Trabajo Fin de Máster
Ingeniería Industrial

Comparación de algoritmos para la predicción de demanda del número de vehículos de una empresa logística

Autor: F. Javier Medrano Avedillo
Tutor: Daniel Gutiérrez Reina

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Máster
Ingeniería Industrial

Comparación de algoritmos para la predicción de demanda del número de vehículos de una empresa logística

Autor:
F. Javier Medrano Avedillo

Tutor:
Daniel Gutiérrez Reina
Profesor Titular de Universidad

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Trabajo Fin de Máster: Comparación de algoritmos para la predicción de demanda del número de vehículos de una empresa logística

Autor: F. Javier Medrano

Tutor: Daniel Gutiérrez Reina

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

A mi familia

A mis amigos

A mis profesores

Agradecimientos

En primer lugar, agradecer a Daniel Gutiérrez Reina por haberme dado la oportunidad de realizar un proyecto que, por un lado, me permitiese aumentar mis conocimientos en aprendizaje automático, y por otro, dar respuesta a un problema concreto de mi actual empresa.

Por otro lado, me gustaría también dar las gracias al departamento de Ciencias Aplicadas al transporte de Amazon, y en concreto a Fabien Cassassolles, por todo la ayuda que me han brindado a la hora de formalizar el problema, obtener los datos requeridos de los sistemas de Amazon y haber estado siempre disponible para resolver cualquiera de mis dudas.

Por último, estos serán mis últimos créditos cursados en la Escuela Técnica Superior de Ingeniería de Sevilla. Gracias a mis compañeros y profesores que han permitido convertirme en quien soy hoy, no solo en el ámbito profesional sino también en el personal. Gracias a mis padres, Javier y María José, a mi hermano Manuel y a mi pareja Julia por haberme apoyado durante los largos y duros años de estudio.

F. Javier Medrano Avedillo

Sevilla, 2022

Resumen

La inteligencia artificial, el análisis de datos y el aprendizaje automático están cada vez más presentes y con un mayor impacto en los diferentes sectores e industrias. En concreto, en el sector logístico, se encuentra presente en numerosos casos de aplicación como predicción de la demanda, optimización de rutas o robotización de los almacenes y centros logísticos.

En este trabajo se aplican y comparan diferentes algoritmos de aprendizaje automático con el fin de predecir el número de camiones que una unidad de negocio concreