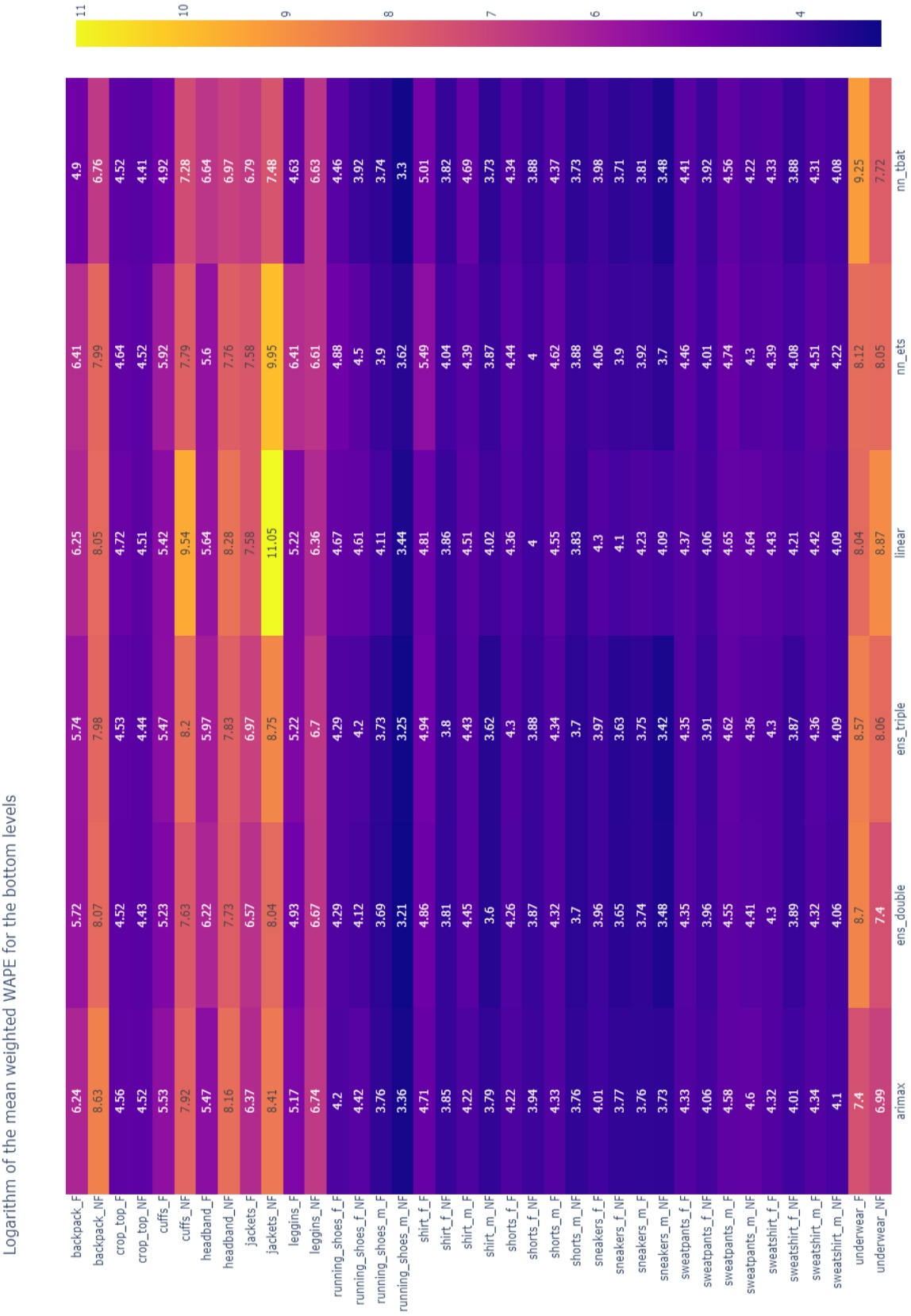


*Figure 3.8*

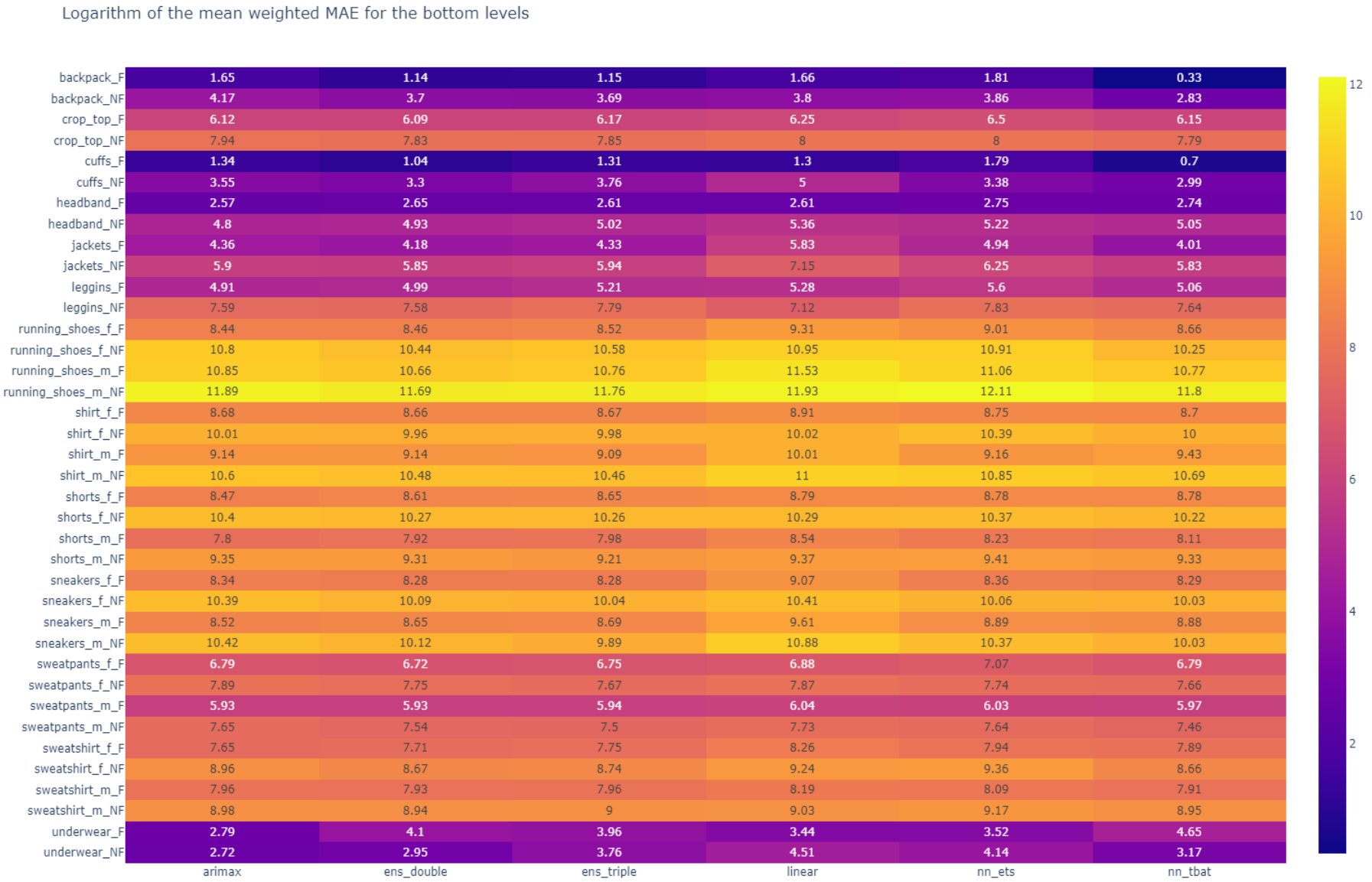
Logarithm of the mean weighted MAE for the bottom levels
backpack_F 1.65 1.14 1.15 1.66 1.81 0.33
backpack_NF 4.17 3.7 3.69 3.8 3.86 2.83
crop_top_F 6.12 6.09 6.17 6.25 6.5 6.15
crop_top_NF 7.94 7.83 7.85 8 8 7.79
cuffs_F 1.34 1.04 1.31 1.3 1.79 0.7
cuffs_NF 3.55 3.3 3.76 5 3.38 2.99
headband_F 2.57 2.65 2.61 2.61 2.75 2.74
headband_NF 4.8 4.93 5.02 5.36 5.22 5.05
jackets_F 4.36 4.18 4.33 5.83 4.94 4.01
jackets_NF 5.9 5.85 5.94 7.15 6.25 5.83
leggins_F 4.91 4.99 5.21 5.28 5.6 5.06
leggins_NF 7.59 7.58 7.79 7.12 7.83 7.64
running_shoes_f_F 8.44 8.46 8.52 9.31 9.01 8.66
running_shoes_f_NF 10.8 10.44 10.58 10.95 10.91 10.25
running_shoes_m_F 10.85 10.66 10.76 11.53 11.06 10.77
running_shoes_m_NF 11.89 11.69 11.76 11.93 12.11 11.8
shirt_f_F 8.68 8.66 8.67 8.91 8.75 8.7
shirt_f_NF 10.01 9.96 9.98 10.02 10.39 10
shirt_m_F 9.14 9.14 9.09 10.01 9.16 9.43
shirt_m_NF 10.6 10.48 10.46 11 10.85 10.69
shorts_f_F 8.47 8.61 8.65 8.79 8.78 8.78
shorts_f_NF 10.4 10.27 10.26 10.29 10.37 10.22
shorts_m_F 7.8 7.92 7.98 8.54 8.23 8.11
shorts_m_NF 9.35 9.31 9.21 9.37 9.41 9.33
sneakers_f_F 8.34 8.28 8.28 9.07 8.36 8.29
sneakers_f_NF 10.39 10.09 10.04 10.41 10.06 10.03
sneakers_m_F 8.52 8.65 8.69 9.61 8.89 8.88
sneakers_m_NF 10.42 10.12 9.89 10.88 10.37 10.03
sweatpants_f_F 6.79 6.72 6.75 6.88 7.07 6.79
sweatpants_f_NF 7.89 7.75 7.67 7.87 7.74 7.66
sweatpants_m_F 5.93 5.93 5.94 6.04 6.03 5.97
sweatpants_m_NF 7.65 7.54 7.5 7.73 7.64 7.46
sweatshirt_f_F 7.65 7.71 7.75 8.26 7.94 7.89
sweatshirt_f_NF 8.96 8.67 8.74 9.24 9.36 8.66
sweatshirt_m_F 7.96 7.93 7.96 8.19 8.09 7.91
sweatshirt_m_NF 8.98 8.94 9 9.03 9.17 8.95
underwear_F 2.79 4.1 3.96 3.44 3.52 4.65
underwear_NF 2.72 2.95 3.76 4.51 4.14 3.17
arimax
ens_double
ens_triple
linear
nn_ets
nn_tbat
12
10
8
6
4
2

*Figure 3.9*

Another deduction that can be made is that all the models have similar performances, in the sense that if a series is problematic for a model, it is (maybe with a different magnitude) for the others too.

The next performance is checked on the middle level, so the products split, of the series. It's clear that the same scale issue that the bottom level maps is in the new ones in Figure 6.4 and Figure 6.5, so the results are once again scaled with the natural logarithm.

The conclusions that were deducted from the bottom levels are also confirmed by the maps in Figure 6.6 and Figure 6.7: if a series returns bad forecasts, it does so for all the models, and the highest WAPE is related to the lowest MAE. One more thing can be said: the results are slightly better overall in this aggregation level, which can be seen from the scales the metrics have. It looks like the higher the aggregation the better the models handle the series.

The next results should be the mean metrics for all the countries on the total aggregation level. Since those results could be shown with just a single row, the next maps that are shown are the total ones divided by country. By looking at these WAPE values in Figure 6.8 of the appendix, one can see that the errors are not as high as with the other disaggregation levels, probably because in the total series there are no intermittent sales or strange fluctuations. There still is a problem with how the heatmap seems to have just a few spikes in Figure 6.9, so the logarithms of it is shown in Figure 6.10. It seems that the worst models are the linear and the hybrid ETS-NN, it also gives a clear idea of what are the most challenging countries for the models, which are some outlets, Arabian Emirates (AE), Cile (CL), and Saudi Arabia (SA), and which prediction algorithms struggle the most.

To assess the general performance of all the levels for all the countries, the total aggregation level is computed in three ways: by looking directly at the forecast of the total, by aggregating the middle-level forecasts, or by aggregating the bottom-level forecasts. These are essentially the results of a bottom-up approach for reconciliation and will return a mean behavior of the performance on the lower-level predictions. Once the metrics were computed for each of these levels, a weighted average was computed based on the data in the training set, to have a synthetic metric describing the overall performance.

| | ARIMAX | Ensemble double | Ensemble triple | Linear | ETS + NN | TBATS + NN |
|---|---|---|---|---|---|---|
| bot_agg | 20.04 | 17.16 | 18.55 | 25.49 | 26.6 | 18.25 |
| mid_agg | 18.4 | 17.12 | 18.5 | 25.61 | 26.65 | 19.1 |
| tot | 20.19 | 17.42 | 18.85 | 25.2 | 28.59 | 19.12 |

*Table 3.2: weighted models WAPE*

| | ARIMAX | Ensemble double | Ensemble triple | Linear | ETS + NN | TBATS + NN |
|---|---|---|---|---|---|---|
| bot_agg | 267812 | 220259 | 241290 | 403411 | 355759 | 240800 |
| mid_agg | 250918 | 240948 | 255465 | 408512 | 357581 | 283736 |
| tot | 284239 | 242786 | 259086 | 396235 | 388453 | 263630 |

*Table 3.3: weighted models MAE*

| | ARIMAX | Ensemble double | Ensemble triple | Linear | ETS + NN | TBATS + NN |
|---|---|---|---|---|---|---|
| bot_agg | 20.39 | 17.55 | 19.24 | 25.97 | 27.71 | 18.58 |
| mid_agg | 18.71 | 17.28 | 18.97 | 26.16 | 27.71 | 19.14 |
| tot | 20.18 | 17.33 | 19.09 | 25.5 | 29.86 | 19.1 |

*Table 3.4: weighted models MAPE*

Overall, the best models in terms of minimizing the error are the ARIMAX, the TBATS-NN hybrid and their ensemble.

### 3.3.2 TIME PERFORMANCE

The time performance is another key aspect in the analysis: a good model can be hindered by a long training time, as it would make it unfeasible to use in most real-life scenarios.

| Model | Single country fit time | Whole dataset fit time |
|---|---|---|
| ARIMAX | 6m 7s | 3h 20m 40s |
| ETS | 16s | 18m 52s |
| + NN | + 9s | + 3m 56s |
| TBATS | 15m 21s | 10h 47m* |
| + NN | + 10s | + 3m 52s |

*Table 3.5*

In Table 3.5 are displayed the time it took every model to be trained, both for a single country and for the whole dataset, on-premise. The * mark in the TBATS whole dataset fit time indicates that that training time was the only one that did not use any kind of parallelization: as soon as parallelizing by the country was tried, the model took way longer to train, and it returned the fit for just a couple of levels per country. It is not clear what caused the problem, it probably has to do with what the TBATS Python library does when its methods are called, but it seems like it is a known issue (Nadeem, 2021).

One can also notice that the parallelized version of the ETS method does not return particular benefits, on the contrary, it seems to worsen the time performance. This is most likely a problem with the on-premise workplace: the available cores to use were 7, but it needs to be taken into

account that the 7 process to be parallelized need to be prepared and then started, which can be quite computationally demanding.

Nonetheless, the single country fit time returns a general idea on how the model would work in the real-life scenario, especially since the need for the forecast for all the countries at once would be quite rare. Looking at the results the fastest algorithm is the ETS-NN hybrid model, followed by the ARIMAX and ending with the TBATS-NN hybrid.

### 3.3.3 SPACE PERFORMANCE

The number of parameters a model requires is another characteristic of the model that needs to be considered. Besides the obvious memory issue, one could have if the parameters are too many, using them requires more computations, which leads to higher energy costs to keep the machines running. If, hypothetically, the cloud on which the algorithm is running has also a cost based on the number of computations, this is another cost that would be added.

Considering all the models that were used:

- Linear: it just needs a parameter for each exogenous variable +1 for the intercept. In the worst-case scenario where all the variables are non-zero and they all are significant, the parameters would be 32 + 1, so 33.
- ARIMAX: this model has a linear fit as its base, so the starting point for all the models in the most pessimistic scenario is again 33. Building from this, the SARIMA fitted on the residuals were forced to have at most 1 for $p, d$ and $q$ and their seasonal counterparts. All of them count as a parameter, so if all of them are significant the model would have 33 + 6 parameters, so 39.
- ETS: this smoothing technique simply has one parameter to estimate for the level, the slope and the seasonal component of the series, so at most 3.
- TBATS: for the base models this is the most complex one. The Box-Cox transformation needs a parameter, and so does the dampening part. The seasonal cycle is composed of a number of harmonics, but it's difficult to assess a priori how many will be there, as there is no way to set boundaries on the ARMA errors. While this might seem problematic, this number would for sure become non-significant when compared to the NN part.
- NN: 12444 parameters are trained for the neural network to work.

The main takeaway is that both the methods that implement the Neural Network, and by extension both the ensembles that use the, have a number of parameters which is a lot bigger than all the other models. In the end, the best one in terms of memory occupied by the parameters is the linear, followed by the ARIMAX and then by the NN hybrids.

## 3.4 THE BEST MODEL

Looking at just the performances, if time and memory allow it, the best model would be the ensemble between the ARIMAX and the TBATS-NN hybrid models. If there were any problems with

the large training times or the number of parameters to store and then use to compute, the more suitable model would then be just the ARIMAX. Both the ETS-NN hybrid and the linear models are too weak in terms of performance to justify using them, even if they are quick to train and, for the second one, have really little memory requirements for the parameters.

In the next chapter, the reconciliation part will be tackled. The forecasts that will reconciled are the ARIMAX, the TBATS-NN hybrid, and their ensemble and all the results will be confronted.

# 4 RECONCILIATION

As already mentioned in the first chapter, the main property of HTS is the linear constraints they are subject to, so now that all the forecasts have been computed, it is necessary to reconcile them in order to have properly reconciled outputs.

## 4.1 THE METHODS

In chapter 1.3 some possible ways to reconcile the data were explored. In this chapter, the focus will be on the bottom-up approach, the minimization of the trace and the disaggregation NN. Both the top-down and, by extension, the middle-out approach won't be covered, even if their results could be easily computed with the code at hand. This decision was made because the top-down method results are biased, unlike all the other methods.

### 4.1.1 BOTTOM-UP AND MINIMUM TRACE

To implement the first two methods, the Python library HierachicalForecast was used (Olivares et al., 2022).

Once given the proper training set for the disaggregation, the method computes the reconciliation matrix using the formula in Equation 1.17 and Equation 1.18 for the covariance matrix. It also re-aggregates the bottom-level predictions to compute the bottom-up approach reconciliation.

As the training set for computing the reconciliation matrix the real data of the training set was given, the matrix would then be used to reconcile the models forecasts. Once the reconciled fit is returned, MAE, WAPE and MAPE are computed for each level and forecast (Code 6.6).

### 4.1.2 NND

The other method used implements a simplified version of the NN used by Mancuso et al. (2021) for each of the country. The Network uses as the input fixed segments of the series on the highest level of the hierarchy which are passed to a three-layers Convolution Neural Network and the external variables which are fed to a three-layers MLP. The results of the two parts are then concatenated and used by a single hidden layer MLP to return the lowest level of the HTS.

The segments were of length 4 to approximately have the data of one-month sales, so from time $T-3$ to time $T$, and they are used by the network alongside the exogenous variables at time $T$ to get the values of the lowest levels at time $T$. The segments had an overlap of 75% between each other.

Using the same validation technique illustrated in chapter 3.2.2 some parameters of the Neural Network were tuned using a grid search. The parameters tested were:

- The alpha in the special loss function in Equation 1.20, between 0.25, 0.5 and 0.75.

- The number of filters for the convolutional layers, between 16, 32, 64.
- The kernel size of the convolutional layers, between 4, 8 and 16. Notice that the segments are initially of length 4: to avoid problems with any of these kernel sizes, the first kernel size is always 2.
- The number of hidden units in the perceptron layers between 32, 64, 128.

A 4-fold moving-window cross-validation was performed on the three most selling countries (CN, JP, and US) and using the same parameters for the window size and the hop as in chapter 3.2.2: the starting window was set to 105, the ending window to 138, the forecasting window to 11 as well as the expanding step. The metric used to choose the best model was the MAPE on the aggregated total computed summing the parts of the disaggregation. The final results are in Table 4.1, after averaging the metrics by all the validation folds and all the countries.

| Alpha | Filters | Kernel size | Hidden units | MAPE total |
|---|---|---|---|---|
| 0.25 | 64 | 8 | 128 | 0.2215 |
| 0.25 | 32 | 16 | 128 | 0.231 |
| 0.25 | 64 | 8 | 32 | 0.2408 |
| 0.25 | 32 | 8 | 32 | 0.2447 |
| 0.25 | 64 | 4 | 128 | 0.2452 |
| 0.25 | 16 | 8 | 32 | 0.2458 |
| 0.25 | 64 | 16 | 32 | 0.2463 |
| 0.5 | 16 | 16 | 64 | 0.2463 |
| 0.5 | 32 | 16 | 64 | 0.2481 |
| 0.25 | 32 | 8 | 64 | 0.2488 |
| 0.5 | 64 | 8 | 32 | 0.2507 |
| 0.75 | 32 | 16 | 32 | 0.2522 |
| 0.75 | 16 | 8 | 32 | 0.2531 |
| 0.25 | 64 | 16 | 64 | 0.2542 |
| 0.25 | 64 | 8 | 64 | 0.2548 |
| 0.75 | 32 | 8 | 64 | 0.2557 |
| 0.5 | 32 | 16 | 32 | 0.2565 |
| 0.25 | 16 | 16 | 64 | 0.2568 |
| 0.75 | 16 | 16 | 32 | 0.2581 |
| 0.5 | 32 | 4 | 32 | 0.2585 |
| 0.25 | 32 | 4 | 32 | 0.2586 |
| 0.75 | 16 | 8 | 64 | 0.2591 |
| 0.25 | 16 | 16 | 128 | 0.2593 |
| 0.75 | 32 | 16 | 64 | 0.2594 |
| 0.25 | 16 | 4 | 64 | 0.2602 |
| 0.25 | 16 | 8 | 64 | 0.2603 |
| 0.25 | 32 | 16 | 32 | 0.2605 |
| 0.75 | 16 | 16 | 64 | 0.2606 |
| 0.75 | 64 | 8 | 64 | 0.2609 |
| 0.5 | 64 | 16 | 64 | 0.2612 |
| 0.25 | 16 | 4 | 32 | 0.2613 |

| | | | | |
|---|---|---|---|---|
| 0.5 | 32 | 8 | 128 | 0.2619 |
| 0.25 | 32 | 4 | 64 | 0.2624 |
| 0.5 | 64 | 4 | 128 | 0.2646 |
| 0.25 | 16 | 16 | 32 | 0.2657 |
| 0.75 | 64 | 4 | 32 | 0.2665 |
| 0.25 | 16 | 4 | 128 | 0.2671 |
| 0.5 | 64 | 4 | 32 | 0.2675 |
| 0.5 | 16 | 4 | 32 | 0.2678 |
| 0.5 | 16 | 16 | 32 | 0.2702 |
| 0.5 | 64 | 8 | 128 | 0.2708 |
| 0.25 | 16 | 8 | 128 | 0.2712 |
| 0.5 | 16 | 8 | 128 | 0.272 |
| 0.75 | 16 | 4 | 64 | 0.2724 |
| 0.5 | 32 | 16 | 128 | 0.2726 |
| 0.75 | 16 | 4 | 32 | 0.2738 |
| 0.5 | 32 | 4 | 128 | 0.2739 |
| 0.5 | 16 | 4 | 64 | 0.2744 |
| 0.5 | 16 | 8 | 32 | 0.2745 |
| 0.75 | 32 | 4 | 128 | 0.2748 |
| 0.5 | 64 | 16 | 128 | 0.2753 |
| 0.5 | 16 | 16 | 128 | 0.2757 |
| 0.75 | 32 | 4 | 64 | 0.2761 |
| 0.25 | 64 | 4 | 64 | 0.2767 |
| 0.25 | 32 | 4 | 128 | 0.2767 |
| 0.75 | 64 | 8 | 32 | 0.2771 |
| 0.25 | 64 | 16 | 128 | 0.2777 |
| 0.75 | 64 | 4 | 64 | 0.2789 |
| 0.25 | 64 | 4 | 32 | 0.2789 |
| 0.75 | 32 | 8 | 32 | 0.2795 |
| 0.25 | 32 | 16 | 64 | 0.2798 |
| 0.75 | 64 | 16 | 64 | 0.2802 |
| 0.5 | 16 | 4 | 128 | 0.2815 |
| 0.25 | 32 | 8 | 128 | 0.2817 |
| 0.5 | 32 | 4 | 64 | 0.2821 |
| 0.5 | 32 | 8 | 64 | 0.2821 |
| 0.75 | 16 | 4 | 128 | 0.2822 |
| 0.75 | 64 | 16 | 128 | 0.2826 |
| 0.75 | 64 | 16 | 32 | 0.2836 |
| 0.75 | 64 | 8 | 128 | 0.2841 |
| 0.75 | 16 | 16 | 128 | 0.2846 |
| 0.75 | 16 | 8 | 128 | 0.2857 |
| 0.5 | 16 | 8 | 64 | 0.2877 |
| 0.5 | 64 | 16 | 32 | 0.2877 |
| 0.75 | 32 | 8 | 128 | 0.2882 |
| 0.5 | 64 | 4 | 64 | 0.2886 |
| 0.75 | 32 | 4 | 32 | 0.2889 |

| | | | | |
|---|---|---|---|---|
| 0.5 | 32 | 8 | 32 | 0.2889 |
| 0.75 | 64 | 4 | 128 | 0.2894 |
| 0.5 | 64 | 8 | 64 | 0.2906 |
| 0.75 | 32 | 16 | 128 | 0.3036 |

*Table 4.1*

Even if it is not the best result, the chosen final model was the one with the alpha of 0.25, 32 filters, kernel size of 8, and hidden layers of size 32. This was done to keep the Network as small as possible, and it still is capable of giving one of the best results.

Once the parameters were set, the network for each country was computed using as training data the real observations on the first 149 periods, and the disaggregation on the test set of the remaining 52 time steps was computed using as input the forecast of the total of all the models which one is interested in.

For each model, the MAE, WAPE, and MAPE are then computed for all the levels. Code 6.7 shows the implementation in Python: notice that it has been called DNN (Disaggregating Neural Network) instead of NDD, and that this notation will be kept in all the measures results heatmaps.

## 4.2 THE RECONCILIATION RESULTS

After reconciliation, the same analyses done for the models' performance in chapter 3.3 are repeated.

### 4.2.1 PERFORMANCE

The first results shown are the weighted average of the WAPE and MAE for the different models and reconciliations on the bottom level of the hierarchy. Of course, the bottom-up approach has the same results that were previously shown because it uses at the bottom-level forecasts the ones computed by the model.

Once again, the maps in Figure 6.11 and Figure 6.12 in the appendix give little visual information because of some high values: the most problematic one seems to be the DNN approach forecast of the running_shoes_m_NF, which have a really high MAE, and the underwear_F for the bottom-up approach for the WAPE.

The logarithm returns better visual results to see how the various models are performing overall compared to each other.

Looking at Figure 4.1 and Figure 6.13 one can see that the NND seems to give really good results on the bottom level of the series, especially in the series related to products with low sales. On the other hand, the bottom-up and the MinT have a slightly better performance on the higher-selling products.

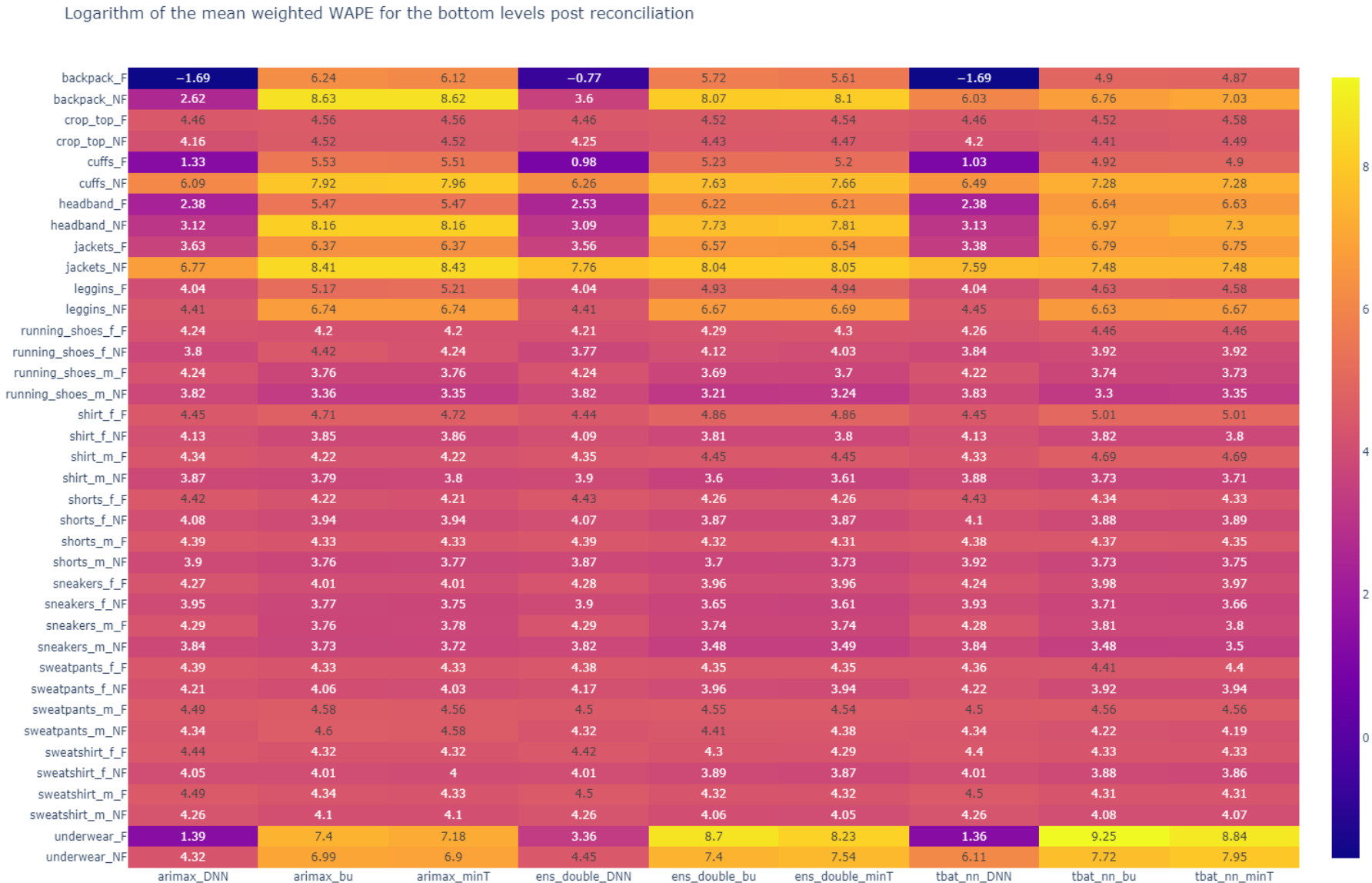
Logarithm of the mean weighted WAPE for the bottom levels post reconciliation
backpack_F −1.69 6.24 6.12 −0.77 5.72 5.61 −1.69 4.9 4.87
backpack_NF 2.62 8.63 8.62 3.6 8.07 8.1 6.03 6.76 7.03
crop_top_F 4.46 4.56 4.56 4.46 4.52 4.54 4.46 4.52 4.58
crop_top_NF 4.16 4.52 4.52 4.25 4.43 4.47 4.2 4.41 4.49
cuffs_F 1.33 5.53 5.51 0.98 5.23 5.2 1.03 4.92 4.9
cuffs_NF 6.09 7.92 7.96 6.26 7.63 7.66 6.49 7.28 7.28
headband_F 2.38 5.47 5.47 2.53 6.22 6.21 2.38 6.64 6.63
headband_NF 3.12 8.16 8.16 3.09 7.73 7.81 3.13 6.97 7.3
jackets_F 3.63 6.37 6.37 3.56 6.57 6.54 3.38 6.79 6.75
jackets_NF 6.77 8.41 8.43 7.76 8.04 8.05 7.59 7.48 7.48
leggins_F 4.04 5.17 5.21 4.04 4.93 4.94 4.04 4.63 4.58
leggins_NF 4.41 6.74 6.74 4.41 6.67 6.69 4.45 6.63 6.67
running_shoes_f_F 4.24 4.2 4.2 4.21 4.29 4.3 4.26 4.46 4.46
running_shoes_f_NF 3.8 4.42 4.24 3.77 4.12 4.03 3.84 3.92 3.92
running_shoes_m_F 4.24 3.76 3.76 4.24 3.69 3.7 4.22 3.74 3.73
running_shoes_m_NF 3.82 3.36 3.35 3.82 3.21 3.24 3.83 3.3 3.35
shirt_f_F 4.45 4.71 4.72 4.44 4.86 4.86 4.45 5.01 5.01
shirt_f_NF 4.13 3.85 3.86 4.09 3.81 3.8 4.13 3.82 3.8
shirt_m_F 4.34 4.22 4.22 4.35 4.45 4.45 4.33 4.69 4.69
shirt_m_NF 3.87 3.79 3.8 3.9 3.6 3.61 3.88 3.73 3.71
shorts_f_F 4.42 4.22 4.21 4.43 4.26 4.26 4.43 4.34 4.33
shorts_f_NF 4.08 3.94 3.94 4.07 3.87 3.87 4.1 3.88 3.89
shorts_m_F 4.39 4.33 4.33 4.39 4.32 4.31 4.38 4.37 4.35
shorts_m_NF 3.9 3.76 3.77 3.87 3.7 3.73 3.92 3.73 3.75
sneakers_f_F 4.27 4.01 4.01 4.28 3.96 3.96 4.24 3.98 3.97
sneakers_f_NF 3.95 3.77 3.75 3.9 3.65 3.61 3.93 3.71 3.66
sneakers_m_F 4.29 3.76 3.78 4.29 3.74 3.74 4.28 3.81 3.8
sneakers_m_NF 3.84 3.73 3.72 3.82 3.48 3.49 3.84 3.48 3.5
sweatpants_f_F 4.39 4.33 4.33 4.38 4.35 4.35 4.36 4.41 4.4
sweatpants_f_NF 4.21 4.06 4.03 4.17 3.96 3.94 4.22 3.92 3.94
sweatpants_m_F 4.49 4.58 4.56 4.5 4.55 4.54 4.5 4.56 4.56
sweatpants_m_NF 4.34 4.6 4.58 4.32 4.41 4.38 4.34 4.22 4.19
sweatshirt_f_F 4.44 4.32 4.32 4.42 4.3 4.29 4.4 4.33 4.33
sweatshirt_f_NF 4.05 4.01 4 4.01 3.89 3.87 4.01 3.88 3.86
sweatshirt_m_F 4.49 4.34 4.33 4.5 4.32 4.32 4.5 4.31 4.31
sweatshirt_m_NF 4.26 4.1 4.1 4.26 4.06 4.05 4.26 4.08 4.07
underwear_F 1.39 7.4 7.18 3.36 8.7 8.23 1.36 9.25 8.84
underwear_NF 4.32 6.99 6.9 4.45 7.4 7.54 6.11 7.72 7.95
arimax_DNN
arimax_bu
arimax_minT
ens_double_DNN
ens_double_bu
ens_double_minT
tbat_nn_DNN
tbat_nn_bu
tbat_nn_minT
8
6
4
2
0

*Figure 4.1*

The next performance assessment is done on the middle level of the series. The results in Figure 6.14 and Figure 6.15 have the usual scale issue, so using again the logarithm to have a better understanding of how the model differ we get Figure 6.16 and Figure 6.17. The results are similar to the ones on the bottom level: the NND captures well the products sales for the merchandise with low profit, while the bottom-up and the MinT have a slight edge on the most selling ones.

Analyzing how the models behave on the total level there is no need to average the metrics, and one can directly see the performance on the different countries.

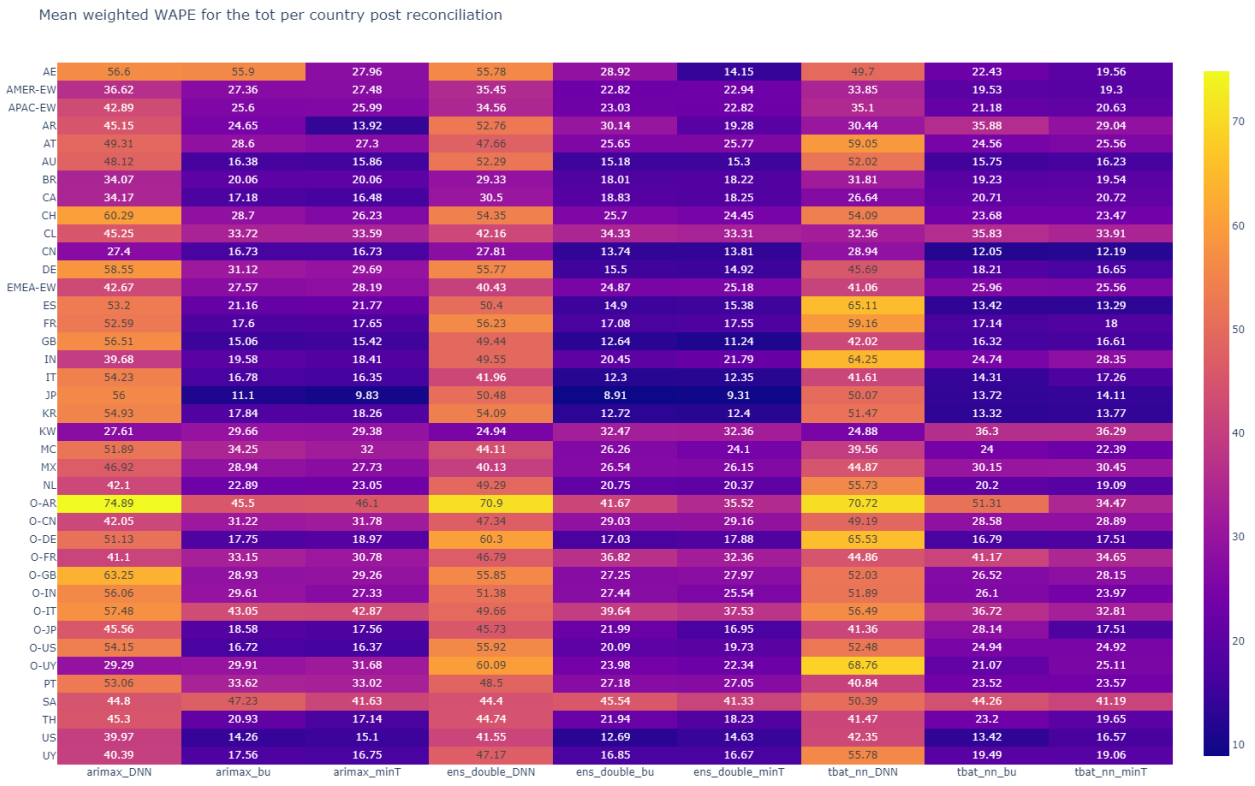


*Figure 4.2*

In Figure 4.2 it is clear that once all the levels are aggregated, the performance of the NND is the worst: its slightly worse performance on the top-selling products leads to a high error in the total sales. At first glance, one can also notice that the minT approach seems a little better than the bottom-up one.

Some countries seem more troublesome than others, such as some outlets (AR, IT, and CN in particular) or SA, as all the models struggle to have a low WAPE on them.

Since Figure 6.18 has the scale issue, the logarithm will be shown only for the MAE.

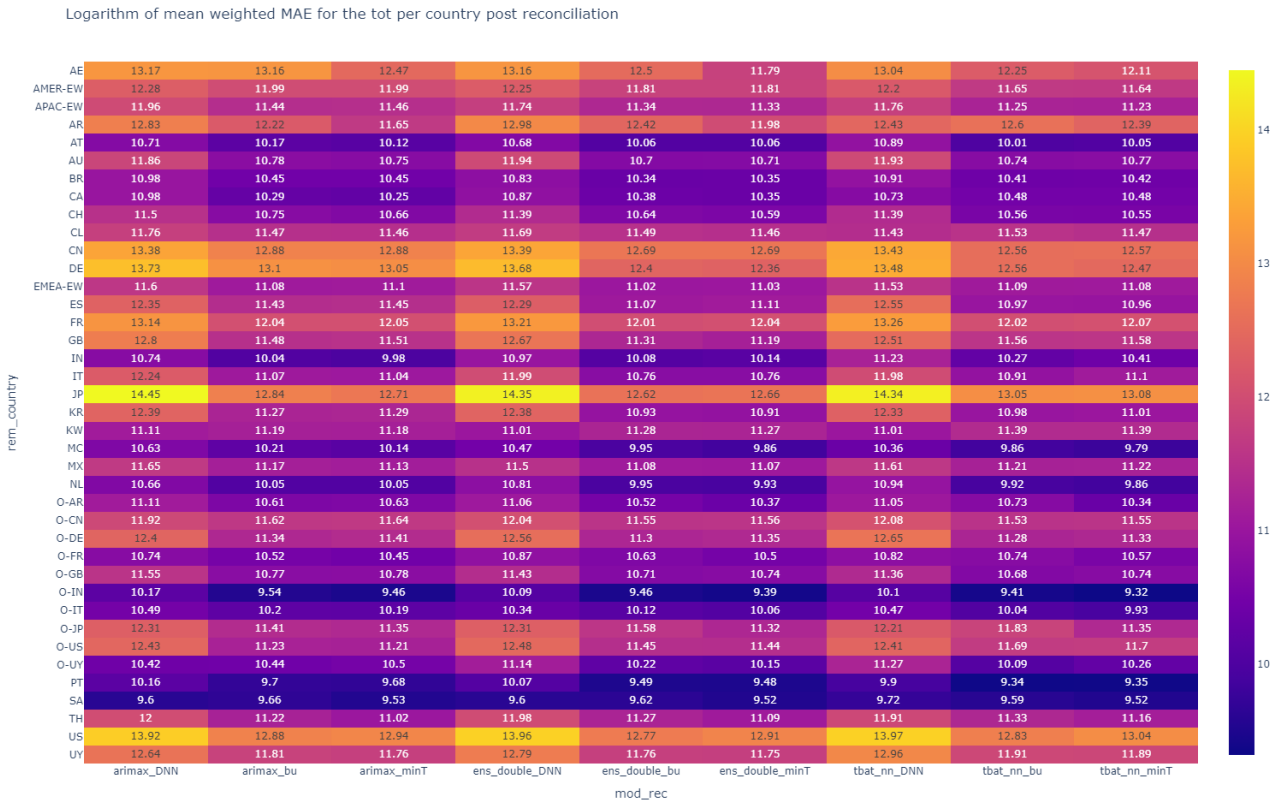


*Figure 4.3*

In Figure 4.3 one can see that the NND is indeed the worst reconciliation model when looking at the total sales for all the countries. The most problematic countries highlighted by the WAPE do not seem to have the highest MAE, so they should not pose a big problem overall.

By using again the weighted average of Equation 3.21 a more synthetic measure can be computed to confront all the models and all the reconciliation methods.

| | ARIMAX | Ensemble | TBATS NN |
|---|---|---|---|
| NND | 44.16 | 44.23 | 43.49 |
| BU | 20.04 | 17.16 | 18.25 |
| minT | 18.66 | 16.54 | 18.38 |

*Table 4.2: WAPE per model post reconciliation*

| | ARIMAX | Ensemble | TBATS NN |
|---|---|---|---|
| NND | 42.4 | 42.48 | 41.86 |
| BU | 20.39 | 17.55 | 18.58 |
| minT | 19.01 | 16.82 | 18.52 |

*Table 4.3: MAPE per model post reconciliation*

| | ARIMAX | Ensemble | TBATS NN |
|---|---|---|---|
| NND | 703104.0 | 696915.0 | 693768.0 |
| BU | 267812.0 | 220259.0 | 240800.0 |
| minT | 255475.0 | 225431.0 | 257884.0 |

*Table 4.4: MAE per model post reconciliation*

In Table 4.2, Table 4.3 and Table 4.4 the results are clear: the best model in terms of accuracy is ensemble of the other two reconciled using the minimization of the trace.

### 4.2.2 TIME PERFORMANCE

As far as it concerns the time performance, the implementation in Python of the bottom-up and minT approaches was quick, with the reconciliation of a country taking just 10 seconds and for the whole dataset 6 minutes and 14 seconds. On the other hand, the NND takes time to train, for instance, it takes almost 1 minute for a single country and around 40 minutes for the whole dataset. It's clear that on the time aspect the bottom-up and minimum trace approaches are better.

### 4.2.3 SPACE PERFORMANCE

Looking at the number of parameters needed for the bottom-up methods, there are none: the only thing needed is the summing matrix Equation 1.5 which, multiplied by the bottom levels obtained from the models, returns all the levels of the HTS.

The minimum trace method needs, besides the summing matrix, the positive definite covariance matrix (Equation 1.17), which has a $52 \times 52$ matrix of parameters to store.

Finally, the NND network has, with the structure chosen from the cross-validation, 31908 parameters needed for the whole computation.

When considering the space performance, one needs to keep in mind that the bottom-up method requires the bottom-level of the series, the MinT needs all the levels while the DNN only needs the top one, so the actual number of parameters may vary since the number of forecasting algorithms needed for each are really different.

## 4.3 THE BEST COMBINATION

Looking at all the performances and trying to balance them out, the best model that could be chosen is the ARIMAX reconciled with the minT method. It gives some of the best results, without requiring building really complex neural networks that, if used, would just improve slightly the results at the cost of longer times and bigger computational and storage load.

Another option could be using the bottom-up approach for the reconciliation, as its performance, on average, matches the minT ones without requiring the forecasts for all the levels of the series.

One method that does not return good results in this scenario is the DNN: it is the one that takes the most time to fit, and the one with the highest number of parameters needed, while its results are far worse than the others. This performance issue might be caused by the size of the dataset, which is pretty small for a NN implementation, and the lack of specific external variables that can help distinguish between different products' behavior.

If one were to choose the model just on the performance, so not caring about the computational load and the time it takes for the model to be used, then the double ensemble with the minT reconciliation is of course the best one.

The last step is to confront the performance with the previous NN that was implemented by the company, to check if there is an actual improvement of the forecasts. Both the ARIMAX and the ensemble are used in the confrontation, to both the model with the best overall performance and the one with the best forecasts, and also the TBATS-NN since it is the core of this thesis. All the results shown use the minT reconciliation method, as it is the most consistent over all of them

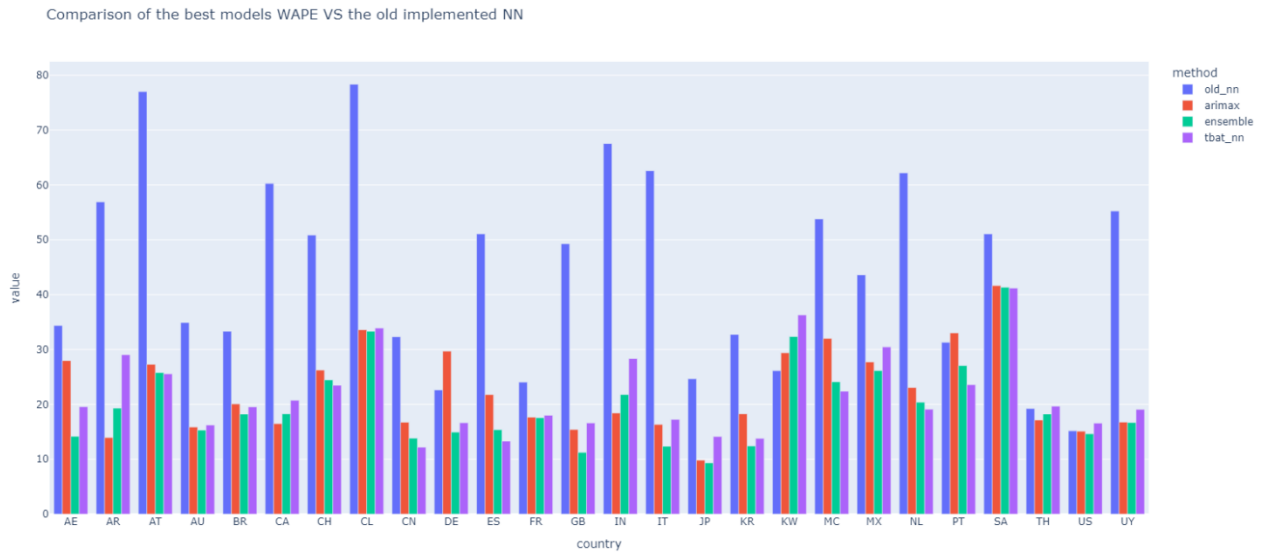


*Figure 4.4*

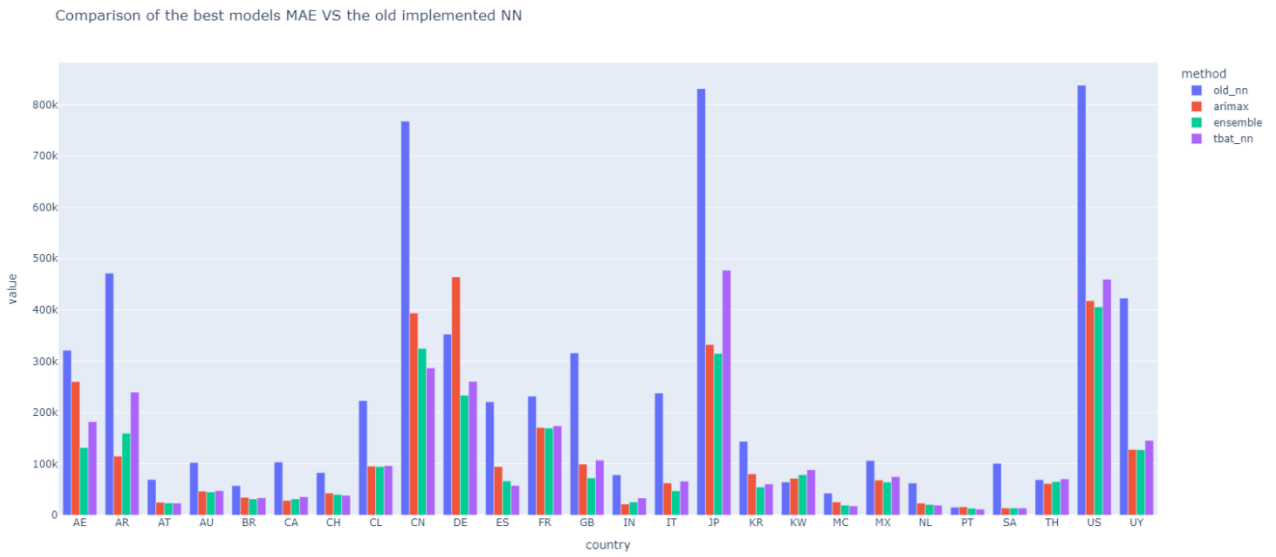


*Figure 4.5*

Not all the countries were used for the old results, for example, there are no outlets, so the comparison is done only on the available ones.

In Figure 4.4 and Figure 4.5 one can see that all the model hit the brief and are an improvement over the old NN that was used, with the only exception being the ARIMAX model in DE and all of them in KW.

# 5 CONCLUSIONS

The scope of the thesis was showcasing how to perform, in Python, an analysis of hierarchical data and how to implement some forecasting methods and reconcile the results properly, while also trying a new approach: the TBATS-NN hybrid. Once all of this was done, the results were confronted to asses which model and reconciliation technique were the best.

## 5.1 PROCEDURE RECAP

A client of a counseling company asked for the improvement of an already assessed forecasting algorithm for hierarchical series data regarding the company's sales which could implement some external variables that they were capable of predicting themselves.

The initial step was cleaning the data. The first thing that was done was checking the data itself: the starting point was a grouped time series with 8 disaggregating factors. By checking with the client, the disaggregation levels were reduced to just 4: country, product, clientele, and markdown.

After a quick look at the overall behavior of all the different countries' sales, it was decided that the sales data related to the COVID pandemic should be removed since it is highly unlikely that an event like that could repeat itself anytime soon.

The next part was checking that all the countries for which we had data had enough data points for implementing an actual algorithm. By removing the shorter ones that did not have enough observations, the 62 starting countries were reduced to 39.

A similar procedure was done for the products: out of the 25 that were there at the start, by eliminating the ones which had extremely localized sales spikes and the ones which only sold in the last year, the number was reduced to 19.

By analysing how the sales split using the clientele type and the markdown, it was discovered that the markdown split was a legacy of an old classification system the client had. Now, all the products which were discounted and sold in a country would be simply addressed as the sales of the country's outlet. With this, another level of disaggregation was directly removed from the analysis.

Looking at the external variables that were given, the ones COVID-related were discarded since the pandemic period was removed from the data and all of their information seemed already incorporated into other variables such as the number of days the shops were open.

Once the data was properly cleaned, the next step was implementing various forecasting models which implemented external variables. The multi-linear model was the simplest and the first implemented: it just fits the data using external variables as predictors.

Working with a time series the linear model can be lacking, since it does not consider the autocorrelation in the data. To improve it, the ARIMAX model was also used: after fitting the linear model, the residuals were fitted in an autoregressive linear fashion with an SARIMA.

The ETS model was used next. While this model does not use external variables, it is able to fit the data in an autoregressive fashion to find the trend and seasonal components of the data. To improve it was decided to model its residuals with an ANN. The main benefit when compared to the ARIMA residuals is that the neural network can fit non-linear relationships.

Trying to upgrade and improve on the ETS-NN hybrid, a new approach was tried. Instead of fitting the data with an ETS, a more advanced exponential smoothing technique was used: the TBATS. The idea is that a more powerful forecasting tool combined with the capability of the NN to capture non-linear patterns should result in better predictions for the series.

A couple of ensembles, one between the ARIMAX and the TBATS-NN, and one using all the models except for the linear, were also implemented. The ensembles were kept as simple as possible with just the arithmetic mean being used to combine the results.

The models were fitted on all the countries and on all their levels, and the predictions for each of them for one year were computed. The WAPE, MAPE and MAE for all the models at each level were computed, and the weighted average of them (using the countries sales in the training set as proportions) were calculated for each level of the hierarchy. The results were compared to choose the best models out of them.

What was left to do with the best models was reconcile their results. The first reconciliation technique used was the bottom-up approach: by simply using the bottom-level forecasts, the results were re-aggregated into all the higher levels. Both the top-down and middle-out approach were not used as they return biased predictions.

The optimal reconciliation technique known as minimization of the trace (minT) was implemented next: it requires the forecasts for all the levels of the series, and reconciles while trying to minimize the trace of the variance matrix of the errors after the reconciliation.

The last method was the Neural Network Disaggregation (NND). This technique is similar to the top-down approach in a way since it starts from the top-level forecasts which are fed to a NN trained using a special loss function and returns the bottom level forecasts that can then be reaggregated to all the top levels. One of the main perks of this network is that it can also use external variables to improve the split.

After the reconciliation, all the measures were re-computed and compared in order to get the top-performing model-technique combination out of them. The best ones were then compared with the old model's results.

## 5.2 KEY RESULTS

Considering all the aspects available, so time, space and performance, the best overall method found was the ARIMAX reconciled using the minimization of the trace. Looking only at the performance, the best model is the ARIMAX and TBATS-NN ensemble, but it has a much higher training time and higher computational costs. Both the methods still have better performance compared to the previously implemented model, as shown in Figure 4.4 and Figure 4.5.

The new model, the TBATS-NN hybrid, returns really promising results, and in some countries it also outperforms the other models. Given its complexity, this new approach should be considered only if the high computational load the TBATS needs to check all the different alternatives and the training and usage of the residual NN are not a problem.

Other insights are that the NND technique for the reconciliation does not seem to work well for this problem: while it captures well how the lower levels of the least selling products behave, its performance worsens with the most selling products and leads to one of the worst accuracies at the top level.

## 5.3 FUTURE WORK

In this work, the new hybrid method shows promise, but it should be implemented again. The model could be used on other publicly available bigger datasets, to check if it is an actual improvement over other forecasting models.

Looking at the work for the company itself, there are a few things that could be improved given the time. For example, the simplest ensemble returned great results, so optimizing it in order to optimally combine the results of the two models it uses could improve even more the forecasts.

Another upgrade that could be proposed is using the best model found for each country: as seen in Figure 4.4, there is no real winner for all of them. The client asked for only one algorithm, but to improve the predictions it could be a good idea to have the program confront all the possible alternatives that were described and then choose the top-performing one. This of course would lead to a much higher computational cost and longer times, so this option can be considered only if this is not an issue for the company.

Once more data is collected, the NDD approach could be tried again. The lack of results is likely due to the little data available, which is highly detrimental to NN algorithms. The whole validation process should be repeated once again to find the optimal combination of parameters. Investing in new exogenous variables which are capable of distinguish better how the different products and their split behave could also improve the algorithm.

# 6 APPENDIX

## 6.1 METRICS RESULTS

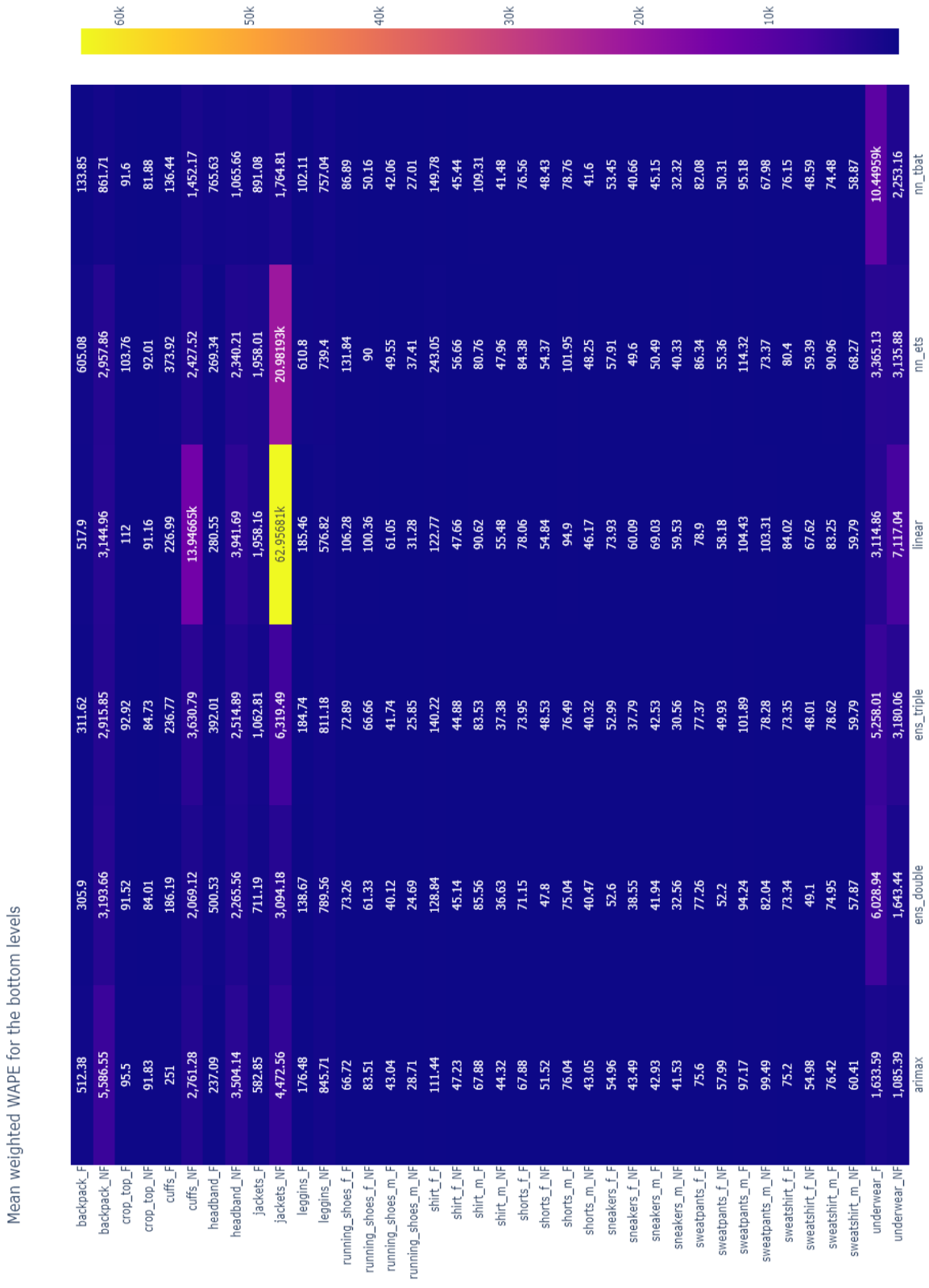


*Figure 6.1*

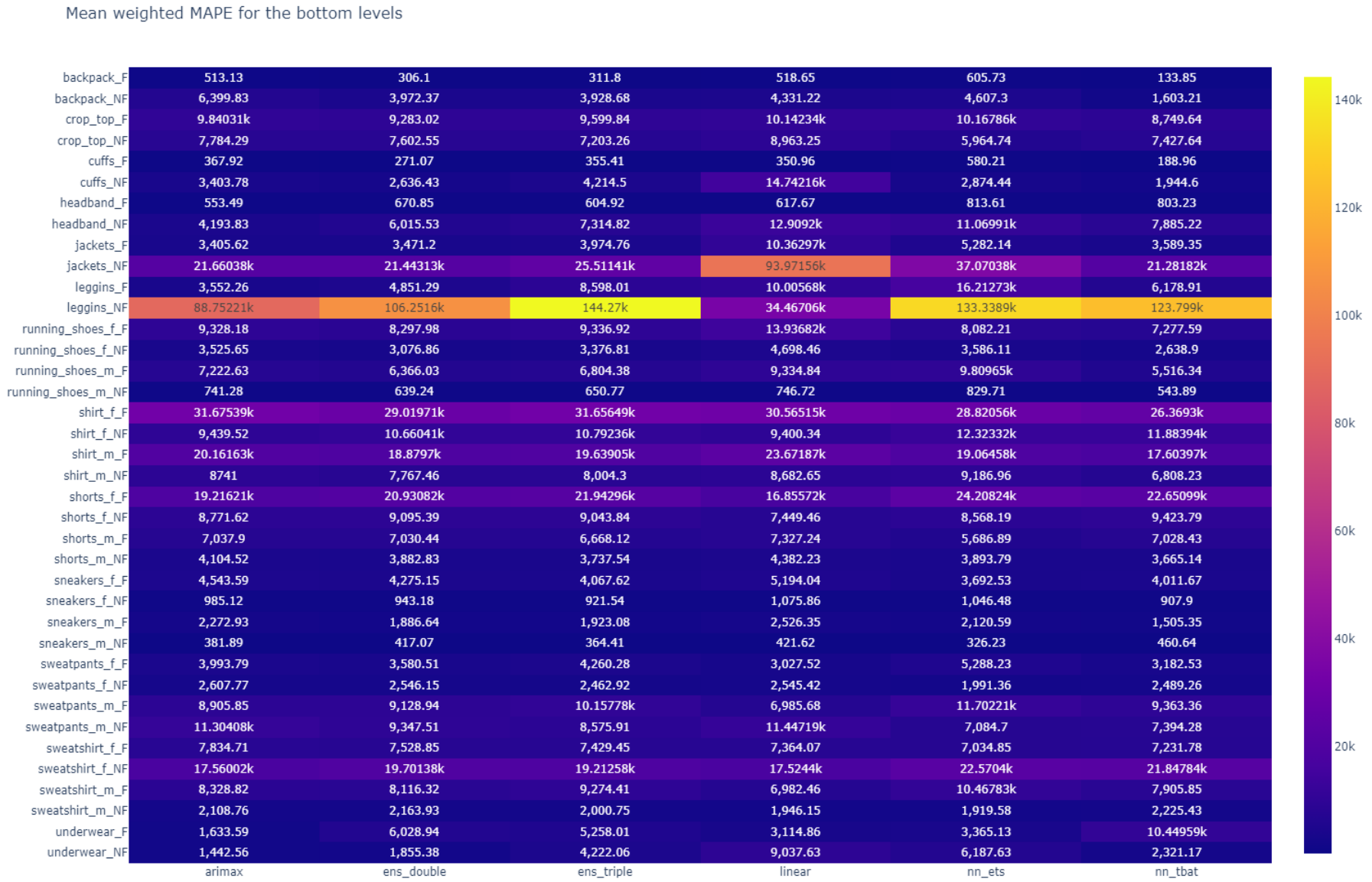
Mean weighted MAPE for the bottom levels
backpack_F 513.13 306.1 311.8 518.65 605.73 133.85
backpack_NF 6,399.83 3,972.37 3,928.68 4,331.22 4,607.3 1,603.21
crop_top_F 9.84031k 9,283.02 9,599.84 10.14234k 10.16786k 8,749.64
crop_top_NF 7,784.29 7,602.55 7,203.26 8,963.25 5,964.74 7,427.64
cuffs_F 367.92 271.07 355.41 350.96 580.21 188.96
cuffs_NF 3,403.78 2,636.43 4,214.5 14.74216k 2,874.44 1,944.6
headband_F 553.49 670.85 604.92 617.67 813.61 803.23
headband_NF 4,193.83 6,015.53 7,314.82 12.9092k 11.06991k 7,885.22
jackets_F 3,405.62 3,471.2 3,974.76 10.36297k 5,282.14 3,589.35
jackets_NF 21.66038k 21.44313k 25.51141k 93.97156k 37.07038k 21.28182k
leggins_F 3,552.26 4,851.29 8,598.01 10.00568k 16.21273k 6,178.91
leggins_NF 88.75221k 106.2516k 144.27k 34.46706k 133.3389k 123.799k
running_shoes_f_F 9,328.18 8,297.98 9,336.92 13.93682k 8,082.21 7,277.59
running_shoes_f_NF 3,525.65 3,076.86 3,376.81 4,698.46 3,586.11 2,638.9
running_shoes_m_F 7,222.63 6,366.03 6,804.38 9,334.84 9.80965k 5,516.34
running_shoes_m_NF 741.28 639.24 650.77 746.72 829.71 543.89
shirt_f_F 31.67539k 29.01971k 31.65649k 30.56515k 28.82056k 26.3693k
shirt_f_NF 9,439.52 10.66041k 10.79236k 9,400.34 12.32332k 11.88394k
shirt_m_F 20.16163k 18.8797k 19.63905k 23.67187k 19.06458k 17.60397k
shirt_m_NF 8741 7,767.46 8,004.3 8,682.65 9,186.96 6,808.23
shorts_f_F 19.21621k 20.93082k 21.94296k 16.85572k 24.20824k 22.65099k
shorts_f_NF 8,771.62 9,095.39 9,043.84 7,449.46 8,568.19 9,423.79
shorts_m_F 7,037.9 7,030.44 6,668.12 7,327.24 5,686.89 7,028.43
shorts_m_NF 4,104.52 3,882.83 3,737.54 4,382.23 3,893.79 3,665.14
sneakers_f_F 4,543.59 4,275.15 4,067.62 5,194.04 3,692.53 4,011.67
sneakers_f_NF 985.12 943.18 921.54 1,075.86 1,046.48 907.9
sneakers_m_F 2,272.93 1,886.64 1,923.08 2,526.35 2,120.59 1,505.35
sneakers_m_NF 381.89 417.07 364.41 421.62 326.23 460.64
sweatpants_f_F 3,993.79 3,580.51 4,260.28 3,027.52 5,288.23 3,182.53
sweatpants_f_NF 2,607.77 2,546.15 2,462.92 2,545.42 1,991.36 2,489.26
sweatpants_m_F 8,905.85 9,128.94 10.15778k 6,985.68 11.70221k 9,363.36
sweatpants_m_NF 11.30408k 9,347.51 8,575.91 11.44719k 7,084.7 7,394.28
sweatshirt_f_F 7,834.71 7,528.85 7,429.45 7,364.07 7,034.85 7,231.78
sweatshirt_f_NF 17.56002k 19.70138k 19.21258k 17.5244k 22.5704k 21.84784k
sweatshirt_m_F 8,328.82 8,116.32 9,274.41 6,982.46 10.46783k 7,905.85
sweatshirt_m_NF 2,108.76 2,163.93 2,000.75 1,946.15 1,919.58 2,225.43
underwear_F 1,633.59 6,028.94 5,258.01 3,114.86 3,365.13 10.44959k
underwear_NF 1,442.56 1,855.38 4,222.06 9,037.63 6,187.63 2,321.17
arimax ens_double ens_triple linear nn_ets nn_tbat
140k
120k
100k
80k
60k
40k
20k

*Figure 6.2*

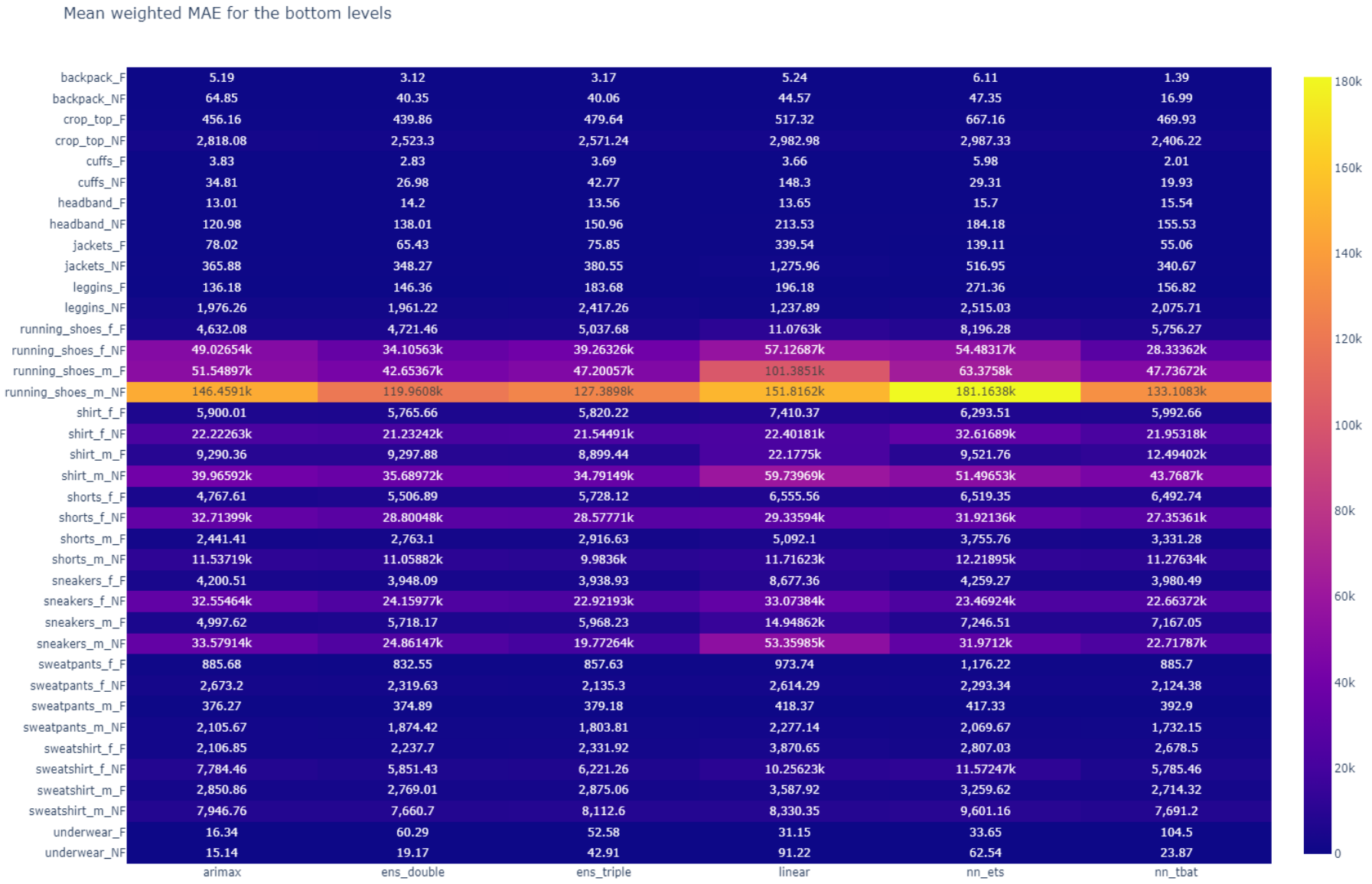
Mean weighted MAE for the bottom levels
backpack_F 5.19 3.12 3.17 5.24 6.11 1.39
backpack_NF 64.85 40.35 40.06 44.57 47.35 16.99
crop_top_F 456.16 439.86 479.64 517.32 667.16 469.93
crop_top_NF 2,818.08 2,523.3 2,571.24 2,982.98 2,987.33 2,406.22
cuffs_F 3.83 2.83 3.69 3.66 5.98 2.01
cuffs_NF 34.81 26.98 42.77 148.3 29.31 19.93
headband_F 13.01 14.2 13.56 13.65 15.7 15.54
headband_NF 120.98 138.01 150.96 213.53 184.18 155.53
jackets_F 78.02 65.43 75.85 339.54 139.11 55.06
jackets_NF 365.88 348.27 380.55 1,275.96 516.95 340.67
leggins_F 136.18 146.36 183.68 196.18 271.36 156.82
leggins_NF 1,976.26 1,961.22 2,417.26 1,237.89 2,515.03 2,075.71
running_shoes_f_F 4,632.08 4,721.46 5,037.68 11.0763k 8,196.28 5,756.27
running_shoes_f_NF 49.02654k 34.10563k 39.26326k 57.12687k 54.48317k 28.33362k
running_shoes_m_F 51.54897k 42.65367k 47.20057k 101.3851k 63.3758k 47.73672k
running_shoes_m_NF 146.4591k 119.9608k 127.3898k 151.8162k 181.1638k 133.1083k
shirt_f_F 5,900.01 5,765.66 5,820.22 7,410.37 6,293.51 5,992.66
shirt_f_NF 22.22263k 21.23242k 21.54491k 22.40181k 32.61689k 21.95318k
shirt_m_F 9,290.36 9,297.88 8,899.44 22.1775k 9,521.76 12.49402k
shirt_m_NF 39.96592k 35.68972k 34.79149k 59.73969k 51.49653k 43.7687k
shorts_f_F 4,767.61 5,506.89 5,728.12 6,555.56 6,519.35 6,492.74
shorts_f_NF 32.71399k 28.80048k 28.57771k 29.33594k 31.92136k 27.35361k
shorts_m_F 2,441.41 2,763.1 2,916.63 5,092.1 3,755.76 3,331.28
shorts_m_NF 11.53719k 11.05882k 9.9836k 11.71623k 12.21895k 11.27634k
sneakers_f_F 4,200.51 3,948.09 3,938.93 8,677.36 4,259.27 3,980.49
sneakers_f_NF 32.55464k 24.15977k 22.92193k 33.07384k 23.46924k 22.66372k
sneakers_m_F 4,997.62 5,718.17 5,968.23 14.94862k 7,246.51 7,167.05
sneakers_m_NF 33.57914k 24.86147k 19.77264k 53.35985k 31.9712k 22.71787k
sweatpants_f_F 885.68 832.55 857.63 973.74 1,176.22 885.7
sweatpants_f_NF 2,673.2 2,319.63 2,135.3 2,614.29 2,293.34 2,124.38
sweatpants_m_F 376.27 374.89 379.18 418.37 417.33 392.9
sweatpants_m_NF 2,105.67 1,874.42 1,803.81 2,277.14 2,069.67 1,732.15
sweatshirt_f_F 2,106.85 2,237.7 2,331.92 3,870.65 2,807.03 2,678.5
sweatshirt_f_NF 7,784.46 5,851.43 6,221.26 10.25623k 11.57247k 5,785.46
sweatshirt_m_F 2,850.86 2,769.01 2,875.06 3,587.92 3,259.62 2,714.32
sweatshirt_m_NF 7,946.76 7,660.7 8,112.6 8,330.35 9,601.16 7,691.2
underwear_F 16.34 60.29 52.58 31.15 33.65 104.5
underwear_NF 15.14 19.17 42.91 91.22 62.54 23.87
arimax ens_double ens_triple linear nn_ets nn_tbat
180k
160k
140k
120k
100k
80k
60k
40k
20k
0

*Figure 6.3*

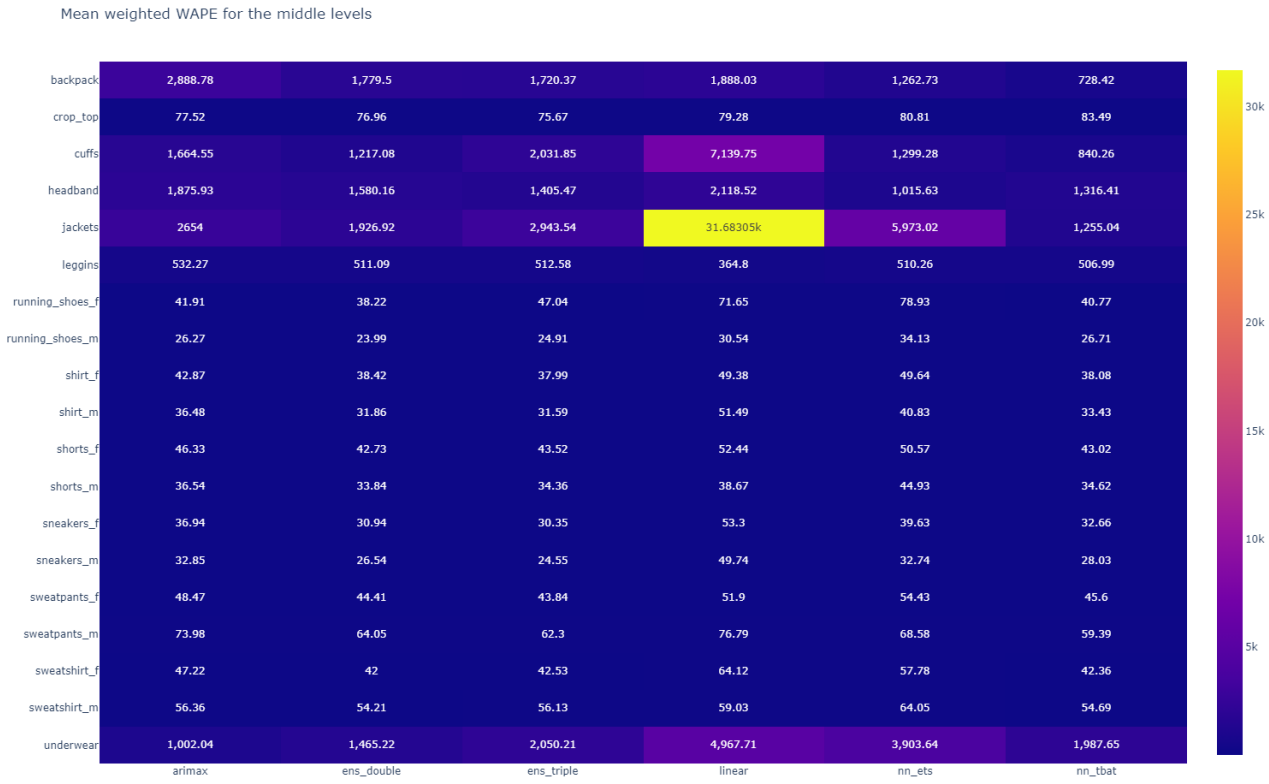


*Figure 6.4*

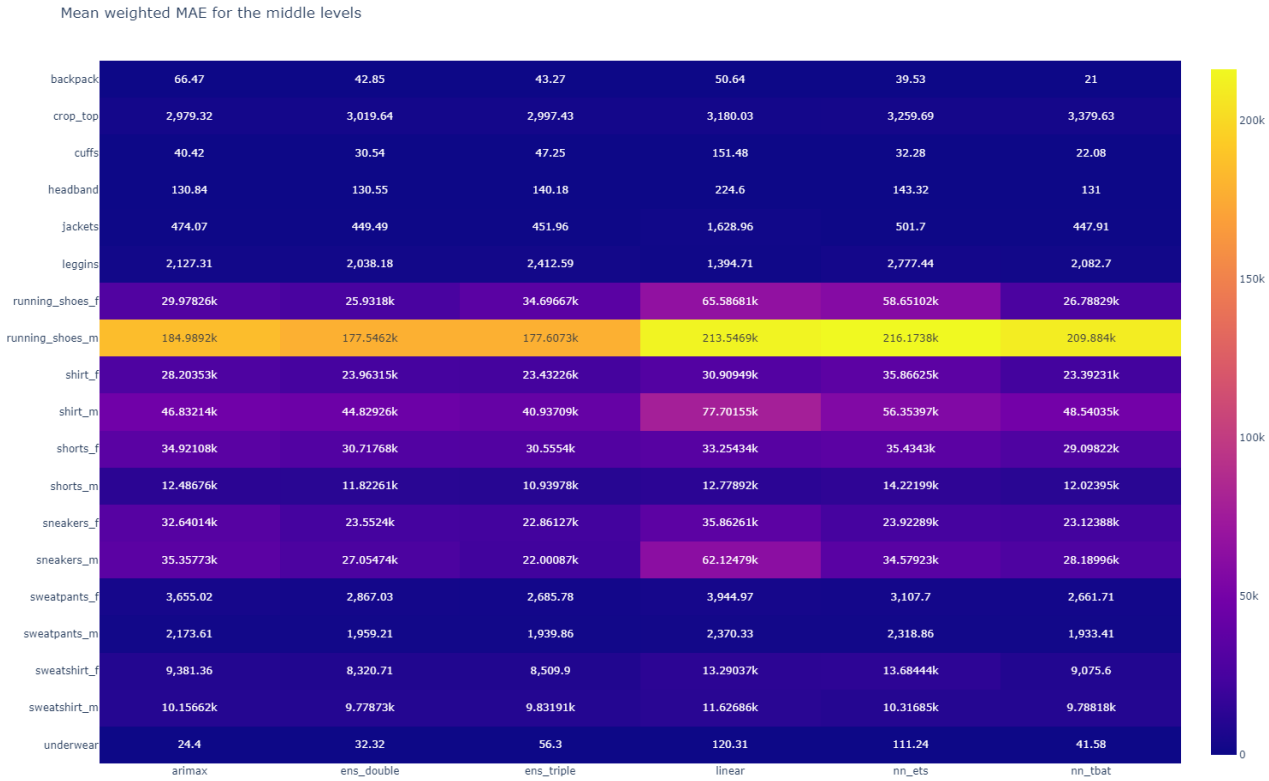


*Figure 6.5*

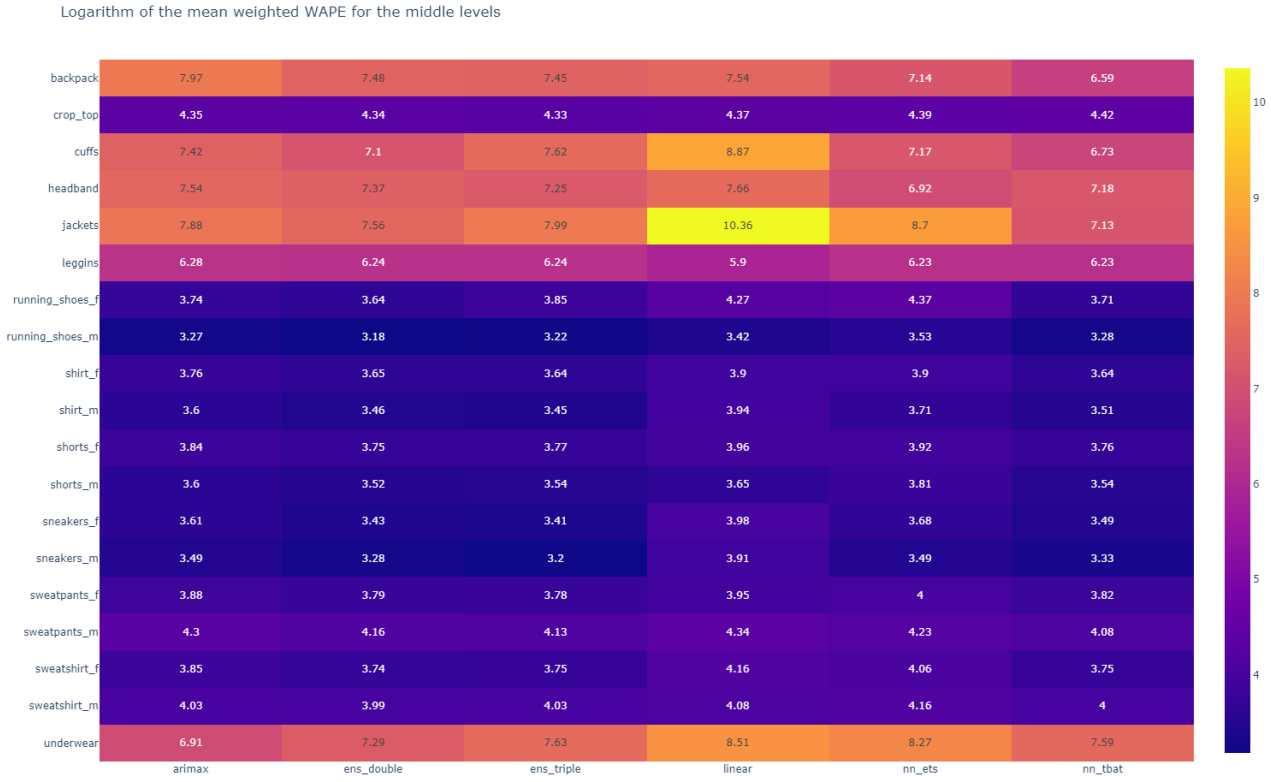


*Figure 6.6*

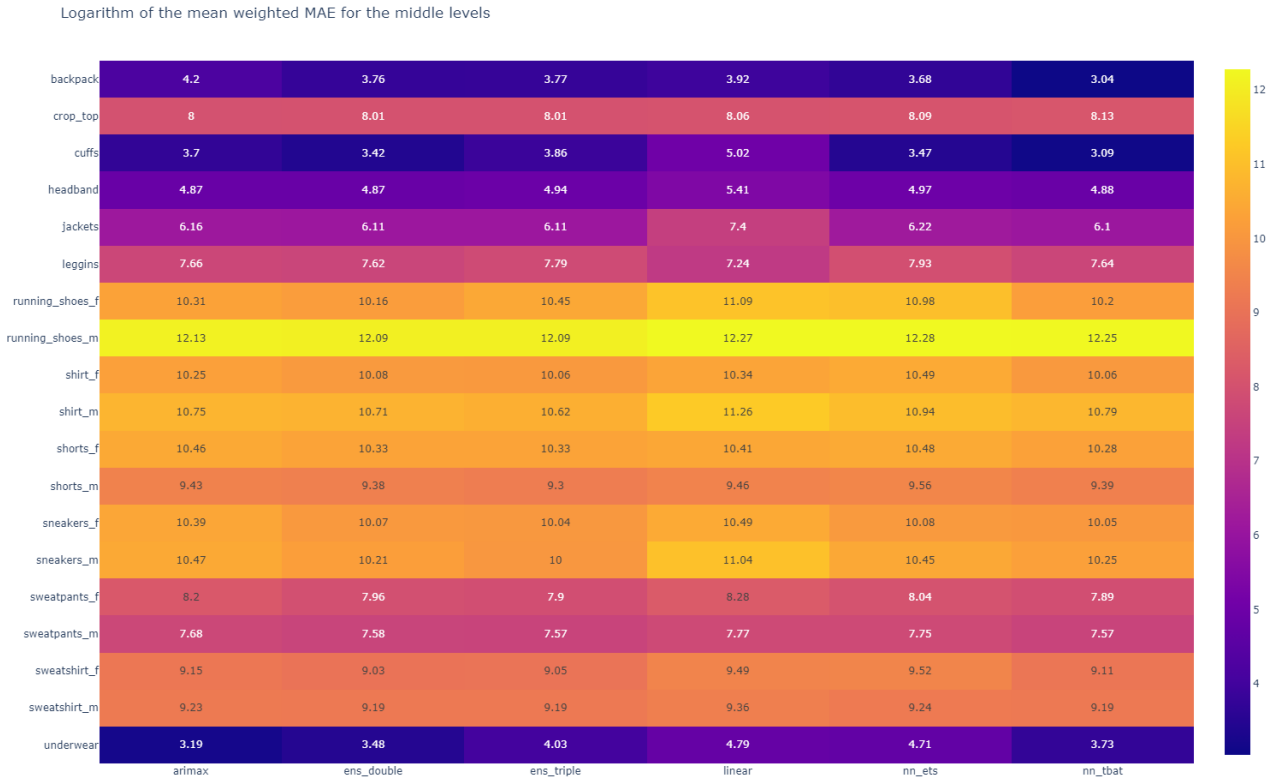


*Figure 6.7*

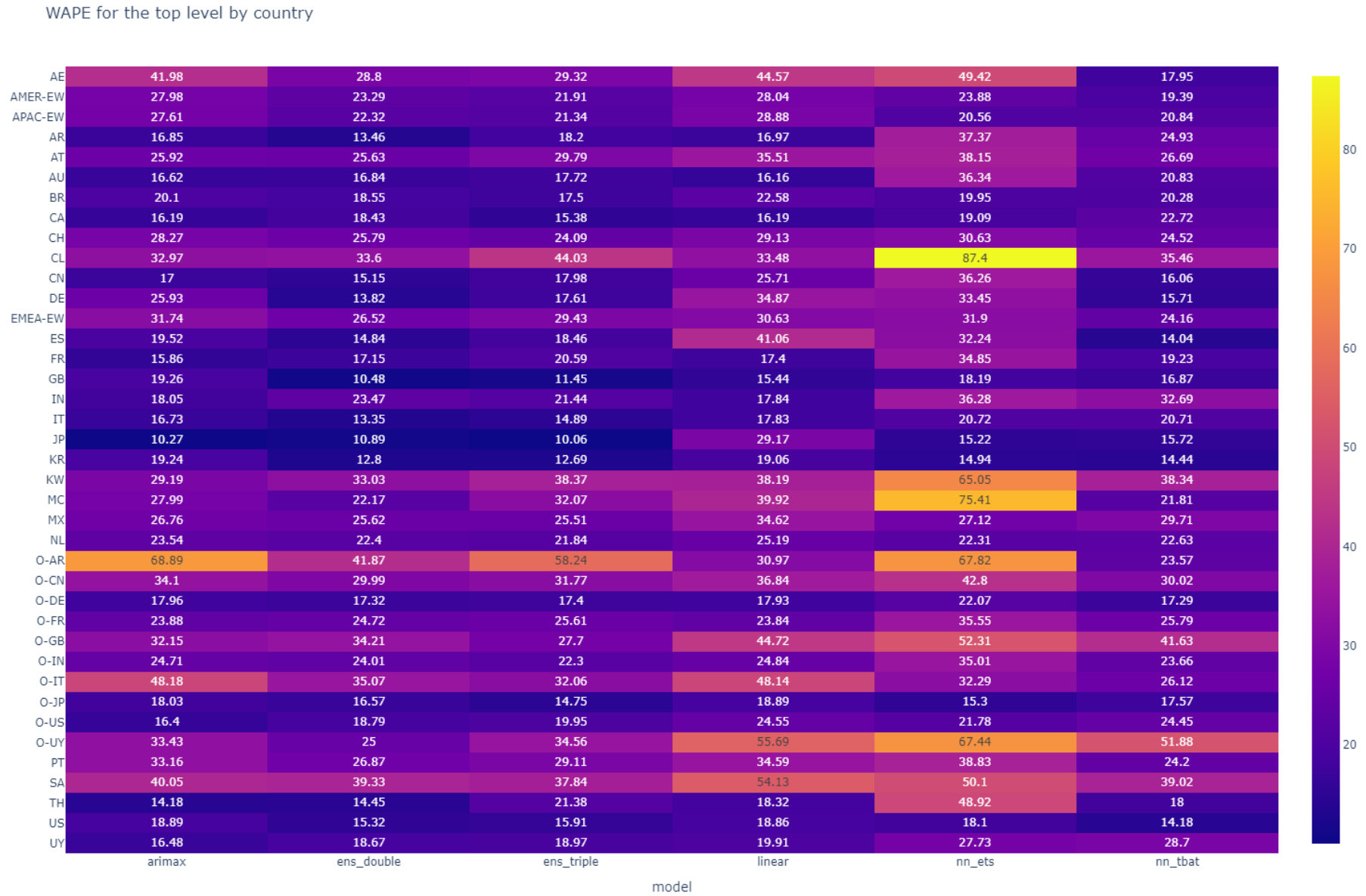
WAPE for the top level by country
AE 41.98 28.8 29.32 44.57 49.42 17.95
AMER-EW 27.98 23.29 21.91 28.04 23.88 19.39
APAC-EW 27.61 22.32 21.34 28.88 20.56 20.84
AR 16.85 13.46 18.2 16.97 37.37 24.93
AT 25.92 25.63 29.79 35.51 38.15 26.69
AU 16.62 16.84 17.72 16.16 36.34 20.83
BR 20.1 18.55 17.5 22.58 19.95 20.28
CA 16.19 18.43 15.38 16.19 19.09 22.72
CH 28.27 25.79 24.09 29.13 30.63 24.52
CL 32.97 33.6 44.03 33.48 87.4 35.46
CN 17 15.15 17.98 25.71 36.26 16.06
DE 25.93 13.82 17.61 34.87 33.45 15.71
EMEA-EW 31.74 26.52 29.43 30.63 31.9 24.16
ES 19.52 14.84 18.46 41.06 32.24 14.04
FR 15.86 17.15 20.59 17.4 34.85 19.23
GB 19.26 10.48 11.45 15.44 18.19 16.87
IN 18.05 23.47 21.44 17.84 36.28 32.69
IT 16.73 13.35 14.89 17.83 20.72 20.71
JP 10.27 10.89 10.06 29.17 15.22 15.72
KR 19.24 12.8 12.69 19.06 14.94 14.44
KW 29.19 33.03 38.37 38.19 65.05 38.34
MC 27.99 22.17 32.07 39.92 75.41 21.81
MX 26.76 25.62 25.51 34.62 27.12 29.71
NL 23.54 22.4 21.84 25.19 22.31 22.63
O-AR 68.89 41.87 58.24 30.97 67.82 23.57
O-CN 34.1 29.99 31.77 36.84 42.8 30.02
O-DE 17.96 17.32 17.4 17.93 22.07 17.29
O-FR 23.88 24.72 25.61 23.84 35.55 25.79
O-GB 32.15 34.21 27.7 44.72 52.31 41.63
O-IN 24.71 24.01 22.3 24.84 35.01 23.66
O-IT 48.18 35.07 32.06 48.14 32.29 26.12
O-JP 18.03 16.57 14.75 18.89 15.3 17.57
O-US 16.4 18.79 19.95 24.55 21.78 24.45
O-UY 33.43 25 34.56 55.69 67.44 51.88
PT 33.16 26.87 29.11 34.59 38.83 24.2
SA 40.05 39.33 37.84 54.13 50.1 39.02
TH 14.18 14.45 21.38 18.32 48.92 18
US 18.89 15.32 15.91 18.86 18.1 14.18
UY 16.48 18.67 18.97 19.91 27.73 28.7
arimax ens_double ens_triple linear nn_ets nn_tbat
model
80 70 60 50 40 30 20

*Figure 6.8*

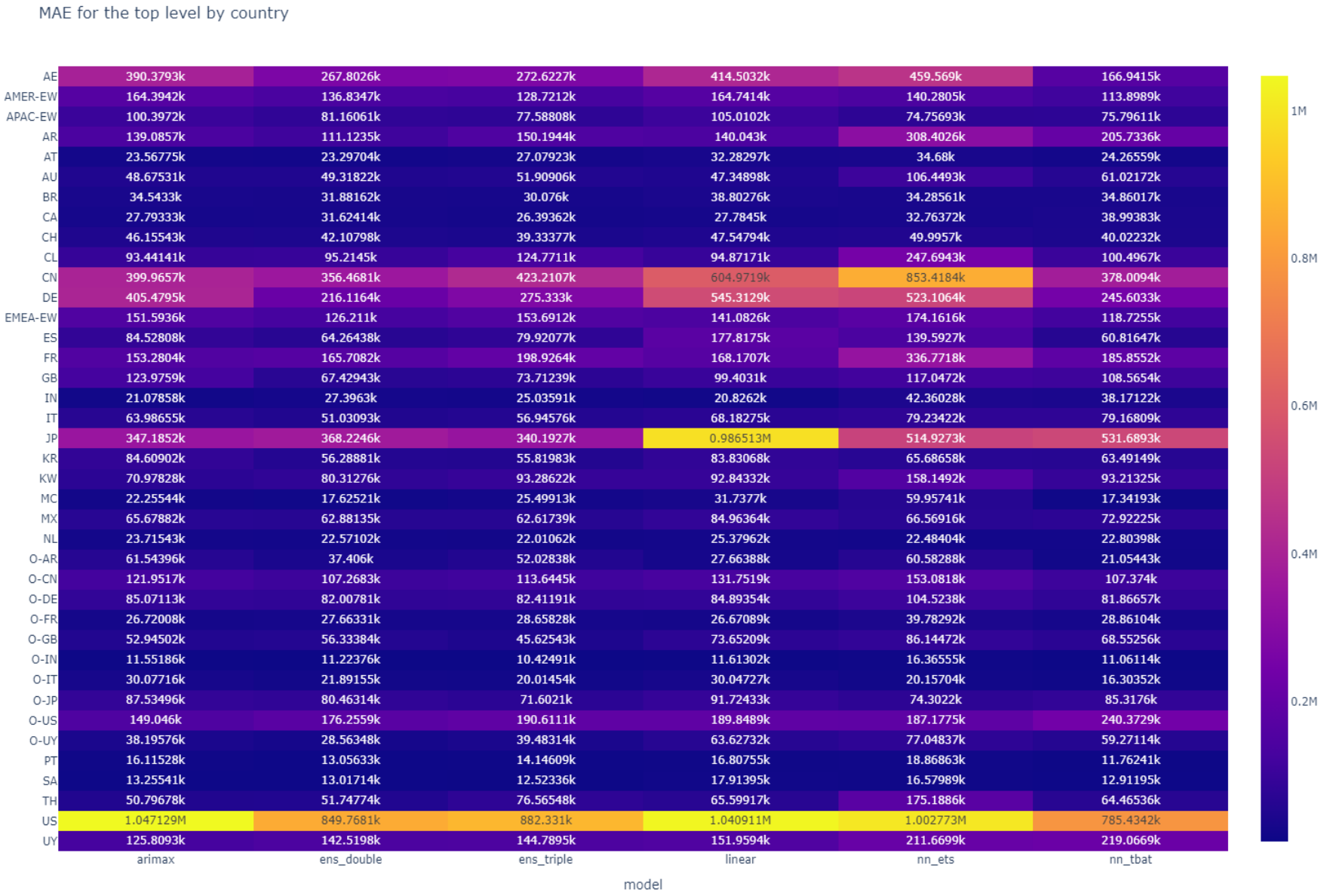
MAE for the top level by country
AE 390.3793k 267.8026k 272.6227k 414.5032k 459.569k 166.9415k
AMER-EW 164.3942k 136.8347k 128.7212k 164.7414k 140.2805k 113.8989k
APAC-EW 100.3972k 81.16061k 77.58808k 105.0102k 74.75693k 75.79611k
AR 139.0857k 111.1235k 150.1944k 140.043k 308.4026k 205.7336k
AT 23.56775k 23.29704k 27.07923k 32.28297k 34.68k 24.26559k
AU 48.67531k 49.31822k 51.90906k 47.34898k 106.4493k 61.02172k
BR 34.5433k 31.88162k 30.076k 38.80276k 34.28561k 34.86017k
CA 27.79333k 31.62414k 26.39362k 27.7845k 32.76372k 38.99383k
CH 46.15543k 42.10798k 39.33377k 47.54794k 49.9957k 40.02232k
CL 93.44141k 95.2145k 124.7711k 94.87171k 247.6943k 100.4967k
CN 399.9657k 356.4681k 423.2107k 604.9719k 853.4184k 378.0094k
DE 405.4795k 216.1164k 275.333k 545.3129k 523.1064k 245.6033k
EMEA-EW 151.5936k 126.211k 153.6912k 141.0826k 174.1616k 118.7255k
ES 84.52808k 64.26438k 79.92077k 177.8175k 139.5927k 60.81647k
FR 153.2804k 165.7082k 198.9264k 168.1707k 336.7718k 185.8552k
GB 123.9759k 67.42943k 73.71239k 99.4031k 117.0472k 108.5654k
IN 21.07858k 27.3963k 25.03591k 20.8262k 42.36028k 38.17122k
IT 63.98655k 51.03093k 56.94576k 68.18275k 79.23422k 79.16809k
JP 347.1852k 368.2246k 340.1927k 0.986513M 514.9273k 531.6893k
KR 84.60902k 56.28881k 55.81983k 83.83068k 65.68658k 63.49149k
KW 70.97828k 80.31276k 93.28622k 92.84332k 158.1492k 93.21325k
MC 22.25544k 17.62521k 25.49913k 31.7377k 59.95741k 17.34193k
MX 65.67882k 62.88135k 62.61739k 84.96364k 66.56916k 72.92225k
NL 23.71543k 22.57102k 22.01062k 25.37962k 22.48404k 22.80398k
O-AR 61.54396k 37.406k 52.02838k 27.66388k 60.58288k 21.05443k
O-CN 121.9517k 107.2683k 113.6445k 131.7519k 153.0818k 107.374k
O-DE 85.07113k 82.00781k 82.41191k 84.89354k 104.5238k 81.86657k
O-FR 26.72008k 27.66331k 28.65828k 26.67089k 39.78292k 28.86104k
O-GB 52.94502k 56.33384k 45.62543k 73.65209k 86.14472k 68.55256k
O-IN 11.55186k 11.22376k 10.42491k 11.61302k 16.36555k 11.06114k
O-IT 30.07716k 21.89155k 20.01454k 30.04727k 20.15704k 16.30352k
O-JP 87.53496k 80.46314k 71.6021k 91.72433k 74.3022k 85.3176k
O-US 149.046k 176.2559k 190.6111k 189.8489k 187.1775k 240.3729k
O-UY 38.19576k 28.56348k 39.48314k 63.62732k 77.04837k 59.27114k
PT 16.11528k 13.05633k 14.14609k 16.80755k 18.86863k 11.76241k
SA 13.25541k 13.01714k 12.52336k 17.91395k 16.57989k 12.91195k
TH 50.79678k 51.74774k 76.56548k 65.59917k 175.1886k 64.46536k
US 1.047129M 849.7681k 882.331k 1.040911M 1.002773M 785.4342k
UY 125.8093k 142.5198k 144.7895k 151.9594k 211.6699k 219.0669k
arimax ens_double ens_triple linear nn_ets nn_tbat
model
1M
0.8M
0.6M
0.4M
0.2M

*Figure 6.9*

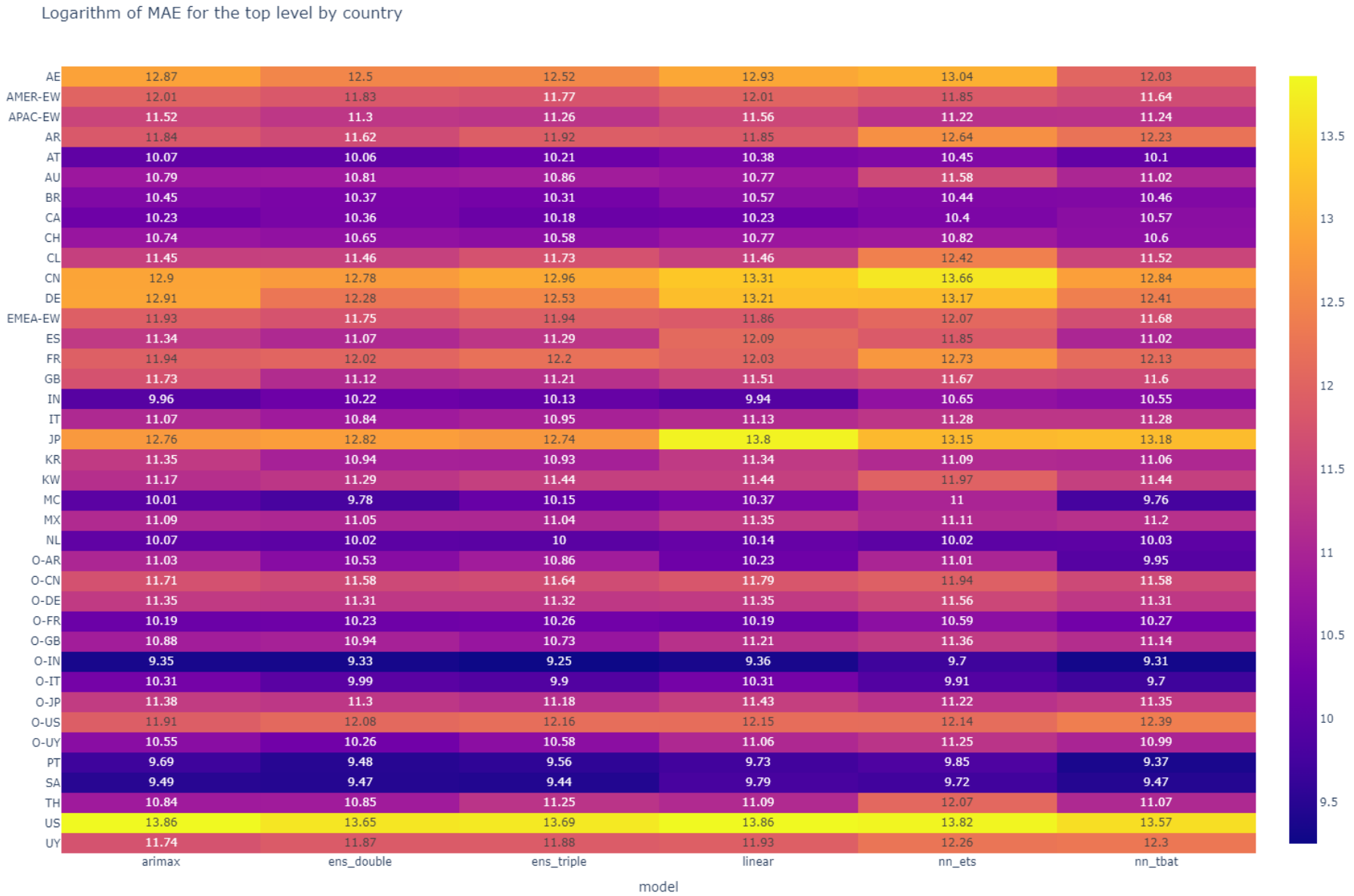
Logarithm of MAE for the top level by country
AE 12.87 12.5 12.52 12.93 13.04 12.03
AMER-EW 12.01 11.83 11.77 12.01 11.85 11.64
APAC-EW 11.52 11.3 11.26 11.56 11.22 11.24
AR 11.84 11.62 11.92 11.85 12.64 12.23
AT 10.07 10.06 10.21 10.38 10.45 10.1
AU 10.79 10.81 10.86 10.77 11.58 11.02
BR 10.45 10.37 10.31 10.57 10.44 10.46
CA 10.23 10.36 10.18 10.23 10.4 10.57
CH 10.74 10.65 10.58 10.77 10.82 10.6
CL 11.45 11.46 11.73 11.46 12.42 11.52
CN 12.9 12.78 12.96 13.31 13.66 12.84
DE 12.91 12.28 12.53 13.21 13.17 12.41
EMEA-EW 11.93 11.75 11.94 11.86 12.07 11.68
ES 11.34 11.07 11.29 12.09 11.85 11.02
FR 11.94 12.02 12.2 12.03 12.73 12.13
GB 11.73 11.12 11.21 11.51 11.67 11.6
IN 9.96 10.22 10.13 9.94 10.65 10.55
IT 11.07 10.84 10.95 11.13 11.28 11.28
JP 12.76 12.82 12.74 13.8 13.15 13.18
KR 11.35 10.94 10.93 11.34 11.09 11.06
KW 11.17 11.29 11.44 11.44 11.97 11.44
MC 10.01 9.78 10.15 10.37 11 9.76
MX 11.09 11.05 11.04 11.35 11.11 11.2
NL 10.07 10.02 10 10.14 10.02 10.03
O-AR 11.03 10.53 10.86 10.23 11.01 9.95
O-CN 11.71 11.58 11.64 11.79 11.94 11.58
O-DE 11.35 11.31 11.32 11.35 11.56 11.31
O-FR 10.19 10.23 10.26 10.19 10.59 10.27
O-GB 10.88 10.94 10.73 11.21 11.36 11.14
O-IN 9.35 9.33 9.25 9.36 9.7 9.31
O-IT 10.31 9.99 9.9 10.31 9.91 9.7
O-JP 11.38 11.3 11.18 11.43 11.22 11.35
O-US 11.91 12.08 12.16 12.15 12.14 12.39
O-UY 10.55 10.26 10.58 11.06 11.25 10.99
PT 9.69 9.48 9.56 9.73 9.85 9.37
SA 9.49 9.47 9.44 9.79 9.72 9.47
TH 10.84 10.85 11.25 11.09 12.07 11.07
US 13.86 13.65 13.69 13.86 13.82 13.57
UY 11.74 11.87 11.88 11.93 12.26 12.3
arimax ens_double ens_triple linear nn_ets nn_tbat
model
13.5
13
12.5
12
11.5
11
10.5
10
9.5

*Figure 6.10*

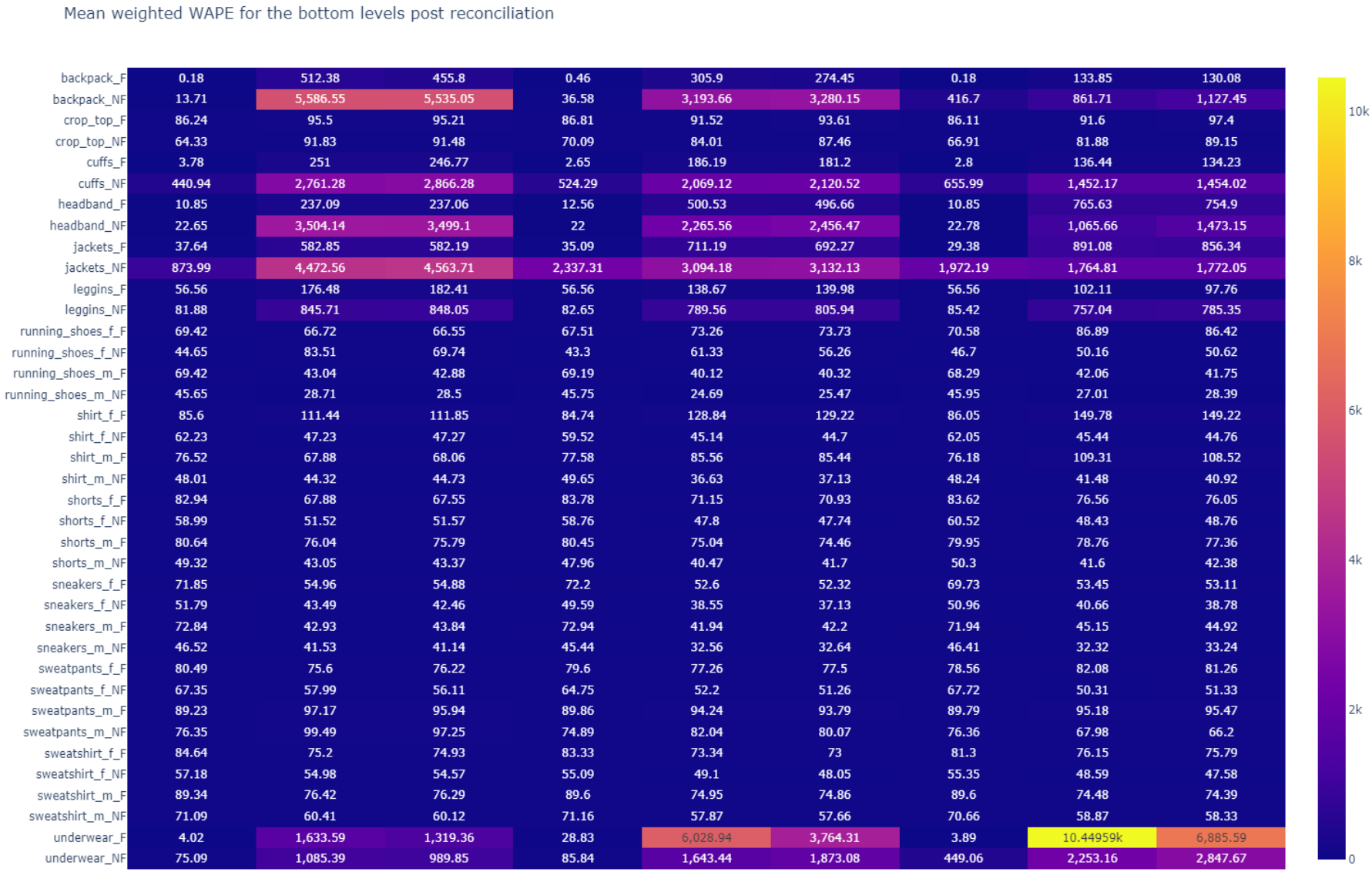
Mean weighted WAPE for the bottom levels post reconciliation
backpack_F 0.18 512.38 455.8 0.46 305.9 274.45 0.18 133.85 130.08
backpack_NF 13.71 5,586.55 5,535.05 36.58 3,193.66 3,280.15 416.7 861.71 1,127.45
crop_top_F 86.24 95.5 95.21 86.81 91.52 93.61 86.11 91.6 97.4
crop_top_NF 64.33 91.83 91.48 70.09 84.01 87.46 66.91 81.88 89.15
cuffs_F 3.78 251 246.77 2.65 186.19 181.2 2.8 136.44 134.23
cuffs_NF 440.94 2,761.28 2,866.28 524.29 2,069.12 2,120.52 655.99 1,452.17 1,454.02
headband_F 10.85 237.09 237.06 12.56 500.53 496.66 10.85 765.63 754.9
headband_NF 22.65 3,504.14 3,499.1 22 2,265.56 2,456.47 22.78 1,065.66 1,473.15
jackets_F 37.64 582.85 582.19 35.09 711.19 692.27 29.38 891.08 856.34
jackets_NF 873.99 4,472.56 4,563.71 2,337.31 3,094.18 3,132.13 1,972.19 1,764.81 1,772.05
leggins_F 56.56 176.48 182.41 56.56 138.67 139.98 56.56 102.11 97.76
leggins_NF 81.88 845.71 848.05 82.65 789.56 805.94 85.42 757.04 785.35
running_shoes_f_F 69.42 66.72 66.55 67.51 73.26 73.73 70.58 86.89 86.42
running_shoes_f_NF 44.65 83.51 69.74 43.3 61.33 56.26 46.7 50.16 50.62
running_shoes_m_F 69.42 43.04 42.88 69.19 40.12 40.32 68.29 42.06 41.75
running_shoes_m_NF 45.65 28.71 28.5 45.75 24.69 25.47 45.95 27.01 28.39
shirt_f_F 85.6 111.44 111.85 84.74 128.84 129.22 86.05 149.78 149.22
shirt_f_NF 62.23 47.23 47.27 59.52 45.14 44.7 62.05 45.44 44.76
shirt_m_F 76.52 67.88 68.06 77.58 85.56 85.44 76.18 109.31 108.52
shirt_m_NF 48.01 44.32 44.73 49.65 36.63 37.13 48.24 41.48 40.92
shorts_f_F 82.94 67.88 67.55 83.78 71.15 70.93 83.62 76.56 76.05
shorts_f_NF 58.99 51.52 51.57 58.76 47.8 47.74 60.52 48.43 48.76
shorts_m_F 80.64 76.04 75.79 80.45 75.04 74.46 79.95 78.76 77.36
shorts_m_NF 49.32 43.05 43.37 47.96 40.47 41.7 50.3 41.6 42.38
sneakers_f_F 71.85 54.96 54.88 72.2 52.6 52.32 69.73 53.45 53.11
sneakers_f_NF 51.79 43.49 42.46 49.59 38.55 37.13 50.96 40.66 38.78
sneakers_m_F 72.84 42.93 43.84 72.94 41.94 42.2 71.94 45.15 44.92
sneakers_m_NF 46.52 41.53 41.14 45.44 32.56 32.64 46.41 32.32 33.24
sweatpants_f_F 80.49 75.6 76.22 79.6 77.26 77.5 78.56 82.08 81.26
sweatpants_f_NF 67.35 57.99 56.11 64.75 52.2 51.26 67.72 50.31 51.33
sweatpants_m_F 89.23 97.17 95.94 89.86 94.24 93.79 89.79 95.18 95.47
sweatpants_m_NF 76.35 99.49 97.25 74.89 82.04 80.07 76.36 67.98 66.2
sweatshirt_f_F 84.64 75.2 74.93 83.33 73.34 73 81.3 76.15 75.79
sweatshirt_f_NF 57.18 54.98 54.57 55.09 49.1 48.05 55.35 48.59 47.58
sweatshirt_m_F 89.34 76.42 76.29 89.6 74.95 74.86 89.6 74.48 74.39
sweatshirt_m_NF 71.09 60.41 60.12 71.16 57.87 57.66 70.66 58.87 58.33
underwear_F 4.02 1,633.59 1,319.36 28.83 6,028.94 3,764.31 3.89 10.44959k 6,885.59
underwear_NF 75.09 1,085.39 989.85 85.84 1,643.44 1,873.08 449.06 2,253.16 2,847.67
arimax_DNN arimax_bu arimax_minT ens_double_DNN ens_double_bu ens_double_minT tbat_nn_DNN tbat_nn_bu tbat_nn_minT
10k
8k
6k
4k
2k
0

*Figure 6.11*

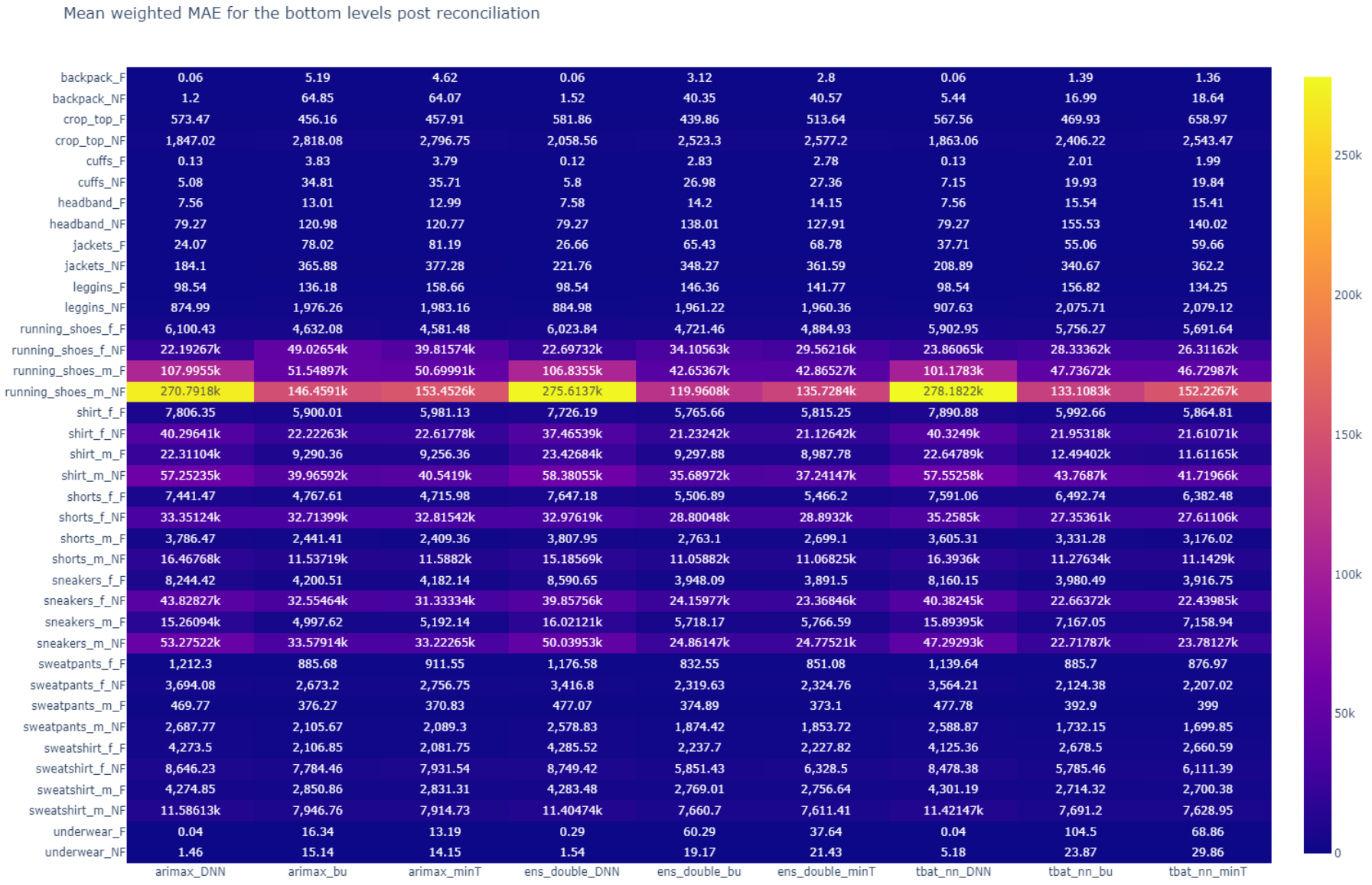
Mean weighted MAE for the bottom levels post reconciliation
backpack_F 0.06 5.19 4.62 0.06 3.12 2.8 0.06 1.39 1.36
backpack_NF 1.2 64.85 64.07 1.52 40.35 40.57 5.44 16.99 18.64
crop_top_F 573.47 456.16 457.91 581.86 439.86 513.64 567.56 469.93 658.97
crop_top_NF 1,847.02 2,818.08 2,796.75 2,058.56 2,523.3 2,577.2 1,863.06 2,406.22 2,543.47
cuffs_F 0.13 3.83 3.79 0.12 2.83 2.78 0.13 2.01 1.99
cuffs_NF 5.08 34.81 35.71 5.8 26.98 27.36 7.15 19.93 19.84
headband_F 7.56 13.01 12.99 7.58 14.2 14.15 7.56 15.54 15.41
headband_NF 79.27 120.98 120.77 79.27 138.01 127.91 79.27 155.53 140.02
jackets_F 24.07 78.02 81.19 26.66 65.43 68.78 37.71 55.06 59.66
jackets_NF 184.1 365.88 377.28 221.76 348.27 361.59 208.89 340.67 362.2
leggins_F 98.54 136.18 158.66 98.54 146.36 141.77 98.54 156.82 134.25
leggins_NF 874.99 1,976.26 1,983.16 884.98 1,961.22 1,960.36 907.63 2,075.71 2,079.12
running_shoes_f_F 6,100.43 4,632.08 4,581.48 6,023.84 4,721.46 4,884.93 5,902.95 5,756.27 5,691.64
running_shoes_f_NF 22.19267k 49.02654k 39.81574k 22.69732k 34.10563k 29.56216k 23.86065k 28.33362k 26.31162k
running_shoes_m_F 107.9955k 51.54897k 50.69991k 106.8355k 42.65367k 42.86527k 101.1783k 47.73672k 46.72987k
running_shoes_m_NF 270.7918k 146.4591k 153.4526k 275.6137k 119.9608k 135.7284k 278.1822k 133.1083k 152.2267k
shirt_f_F 7,806.35 5,900.01 5,981.13 7,726.19 5,765.66 5,815.25 7,890.88 5,992.66 5,864.81
shirt_f_NF 40.29641k 22.22263k 22.61778k 37.46539k 21.23242k 21.12642k 40.3249k 21.95318k 21.61071k
shirt_m_F 22.31104k 9,290.36 9,256.36 23.42684k 9,297.88 8,987.78 22.64789k 12.49402k 11.61165k
shirt_m_NF 57.25235k 39.96592k 40.5419k 58.38055k 35.68972k 37.24147k 57.55258k 43.7687k 41.71966k
shorts_f_F 7,441.47 4,767.61 4,715.98 7,647.18 5,506.89 5,466.2 7,591.06 6,492.74 6,382.48
shorts_f_NF 33.35124k 32.71399k 32.81542k 32.97619k 28.80048k 28.8932k 35.2585k 27.35361k 27.61106k
shorts_m_F 3,786.47 2,441.41 2,409.36 3,807.95 2,763.1 2,699.1 3,605.31 3,331.28 3,176.02
shorts_m_NF 16.46768k 11.53719k 11.5882k 15.18569k 11.05882k 11.06825k 16.3936k 11.27634k 11.1429k
sneakers_f_F 8,244.42 4,200.51 4,182.14 8,590.65 3,948.09 3,891.5 8,160.15 3,980.49 3,916.75
sneakers_f_NF 43.82827k 32.55464k 31.33334k 39.85756k 24.15977k 23.36846k 40.38245k 22.66372k 22.43985k
sneakers_m_F 15.26094k 4,997.62 5,192.14 16.02121k 5,718.17 5,766.59 15.89395k 7,167.05 7,158.94
sneakers_m_NF 53.27522k 33.57914k 33.22265k 50.03953k 24.86147k 24.77521k 47.29293k 22.71787k 23.78127k
sweatpants_f_F 1,212.3 885.68 911.55 1,176.58 832.55 851.08 1,139.64 885.7 876.97
sweatpants_f_NF 3,694.08 2,673.2 2,756.75 3,416.8 2,319.63 2,324.76 3,564.21 2,124.38 2,207.02
sweatpants_m_F 469.77 376.27 370.83 477.07 374.89 373.1 477.78 392.9 399
sweatpants_m_NF 2,687.77 2,105.67 2,089.3 2,578.83 1,874.42 1,853.72 2,588.87 1,732.15 1,699.85
sweatshirt_f_F 4,273.5 2,106.85 2,081.75 4,285.52 2,237.7 2,227.82 4,125.36 2,678.5 2,660.59
sweatshirt_f_NF 8,646.23 7,784.46 7,931.54 8,749.42 5,851.43 6,328.5 8,478.38 5,785.46 6,111.39
sweatshirt_m_F 4,274.85 2,850.86 2,831.31 4,283.48 2,769.01 2,756.64 4,301.19 2,714.32 2,700.38
sweatshirt_m_NF 11.58613k 7,946.76 7,914.73 11.40474k 7,660.7 7,611.41 11.42147k 7,691.2 7,628.95
underwear_F 0.04 16.34 13.19 0.29 60.29 37.64 0.04 104.5 68.86
underwear_NF 1.46 15.14 14.15 1.54 19.17 21.43 5.18 23.87 29.86
arimax_DNN arimax_bu arimax_minT ens_double_DNN ens_double_bu ens_double_minT tbat_nn_DNN tbat_nn_bu tbat_nn_minT
250k
200k
150k
100k
50k
0

*Figure 6.12*

Logarithm of the mean weighted MAE for the bottom levels post reconciliation

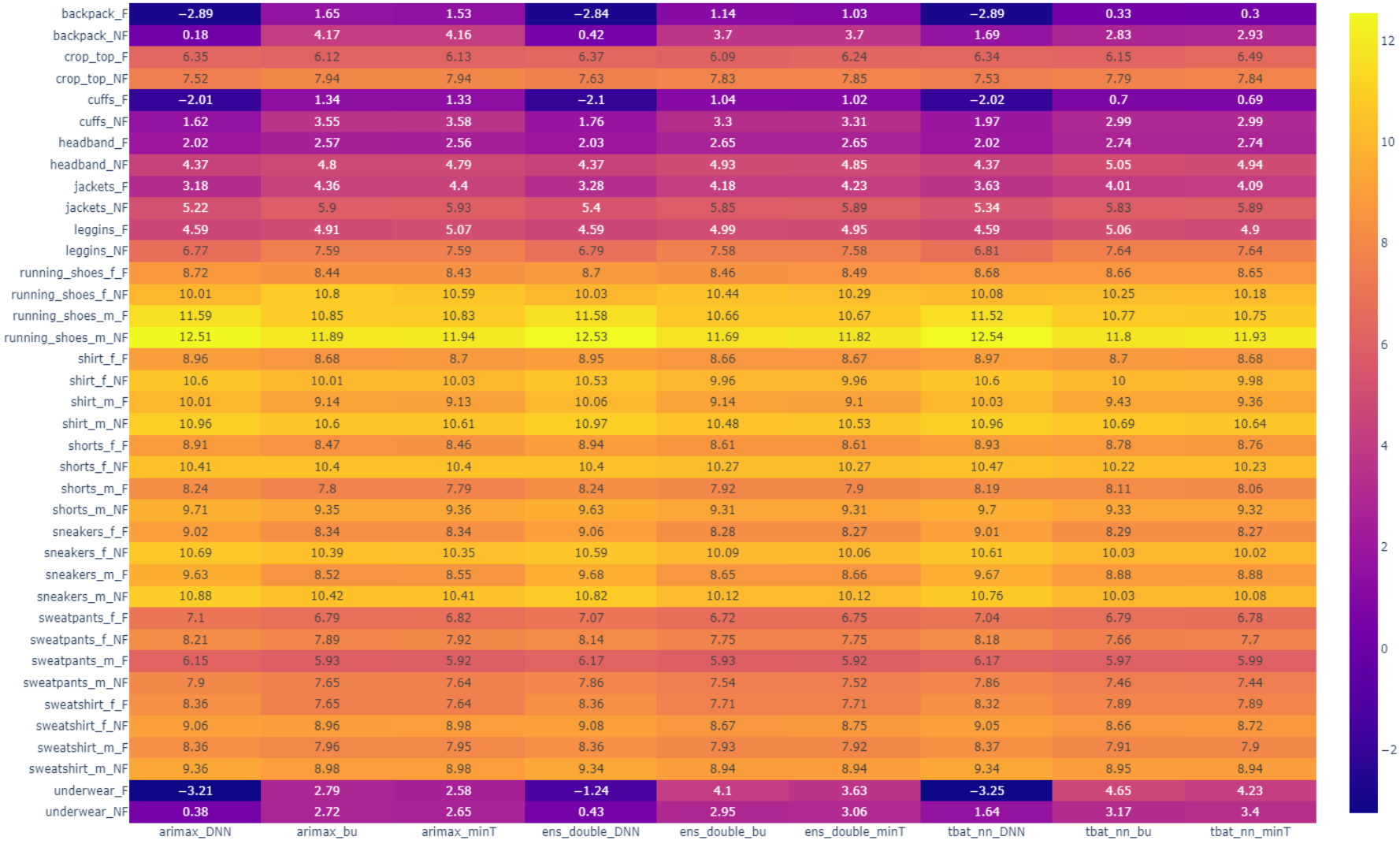


*Figure 6.13*

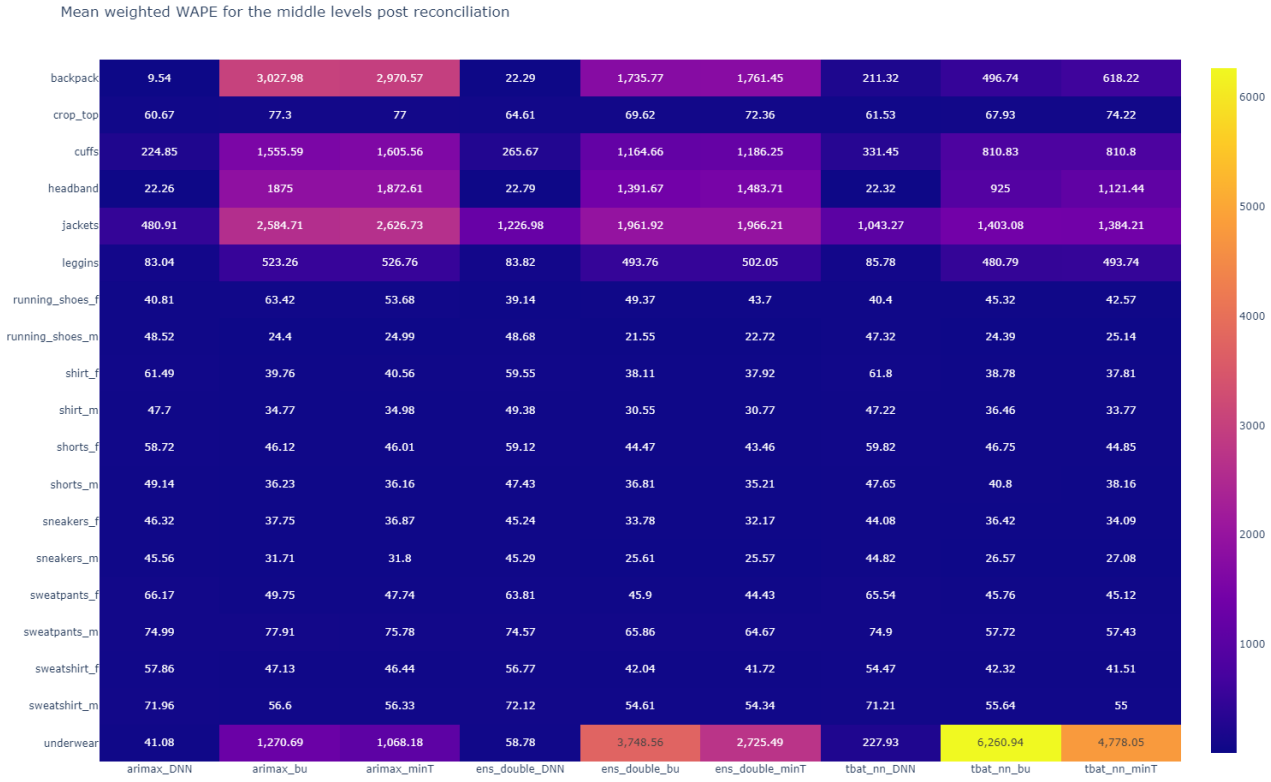


*Figure 6.14*

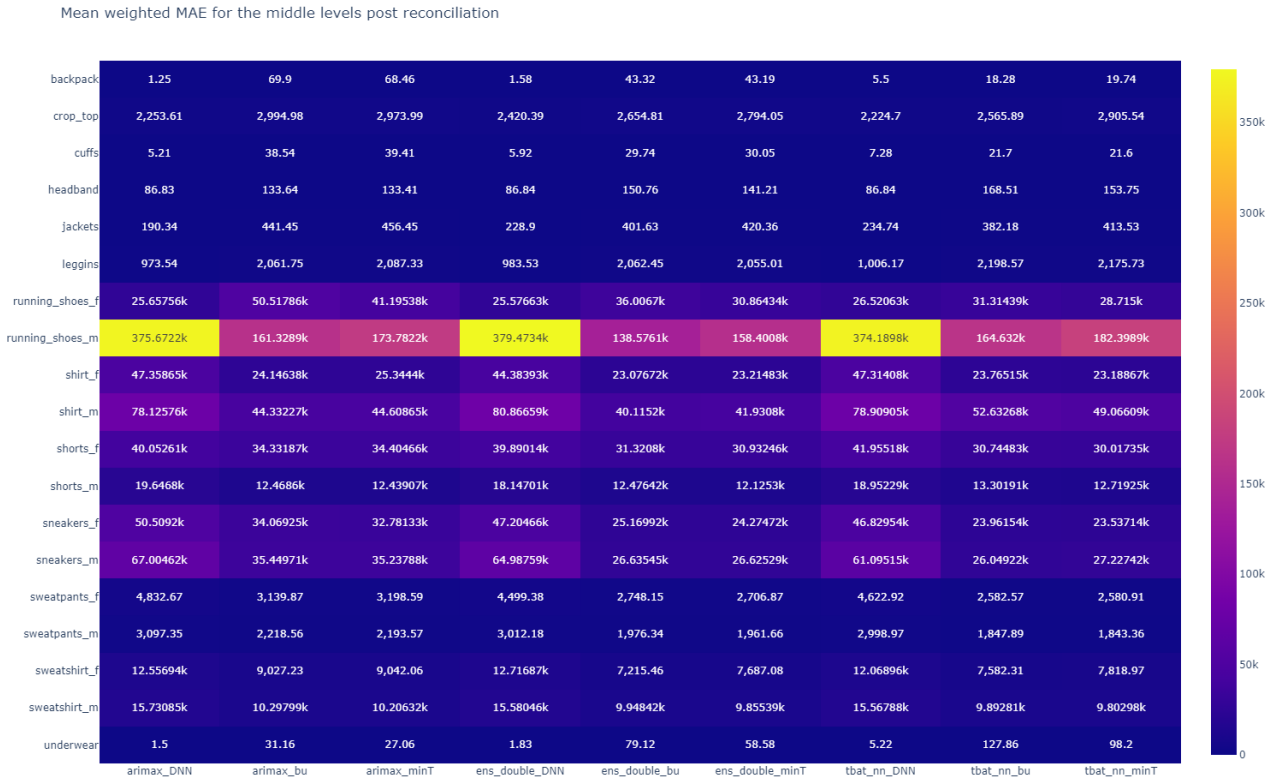


*Figure 6.15*

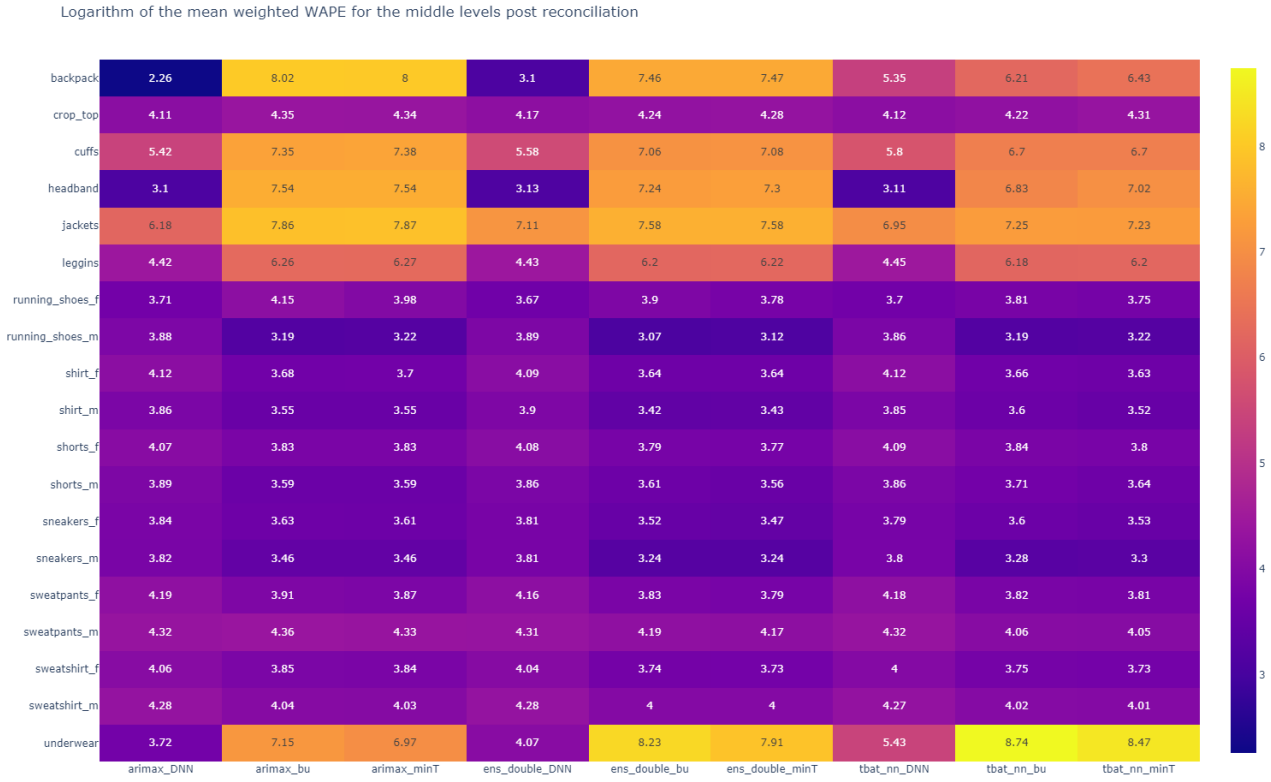


*Figure 6.16*

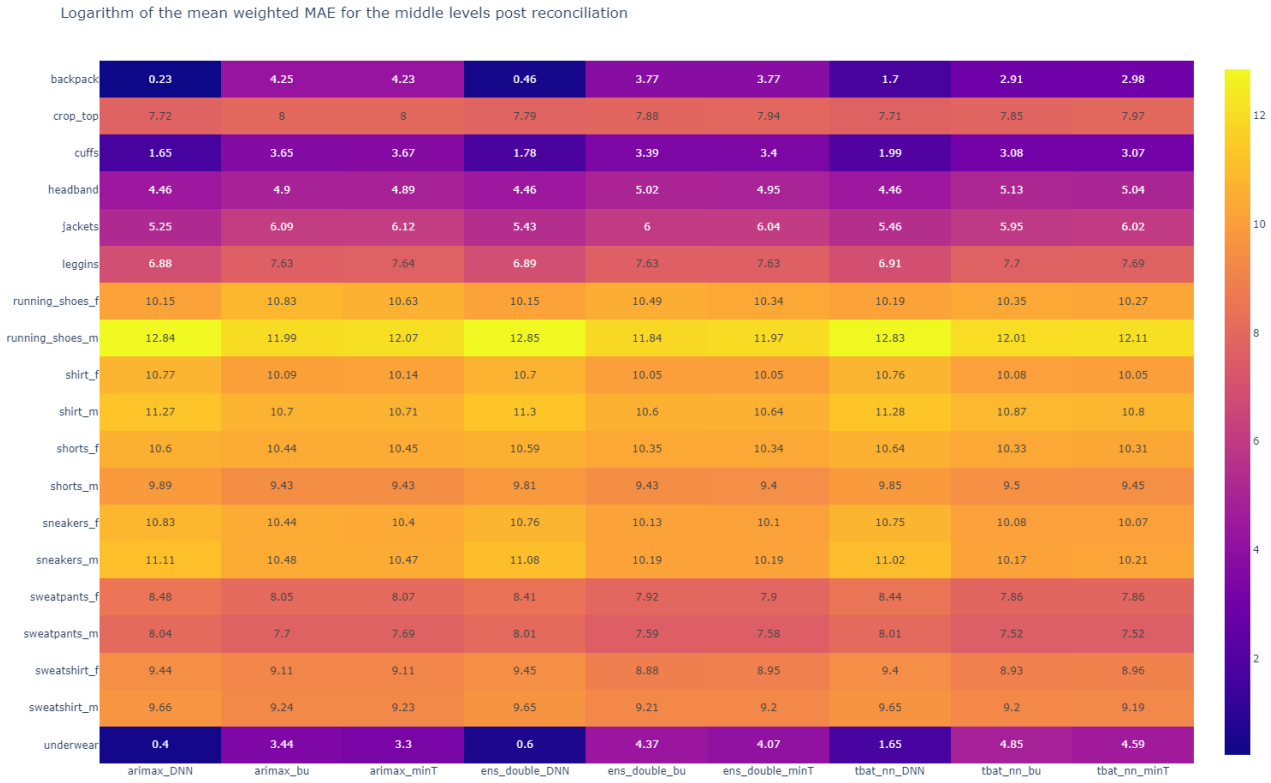


*Figure 6.17*

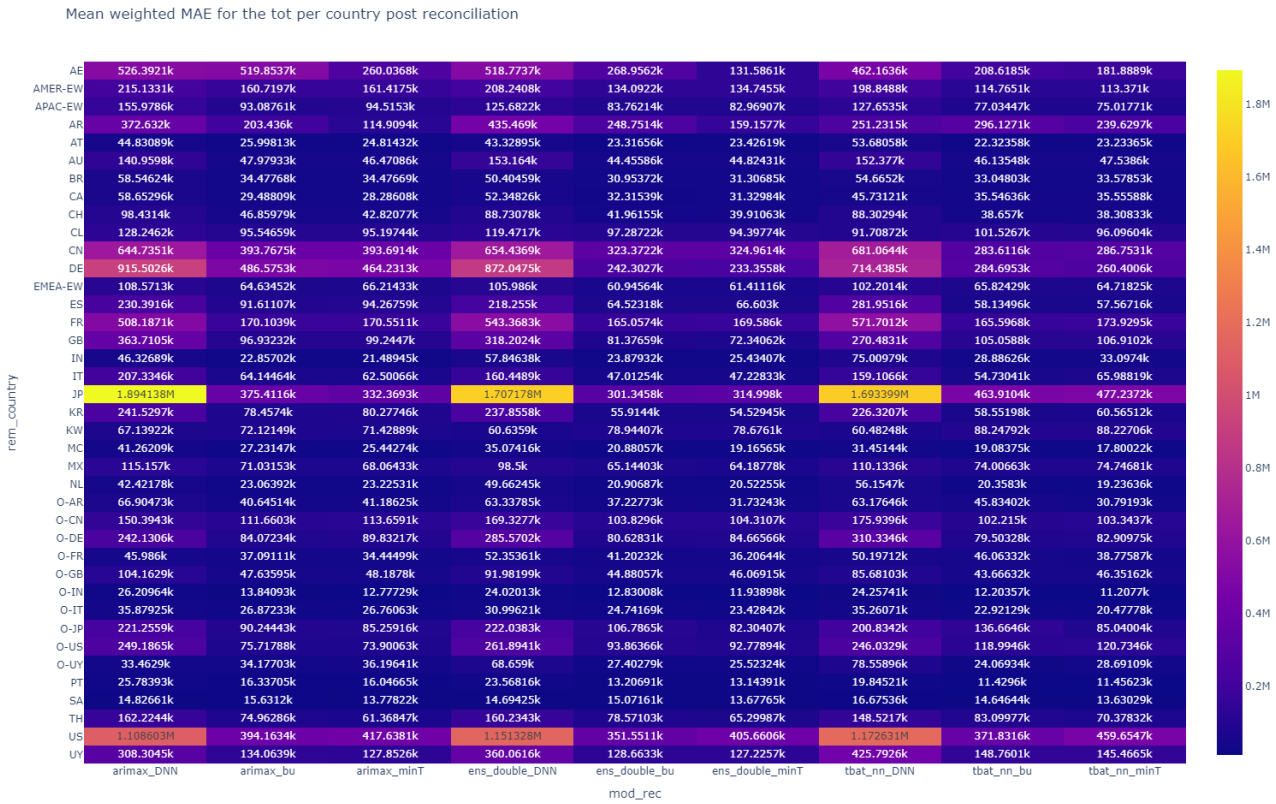


*Figure 6.18*

## 6.2 CODE

In all the following code, the length of the test set is hard-coded by taking the last 52 elements of the dataset. To make the program more flexible, a new variable could be added to the function related to how much of the data should be used for the testing part, and the substitute al the 52 with the new variable.

### 6.2.1 AUXILIARY CODE

```
def get_all_levels(original_df, country):

    """
    Starting with the whole dataframe which only has the lowest aggregation levels
of the HTS, return a DF with all the aggregation levels per date

    Arguments:\\
    original_df -- DF with the lowest level of aggregation, the values should be
ordered nicely in the rows to avoid errors\\
    country -- country of which we want the various aggregation levels

    Returns:\\
```

```
    results_df -- DF with for columns all the various aggregation levels properly
labelled and the cal_date of each
    """

    results = []
    country_df = original_df.loc[original_df['country'] == country].copy()
    country_df.reset_index(drop = True, inplace = True)
    agg_levels = ['tot']                # highest level is the tot of the country

    for i in range(int(country_df.shape[0]/38)):           # since we have 38
series at the bottom levels
        new_row = agg_matrix.dot(country_df['amt_euro'][i*38: i*38 + 38])
        results.append(new_row)

    ex_df = country_df[:][:38]                             # get a mock example for
a date to use to get the various levels names
    for j in range(19):                                     # get the levels name
of country + department code
        agg_levels.append('{0}'.format(ex_df['rem_product'][j*2]))
    for j in range(38):                                     # get the levels names
of country + department code + mark down code
        agg_levels.append('{0}_{1}'.format(ex_df['rem_product'][j],
ex_df['clientele_area'][j]))

    results_df = pd.DataFrame(results, columns = agg_levels)
    dates = sorted(list(set(country_df['cal_date'])))
    results_df['cal_date'] = dates
    results_df = results_df[['cal_date'] + list(results_df.columns)[:-1]]
    return results_df

def get_ex_df(original_df, country):

    """
    Starting from the whole dataframe, return a filtered one with the exogenous
variables of the chosen object code for a specific country

    Arguments:\\
    original_df -- starting DF\\
    country -- country which we want the exogenous of\\

    Returns:\\
    ex_var -- exogenous variables DF of the country chosen.
    """

    reduced_df = original_df.loc[original_df['country'] ==
country].loc[original_df['rem_product'] ==
'shirt_m'].loc[original_df['clientele_area'] == 'NF'].copy()
    ex_var = reduced_df[reduced_df.columns[-32:]].copy()
    ex_var.reset_index(drop = True, inplace = True)
```

```
    for i in ex_var.columns: # eliminate all the constant columns, which for sure
won't improve at all the predictions
        if (ex_var[i][:-52] == ex_var[i][0]).all():
            del ex_var[i]
    return ex_var
```

*Code 6.1*

### 6.2.2 ARIMAX CODE

```
def fit_arimax_single(data, data_ex, level, country, metrics_df, results_tr = None,
results_test = None, store_fit = False):

    tr = data[:-52]
    val = data[-52:]
    len_val = len(val)
    tr_ex, test_ex = data_ex[:-52], data_ex[-52:]

    lr = LinearRegression()
    lr.fit(tr_ex, tr)
    mask = SequentialFeatureSelector(lr,  direction="backward").fit(tr_ex, tr)
    lr_final = LinearRegression()
    lr_final.fit(tr_ex[np.array(tr_ex.columns)[mask.get_support()]], tr)

    tr_lin_fit =
lr_final.predict(tr_ex[np.array(tr_ex.columns)[mask.get_support()]])
    test_lin_fit =
lr_final.predict(test_ex[np.array(test_ex.columns)[mask.get_support()]])

    metrics_df.append([country, level, 'linear', 'tr', 'mape', np.mean(np.abs((tr -
tr_lin_fit) / tr)) * 100])
    metrics_df.append([country, level, 'linear', 'tr', 'mae',
np.mean(np.abs(tr_lin_fit - tr))])
    metrics_df.append([country, level, 'linear', 'tr', 'wape',
(np.sum(np.abs(tr_lin_fit - tr)) / np.sum(tr)) * 100])
    metrics_df.append([country, level, 'linear', 'test', 'mape',
np.mean(np.abs((val - test_lin_fit) / val)) * 100])
    metrics_df.append([country, level, 'linear', 'test', 'mae',
np.mean(np.abs(test_lin_fit - val))])
    metrics_df.append([country, level, 'linear', 'test', 'wape',
(np.sum(np.abs(test_lin_fit - val)) / np.sum(val)) * 100])

    arima_res = auto_arima(tr - tr_lin_fit, start_p = 0, start_q = 0, max_p = 1,
max_d = 1, max_q = 1,
                           start_P = 0, start_Q = 0, max_P = 1, max_D = 1, max_Q =
1, m = 52, seasonal = True,
                           error_action = 'warn', trace = True, stepwise = True,
n_fits = 50)
    try:
```

```
        tr_arimax_fit =  tr_lin_fit + arima_res.fittedvalues().values
        test_arimax_fit = test_lin_fit +
arima_res.predict(len_val).values
    except ValueError:
        tr_arimax_fit = tr_lin_fit
        test_arimax_fit = test_lin_fit

    tr_arimax_fit = np.where(tr_arimax_fit < 0, 0, tr_arimax_fit)
    test_arimax_fit = np.where(test_arimax_fit < 0, 0, test_arimax_fit)

    metrics_df.append([country, level, 'arimax', 'tr', 'mape', np.mean(np.abs((tr -
tr_arimax_fit) / tr)) * 100])
    metrics_df.append([country, level, 'arimax', 'tr', 'mae',
np.mean(np.abs(tr_arimax_fit - tr))])
    metrics_df.append([country, level, 'arimax', 'tr', 'wape',
(np.sum(np.abs(tr_arimax_fit - tr)) / np.sum(tr)) * 100])
    metrics_df.append([country, level, 'arimax', 'test', 'mape',
np.mean(np.abs((val - test_arimax_fit) / val)) * 100])
    metrics_df.append([country, level, 'arimax', 'test', 'mae',
np.mean(np.abs(test_arimax_fit - val))])
    metrics_df.append([country, level, 'arimax', 'test', 'wape',
(np.sum(np.abs(test_arimax_fit - val)) / np.sum(val)) * 100])

    if store_fit:
        results_tr['lin_{0}_{1}'.format(country,level)] = tr_lin_fit
        results_test['lin_{0}_{1}'.format(country,level)] = test_lin_fit
        results_tr['ari_{0}_{1}'.format(country,level)] = tr_arimax_fit
        results_test['ari_{0}_{1}'.format(country,level)] = test_arimax_fit

def fit_arimax_paral(original_df, country, metrics_df, results_tr = None,
results_test = None, store_fit = False):

    data = get_all_levels(original_df, country)
    ex = get_ex_df(original_df, country)
    threads = {}
    n_cpu = mp.cpu_count()
    used_cpu = 0

    if store_fit:
        for level in data.columns:
            if level != 'cal_date' and used_cpu < n_cpu:                    # start
a number of thread which is at most the total cpus available -1
                threads['th_{0}'.format(level)] = threading.Thread(target =
fit_arimax_single, args = (data[level], ex, level, country, metrics_df, results_tr,
results_test, True))
                threads['th_{0}'.format(level)].start()
                used_cpu += 1
```

```
            if used_cpu == n_cpu - 1 or level == data.columns[-1]:          # join
the started threads if all the cpu available is used or if we are at the end of the
columns
                for key in threads.keys():
                    threads[key].join()
                threads = {}
                used_cpu = 0
    else:
        for level in data.columns:
            if level != 'cal_date' and used_cpu < n_cpu:                    # start
a number of thread which is at most the total cpus available -1
                threads['th_{0}'.format(level)] = threading.Thread(target =
fit_arimax_single, args = (data[level], ex, level, country, metrics_df))
                threads['th_{0}'.format(level)].start()
                used_cpu += 1
            if used_cpu == n_cpu - 1 or level == data.columns[-1]:          # join
the started threads if all the cpu available is used or if we are at the end of the
columns
                for key in threads.keys():
                    threads[key].join()
                threads = {}
                used_cpu = 0
```

*Code 6.2*

### 6.2.3 ETS CODE

```
def fit_ets_single(original_df, country, metrics_df, results_tr = None,
results_test = None, store_fit = False):

    country_data = get_all_levels(original_df, country)

    ets_results = pd.DataFrame(columns = country_data.columns)
    ets_results['cal_date'] = country_data['cal_date']
    for level in ets_results.columns:
        if level != 'cal_date':
            model = ETSModel(country_data[level][:-52], error="add", trend="add",
seasonal="add", damped_trend=True, seasonal_periods=52)
            fit = model.fit()
            ets_results.loc[ets_results['cal_date'] <= '2023-04-09', level] =
fit.fittedvalues.astype(np.float64)
            ets_results.loc[ets_results['cal_date'] > '2023-04-09', level] =
fit.forecast(52).astype(np.float64)
    ets_results[ets_results.columns[1:]] =
ets_results[ets_results.columns[1:]].astype(np.float64)
    ets_results[ets_results[ets_results.columns[1:]] < 0] = 0
    for level in ets_results.columns[1:]:
        if store_fit:
```

```
            results_tr['ets_{0}_{1}'.format(country, level)] = ets_results.iloc[:-
52][level].values
            results_test['ets_{0}_{1}'.format(country, level)] = ets_results.iloc[-
52:][level].values
        metrics_df.append([country, level, 'ets', 'tr', 'mape',
np.mean(np.abs((country_data[level][:-52].values - ets_results.iloc[:-
52][level].values) / country_data[level][:-52].values)) * 100])
        metrics_df.append([country, level, 'ets', 'tr', 'mae',
np.mean(np.abs(ets_results.iloc[:-52][level].values - country_data[level][:-
52].values))])
        metrics_df.append([country, level, 'ets', 'tr', 'wape',
(np.sum(np.abs(ets_results.iloc[:-52][level].values - country_data[level][:-
52].values)) / np.sum(country_data[level][:-52].values)) * 100])
        metrics_df.append([country, level, 'ets', 'test', 'mape',
np.mean(np.abs((country_data[level][-52:].values - ets_results.iloc[-
52:][level].values) / country_data[level][-52:].values)) * 100])
        metrics_df.append([country, level, 'ets', 'test', 'mae',
np.mean(np.abs(ets_results.iloc[-52:][level].values - country_data[level][-
52:].values))])
        metrics_df.append([country, level, 'ets', 'test', 'wape',
(np.sum(np.abs(ets_results.iloc[-52:][level].values - country_data[level][-
52:].values)) / np.sum(country_data[level][-52:].values)) * 100])

def fit_ets_paral(original_df, metrics_df, results_tr = None, results_test = None,
store_fit = False):

    country_list = list(set(original_df['country']))
    threads = {}
    n_cpu = mp.cpu_count()
    used_cpu = 0

    if store_fit:
        for country in country_list:
            if used_cpu < n_cpu:                    # start a number of thread
which is at most the total cpus available -1
                threads['th_{0}'.format(country)] = threading.Thread(target =
fit_ets_single, args = (original_df, country, metrics_df, results_tr, results_test,
True))
                threads['th_{0}'.format(country)].start()
                used_cpu += 1
            if used_cpu == n_cpu - 1 or country == country_list[-1]:          #
join the started threads if all the cpu available is used or if we are at the end
of the columns
                for key in threads.keys():
                    threads[key].join()
                threads = {}
                used_cpu = 0
    else:
        for country in country_list:
```

```
            if used_cpu < n_cpu:                    # start a number of thread
which is at most the total cpus available -1
                threads['th_{0}'.format(country)] = threading.Thread(target =
fit_ets_single, args = (original_df, country, metrics_df))
                threads['th_{0}'.format(country)].start()
                used_cpu += 1
            if used_cpu == n_cpu - 1 or country == country_list[-1]:          #
join the started threads if all the cpu available is used or if we are at the end
of the columns
                for key in threads.keys():
                    threads[key].join()
                threads = {}
                used_cpu = 0
```

*Code 6.3*

### 6.2.4 TBATS CODE

```
def fit_tbat_single(data, level, country, metrics_df, results_tr = None,
results_test = None, store_fit = False):

    print('Estimating TBAT for {0} in {1}'.format(level, country))
    estimator = TBATS(seasonal_periods=[52.14])
    fitted_model = estimator.fit(data[:-52])
    tbat_tr_results = fitted_model.y_hat.astype(np.float64)
    tbat_test_results = fitted_model.forecast(52).astype(np.float64)
    tbat_tr_results[tbat_tr_results < 0] = 0
    tbat_test_results[tbat_test_results < 0] = 0
    if store_fit:
        results_tr['tbat_{0}_{1}'.format(country, level)] = tbat_tr_results
        results_test['tbat_{0}_{1}'.format(country, level)] = tbat_test_results
    metrics_df.append([country, level, 'tbat', 'tr', 'mape',
np.mean(np.abs((data[:-52].values - tbat_tr_results) / data[:-52].values)) * 100])
    metrics_df.append([country, level, 'tbat', 'tr', 'mae',
np.mean(np.abs(tbat_tr_results - data[:-52].values))])
    metrics_df.append([country, level, 'tbat', 'tr', 'wape',
(np.sum(np.abs(tbat_tr_results - data[:-52].values)) / np.sum(data[:-52].values)) *
100])
    metrics_df.append([country, level, 'tbat', 'test', 'mape',
np.mean(np.abs((data[-52:].values - tbat_test_results) / data[-52:].values)) *
100])
    metrics_df.append([country, level, 'tbat', 'test', 'mae',
np.mean(np.abs(tbat_test_results - data[-52:].values))])
    metrics_df.append([country, level, 'tbat', 'test', 'wape',
(np.sum(np.abs(tbat_test_results - data[-52:].values)) / np.sum(data[-52:].values))
* 100])

def fit_tbat_paral(original_df, country, metrics_df, results_tr = None,
results_test = None, store_fit = False):
```

```
    data = get_all_levels(original_df, country)
    threads = {}
    n_cpu = mp.cpu_count()
    used_cpu = 0

    if store_fit:
        for level in data.columns:
            if level != 'cal_date' and used_cpu < n_cpu:                    # start
a number of thread which is at most the total cpus available -1
                threads['th_{0}'.format(level)] = threading.Thread(target =
fit_tbat_single, args = (data[level], level, country, metrics_df, results_tr,
results_test, True))
                threads['th_{0}'.format(level)].start()
                used_cpu += 1
            if used_cpu == n_cpu - 1 or level == data.columns[-1]:          # join
the started threads if all the cpu available is used or if we are at the end of the
columns
                for key in threads.keys():
                    threads[key].join()
                threads = {}
                used_cpu = 0
    else:
        for level in data.columns:
            if level != 'cal_date' and used_cpu < n_cpu:                    # start
a number of thread which is at most the total cpus available -1
                threads['th_{0}'.format(level)] = threading.Thread(target =
fit_tbat_single, args = (data[level], level, country, metrics_df))
                threads['th_{0}'.format(level)].start()
                used_cpu += 1
            if used_cpu == n_cpu - 1 or level == data.columns[-1]:          # join
the started threads if all the cpu available is used or if we are at the end of the
columns
                for key in threads.keys():
                    threads[key].join()
                threads = {}
                used_cpu = 0
```

*Code 6.4*

### 6.2.5 TBATS CODE

```
def fit_nn_4_residuals(original_df, country, model, model_tr_fit, model_test_fit,
metrics_df, results_tr = None, results_test = None, store_fit = False):

    country_data = get_all_levels(original_df, country)
    country_ex = get_ex_df(original_df, country)
    num_ex = len(country_ex.columns)
    model_fit = pd.DataFrame(columns = country_data.columns[1:])
```

```
    for level in country_data.columns[1:]:
        model_fit[level] = np.concatenate([model_tr_fit['{0}_{1}_{2}'.format(model,
country, level)], model_test_fit['{0}_{1}_{2}'.format(model, country, level)]])
    model_residuals = country_data[country_data.columns[1:]][:-52] -
model_fit[country_data.columns[1:]][:-52]
    scaler = MinMaxScaler((0,1))
    scaler.fit(model_residuals)
    ex_scaler = MinMaxScaler((0,1))
    ex_scaler.fit(country_ex.iloc[:-52])
    sc_model_residuals = scaler.transform(model_residuals)
    sc_country_ex = pd.DataFrame(ex_scaler.transform(country_ex))

    #create the NN for the residual, using 1 timesteps at the time
    data_points = sc_model_residuals.shape[0]
    tr_set = sc_model_residuals.reshape((data_points,1,58))
    auto_mlp = tf.keras.Sequential([tf.keras.Input((1,58)),
tf.keras.layers.Flatten(), tf.keras.layers.Dense(58, activation = 'sigmoid')])
    mlp = tf.keras.Sequential([tf.keras.Input((1,num_ex)),
tf.keras.layers.Flatten(), tf.keras.layers.Dense(num_ex, activation = 'sigmoid')])
    inp_1 = tf.keras.Input((1,58))
    code = auto_mlp(inp_1)
    inp_2 = tf.keras.Input((1,num_ex))
    exog = mlp(inp_2)
    merge = keras.layers.concatenate([code, exog], axis = 1)
    activate = tf.keras.layers.Dense(58 + num_ex, activation = 'sigmoid')(merge)
    output = tf.keras.layers.Dense(58, activation = 'linear')(activate)
    final_nn = tf.keras.Model([inp_1, inp_2], output)
    final_nn.compile(optimizer = 'adam', loss = 'mean_squared_error')
    final_nn.summary()
    final_nn.fit([tr_set[:-1],
                  sc_country_ex.iloc[1:-52].values.reshape((data_points-
1,1,num_ex))],
                  sc_model_residuals[1:].reshape((data_points-1,1,58)),
                  epochs = 50, batch_size = 1, verbose = 2)
    nn_tr_results =
pd.DataFrame(scaler.inverse_transform(pd.DataFrame(final_nn.predict([tr_set[:-1],

     sc_country_ex.iloc[1:-52].values.reshape((data_points-1,1,num_ex))]))),

     columns = country_data.columns[1:], index = range(1,149))
    tr_tbat_nn_results = nn_tr_results +
model_fit[country_data.columns[1:]].iloc[1:-52]
    tr_tbat_nn_results[tr_tbat_nn_results < 0] = 0
    for level in tr_tbat_nn_results.columns:
        if store_fit:
            results_tr['nn_{0}_{1}_{2}'.format(model,country,level)] =
np.append(np.array([np.nan]), tr_tbat_nn_results[level].values)
```

```
        metrics_df.append([country, level, 'nn_{0}'.format(model), 'tr', 'mape',
np.mean(np.abs((country_data[level][1:-52].values -
tr_tbat_nn_results[level].values) / country_data[level][1:-52].values)) * 100])
        metrics_df.append([country, level, 'nn_{0}'.format(model), 'tr', 'mae',
np.mean(np.abs(tr_tbat_nn_results[level].values - country_data[level][1:-
52].values))])
        metrics_df.append([country, level, 'nn_{0}'.format(model), 'tr', 'wape',
(np.sum(np.abs(tr_tbat_nn_results[level].values - country_data[level][1:-
52].values)) / np.sum(country_data[level][1:-52].values)) * 100])
    test_results = np.array([])
    test_results = np.append(test_results, final_nn.predict([tr_set[-
1].reshape((1,1,58)),
                                                             sc_country_ex.iloc[-
52].values.reshape((1,1,num_ex))]))
    for i in range(1,52):
        test_results = np.append(test_results,
                                 final_nn.predict([test_results[-
58:].reshape((1,1,58)),
                                                   sc_country_ex.iloc[-
52+i].values.reshape((1,1,num_ex))]))
    test_nn_results =
pd.DataFrame(scaler.inverse_transform(pd.DataFrame(test_results.reshape(52,58))),
                                     columns = country_data.columns[1:], index =
range(149,149+52))
    test_tbat_nn_results = test_nn_results +
model_fit[country_data.columns[1:]].iloc[-52:]
    test_tbat_nn_results[test_tbat_nn_results < 0] = 0
    for level in test_tbat_nn_results.columns:
        if store_fit:
            results_test['nn_{0}_{1}_{2}'.format(model,country,level)] =
test_tbat_nn_results[level].values
        metrics_df.append([country, level, 'nn_{0}'.format(model), 'test', 'mape',
np.mean(np.abs((country_data[level][-52:].values -
test_tbat_nn_results[level].values) / country_data[level][-52:].values)) * 100])
        metrics_df.append([country, level, 'nn_{0}'.format(model), 'test', 'mae',
np.mean(np.abs(test_tbat_nn_results[level].values - country_data[level][-
52:].values))])
        metrics_df.append([country, level, 'nn_{0}'.format(model), 'test', 'wape',
(np.sum(np.abs(test_tbat_nn_results[level].values - country_data[level][-
52:].values)) / np.sum(country_data[level][-52:].values)) * 100])

def fit_nn_4_residuals_paral(original_df, model, model_tr_fit, model_test_fit,
metrics_df, results_tr = None, results_test = None, store_fit = False):

    country_list = list(set(original_df['country']))
    threads = {}
    n_cpu = mp.cpu_count()
    used_cpu = 0
```

```
    if store_fit:
        for country in country_list:
            if used_cpu < n_cpu:                    # start a number of thread
which is at most the total cpus available -1
                threads['th_{0}'.format(country)] = threading.Thread(target =
fit_nn_4_residuals, args = (original_df, country, model, model_tr_fit,
model_test_fit, metrics_df, results_tr, results_test, True))
                threads['th_{0}'.format(country)].start()
                used_cpu += 1
            if used_cpu == n_cpu - 1 or country == country_list[-1]:          #
join the started threads if all the cpu available is used or if we are at the end
of the columns
                for key in threads.keys():
                    threads[key].join()
                threads = {}
                used_cpu = 0
    else:
        for country in country_list:
            if used_cpu < n_cpu:                    # start a number of thread
which is at most the total cpus available -1
                threads['th_{0}'.format(country)] = threading.Thread(target =
fit_nn_4_residuals, args = (original_df, country, model, model_tr_fit,
model_test_fit, metrics_df))
                threads['th_{0}'.format(country)].start()
                used_cpu += 1
            if used_cpu == n_cpu - 1 or country == country_list[-1]:          #
join the started threads if all the cpu available is used or if we are at the end
of the columns
                for key in threads.keys():
                    threads[key].join()
                threads = {}
                used_cpu = 0
```

*Code 6.5*

### 6.2.6 BOTTOM-UP AND MINT RECONCILIATION CODE

```
def get_all_levels_df(original_df, country):
    results = []
    country_df = original_df.loc[original_df['country'] == country].copy()
    country_df.reset_index(drop = True, inplace = True)
    agg_levels = ['{0}_tot'.format(country)]

    for i in range(int(country_df.shape[0] / 38)):
        new_row = agg_matrix.dot(country_df['amt_euro'][i*38: i*38 + 38])
        results.append(new_row)

    ex_df = country_df[:38]
    for j in range(19):
```

```
        agg_levels.append('{0}_{1}'.format(country, ex_df['rem_product'][j*2]))
    for j in range(38):
        agg_levels.append('{0}_{1}_{2}'.format(country, ex_df['rem_product'][j],
ex_df['clientele_area'][j]))


    agg_levels_rows = np.array(results).T
    dates = sorted(list(set(original_df['cal_date'])))[:-52]
    label = np.repeat(agg_levels[0], len(dates))
    final_df = pd.DataFrame({'unique_id':label, 'ds':dates,
'y':agg_levels_rows[0][:-52]})
    for i in range(1, len(agg_levels)):
        label = np.repeat(agg_levels[i], len(dates))
        new_df = pd.DataFrame({'unique_id':label, 'ds':dates,
'y':agg_levels_rows[i][:-52]})
        final_df = pd.concat([final_df, new_df], axis = 0)
    #final_df.set_index('unique_id', inplace = True)
    return final_df

def get_hierarchical_df(original_df, country, fit_results_tr, fit_results_test,
model, metrics_df):

    tr_agg_df = get_all_levels_df(original_df, country)         # used for the
training of the hierarchy reconciliation
    tr_agg_df.set_index('unique_id', inplace = True)
    comp_df = get_all_levels(original_df, country)              # complementary df,
has many useful infos that will be used
    cc = [country + '_' + a for a in comp_df.columns[1:]]
    tr_agg_df[model] = pd.melt(fit_results_tr[cc])['value'].values
    tr_agg_df[['y', model]] += np.random.rand(8642,2)          # add random noise
to avoid constant columns, needed for the MinTrace method

    fit_results_dated = fit_results_test.copy()
    fit_results_dated['cal_date'] = get_all_levels(original_df,
country)['cal_date'][-52:].values
    # we need the test fit of all the values in all the aggregation levels from a
model in a DF with specific labels in the columns
    fit_df = pd.melt(fit_results_dated[cc + ['cal_date']],
'cal_date').set_axis(['ds', 'unique_id', model], axis = 'columns')
    fit_df.set_index('unique_id', inplace = True)

    # define the tags of the various levels of aggregation so the model knows how
to handle them
    tags = {'country':[], 'country_prod':[], 'country_prod_cl':[]}
    for level in fit_results_test.columns:
        if level[:len(country)] == country:
            if len(level) == len(country) + 4:
                tags['country'].append(level)
            elif level[-1] == 'F':
```

```
                tags['country_prod_cl'].append(level)
            else:
                tags['country_prod'].append(level)

    # the aggregation matrix need to be rewritten in a way that the columns and the
index of the rows refers to how the aggregation happens
    agg_mat_df = pd.DataFrame(agg_matrix, columns = cc[-38:])
    agg_mat_df.set_index(np.array(cc), inplace = True)
    #print(agg_mat_df)

    # time for the reconciliation
    #reconcilers = [BottomUp(), TopDown(method='forecast_proportions'),
MiddleOut(middle_level='country_prod', top_down_method='forecast_proportions'),
MinTrace('wls_var', nonnegative = True)]
    reconcilers = [BottomUp(), MinTrace('wls_var', nonnegative = True)]
    hrec = HierarchicalReconciliation(reconcilers=reconcilers)
    y_rec_df = hrec.reconcile(Y_hat_df = fit_df, Y_df = tr_agg_df, S = agg_mat_df,
tags = tags)
    y_rec_df = y_rec_df.set_axis(['cal_date', 'original_forecast', 'bu', 'minT'],
axis = 'columns')

    # store the test metrics in the dictionary
    for j in cc:
        level = j[len(country)+1:]
        metrics_df.append([country, level, model, 'bu', 'mape',
np.mean(np.abs((comp_df[level][-52:].values - y_rec_df.loc[j]['bu'].values) /
comp_df[level][-52:].values)) * 100])
        metrics_df.append([country, level, model, 'minT', 'mape',
np.mean(np.abs((comp_df[level][-52:].values - y_rec_df.loc[j]['minT'].values) /
comp_df[level][-52:].values)) * 100])
        metrics_df.append([country, level, model, 'bu', 'mae',
np.mean(np.abs(y_rec_df.loc[j]['bu'].values - comp_df[level][-52:].values))])
        metrics_df.append([country, level, model, 'minT', 'mae',
np.mean(np.abs(y_rec_df.loc[j]['minT'].values - comp_df[level][-52:].values))])
        metrics_df.append([country, level, model, 'bu', 'wape',
np.sum(np.abs(y_rec_df.loc[j]['bu'].values - comp_df[level][-52:].values)) /
np.sum(comp_df[level][-52:].values) * 100])
        metrics_df.append([country, level, model, 'minT', 'wape',
np.sum(np.abs(y_rec_df.loc[j]['minT'].values - comp_df[level][-52:].values)) /
np.sum(comp_df[level][-52:].values) * 100])

    # modify the DF to properly have the data
    y_rec_df.reset_index(inplace = True)
    y_rec_df['country'] = country
    y_rec_df['model'] = model
    if country[:2] != 'O-':
        y_rec_df['rem_country'] = remap_df.loc[remap_df['country'] == country,
'remap'].values[0]
    else:
```

```
        y_rec_df['rem_country'] = 'O-{0}'.format(remap_df.loc[remap_df['country']
== country[2:], 'remap'].values[0])
    y_rec_df['level'] = y_rec_df['unique_id'].apply(lambda x: x[len(country)+1:])
    del y_rec_df['unique_id']
    y_rec_df = y_rec_df[['cal_date', 'country', 'rem_country', 'level', 'model',
'original_forecast', 'bu', 'minT']]

    return y_rec_df
```

*Code 6.6*

### 6.2.7 NDD RECONCILIATION

```
def double_loss(y_true, y_pred):
    coer_part = keras.ops.sum(keras.ops.square(keras.ops.norm(y_true - y_pred, axis
= 0)))
    pred_part = keras.ops.sum(keras.ops.square(keras.ops.sum(y_true, axis = 1) -
keras.ops.sum(y_pred, axis = 1)))
    loss = ((0.75)*coer_part + (0.25)*pred_part)/y_true.shape[1]
    return loss

def decompose_dnn(original_df, country, model, model_tr_fit, model_test_fit,
metrics_df):
    country_data = get_all_levels(original_df, country)
    val_scaler = MinMaxScaler((0,1))
    val_scaler.fit(country_data[country_data.columns[1:]][:-52])
    sc_data =
pd.DataFrame(val_scaler.transform(country_data[country_data.columns[1:]]))
    tr_inp = sc_data[:-52][sc_data.columns[0]]          # here 0 is the tot column
    tr_set = np.array([])
    for i in range(tr_inp.shape[0] - 3):
        tr_set = np.append(tr_set, tr_inp[i:i+4])
    tr_set = tr_set.reshape((146,4,1))
    ex = get_ex_df(original_df, country)
    exog_scaler = MinMaxScaler((0,1))
    exog_scaler.fit(ex[:-52])
    ex_scaled = pd.DataFrame(exog_scaler.transform(ex[:-52]))
    ex_tr = ex_scaled[3:]
    ex_test = ex_scaled[-52:]
    num_ex = len(ex_tr.columns)

    encoder = tf.keras.Sequential([tf.keras.Input((4,1)),
                                   tf.keras.layers.Conv1D(filters = 32, kernel_size
= 2, padding = 'same', activation = 'relu'),
                                   tf.keras.layers.Conv1D(filters = 32, kernel_size
= 8, padding = 'same', activation = 'relu'),
                                   tf.keras.layers.Conv1D(filters = 32, kernel_size
= 8, padding = 'same', activation = 'relu'),
```

```
                                    tf.keras.layers.Conv1D(filters = 32, kernel_size
= 8, padding = 'same', activation = 'relu'),
                                    tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(12, activation = 'relu')])
    ddn_mlp = tf.keras.Sequential([tf.keras.Input((1,num_ex)),
                                    tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(32, activation = 'relu'),
                                    tf.keras.layers.Dense(32, activation = 'relu'),
                                    tf.keras.layers.Dense(32, activation = 'relu')])
    inp_1 = tf.keras.Input((4,1))
    code = encoder(inp_1)
    inp_2 = tf.keras.Input((1,num_ex))
    exog = ddn_mlp(inp_2)
    merge = keras.layers.concatenate([code, exog], axis = 1)
    activate = tf.keras.layers.Dense(38, activation = 'relu')(merge)
    output = tf.keras.layers.Dense(38, activation = 'linear')(activate)
    final_nn = tf.keras.Model([inp_1, inp_2], output)
    final_nn.compile(optimizer = 'adam', loss = double_loss)
    final_nn.summary()
    final_nn.fit([tr_set.reshape((146,4,1)), ex_tr.values.reshape((146,1,num_ex))],
sc_data[sc_data.columns[-38:]][3:-52].values.reshape(146,1,38), epochs = 100,
batch_size = 1, verbose = 2)

    test_set = np.array([])
    cols = ['{0}_{1}'.format(country, column) for column in
country_data.columns[1:]]
    test_df = pd.concat([model_tr_fit[cols][-3:].set_axis(country_data.columns[1:],
axis = 'columns'),
                         model_test_fit[cols].set_axis(country_data.columns[1:],
axis = 'columns')], ignore_index = True)
    sc_test_data = pd.DataFrame(val_scaler.transform(test_df))
    test_data = sc_test_data[0].values
    for i in range(52):
        test_set = np.append(test_set, test_data[i:i+4])
    test_set = test_set.reshape((52,4,1))
    nn_results = pd.DataFrame(final_nn.predict([test_set,
ex_test.values.reshape((52,1,num_ex))]), columns = country_data.columns[-38:])
    nn_results[nn_results < 0] = 0
    final_sc_results = pd.DataFrame(1., index = range(52), columns =
country_data.columns[1:])

    for level in nn_results.columns:
        final_sc_results[level] = nn_results[level]
    final_results = pd.DataFrame(val_scaler.inverse_transform(final_sc_results),
columns = final_sc_results.columns)
    for mid_level in middle_agg:
        final_results[mid_level] = final_results['{0}_F'.format(mid_level)] +
final_results['{0}_NF'.format(mid_level)]
```

```
    final_results['tot'] = np.sum(final_results[final_results.columns[-38:]], axis
= 1)


    for level in final_results.columns:
        metrics_df.append([country, level, model, 'DNN', 'mape',
np.mean(np.abs((country_data[level][-52:].values - final_results[level].values) /
country_data[level][-52:].values)) * 100])
        metrics_df.append([country, level, model, 'DNN', 'mae',
np.mean(np.abs(final_results[level].values - country_data[level][-52:].values))])
        metrics_df.append([country, level, model, 'DNN', 'wape',
np.sum(np.abs(final_results[level].values - country_data[level][-52:].values)) /
np.sum(country_data[level][-52:].values) * 100])

    final_results['cal_date'] = country_data['cal_date'][-52:].values

    return pd.melt(final_results, id_vars = 'cal_date', var_name = 'agg_level',
value_name = 'DNN')
```

*Code 6.7*

# ACKNOWLEDGEMENTS

This thesis would have not been possible without the presence of some really special people in my life, so I think it is only right to spend a few words to show my uttermost gratitude towards them.

The first person I would like to thank is Professor Guidolin for being my supervisor: she was always willing to answer my questions as soon as she could, and I was really excited to write this thesis on a subject that she taught me just a few months ago.

I would also like to thank the whole crew that helped me when I worked on this thesis at the Horsa Insight company: Gianluca, Marta, Mattia, Tommaso, Paolo, and all the others, without you all this work would not be here.

A special mention to all the people I met in the Data Science course with who I bonded the most: Pietro, Anna, Andrea, and Marco, the whole process would have not been the same without you all.

And now a token of gratitude to all the close people in my life who believed in me: Kristina, Mattia, Greta, and Miriam. You were all really good at pretending to understand what I was working on, but I appreciate it nonetheless and love you from the bottom of my heart, your support was priceless.

One last “thank you” to my family, without them I would not be here writing these acknowledgments right now.

# REFERENCES


Athanasopoulos, G., Roman, A. A. & Hyndman, R. J., 2009. Hierarchical forecasts for Australian domestic tourism. *International Journal of Forecasting,* pp. 146-166.

Dangerfield, B. J. & Morris, J. S., 1992. Top-down or bottom-up: Aggregate versus disaggregate extrapolations. *International Journal of Forecasting,* 8(2), pp. 233-241.

De Livera, A. M., Hyndman, R. J. & Snyder, R. D., 2011. Forecasting Time Series With Complex Seasonal Patterns Using Exponential Smoothing. *Journal of the American Statistical Association,* 106(496), pp. 1513-1527.

Goodfellow, I., Bengio, Y. & Courville, A., 2016. *Deep learning.* s.l.:s.n.

Gross, C. & Sohl, J., 1990. Dissagregation methods to expedite product line forecasting. *Journal of Forecasting,* pp. 233-254.

Hyndman, R., Ahmed, R., Athanasopoulos, G. & Lin Shang, H., 2011. Optimal combination forecasts for hierarchical time series. *Computational Statistics and Data Analysis,* pp. 2579-2589.

Hyndman, R. et al., 2024. *Forecast: Forecasting functions for time series and linear models.* [Online]
Available at: https://pkg.robjhyndman.com/forecast/
[Accessed 2024].

Hyndman, R. J. & Athanasopoulos, G., 2021. *Forecasting: Principles and Practice.* 3 ed. Melbourne, Australia: OTexts.

Hyndman, R. J. & Khandakar, Y., 2008. Automatic Time Series Forecasting: The forecast Package for R. *Journal of statistical software,* 27(3), pp. 1-22.

Inc., P. T., 2015. *Plotly Open Source Graphing Library for Python.* [Online]
Available at: https://plotly.com/python/
[Accessed 29 05 2024].

Karabiber, O. A. & Xydis, G., 2019. Electricity Price Forecasting in the Danish Day-Ahead. *Energies,* 12(928), pp. 1-29.

Mancuso, P., Piccialli, V. & Sudoso, A. M., 2021. A machine learning approach for forecasting hierarchical time series. *Expert Systems With Applications ,* Volume 182, pp. 1-17.

Marcilio, I., Hajat, S. & Gouveia, N., 2013. Forecasting Daily Emergency Department. *Official Journal of the Society for the Academic Emergency Medicine,* Volume 20, pp. 769-777.

Medina Maçaira, P., Tavares Thomé, A. M., Cyrino Oliveira, F. L. & Carvalho Ferrer, A. L., 2018. Time series analysis with explanatory variables: A systematic literature. *Environmental Modelling & Software,* Volume 107, pp. 199-209.

Nadeem, 2021. *Time Series Forecasting using TBATS Model.* [Online]
Available at: https://medium.com/analytics-vidhya/time-series-forecasting-using-tbats-model-ce8c429442a9
[Accessed 24 June 2024].

Olivares, K. G. et al., 2022. HierarchicalForecast: A Reference Framework for Hierarchical Forecasting in Python. *Journal of Machine Learning Research,* pp. 1-8.

Panagiotelis, A., Athanasopoulos, G., Gamakumara, P. & Hyndman, R. J., 2021. Forecast reconciliation: A geometric view with new insights. *International Journal of Forecasting,* pp. 343-359.

Panigrahi, S. & Behera, H., 2017. A hybrid ETS-ANN model for time series forecasting. *Engineering Applications of Artificial Intelligence,* Volume 66, pp. 49-59.

Pedregosa & al., e., 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research,* Volume 12, pp. 2825-2830.

Shlifer, E. & Wolff, R. W., 1979. Aggregation and Proration in Forecasting. *Management Science,* 25(6), pp. 505-603.

Smith, T. G. & al., e., 2017. *pmdarima: ARIMA estimators for Python.* [Online]
Available at: http://www.alkaline-ml.com/pmdarima
[Accessed 29 05 2024].

Timmermann, A., 2006. Forecast combinations. *Handbook of Economic Forecasting,* Volume 1, pp. 135-196.

Wickramasuriya, S. L., Athanasopoulos, G. & Hyndman, R. J., 2019. Optimal Forecast Reconciliation for Hierarchical. *Journal of the American Statistical Association,* Volume 114, pp. 803-819.

Zellner, A. & Tobias, J., 2000. A note on aggregation, disaggregation and forecasting performance. *Journal of forecasting,* Volume 19, pp. 457-469.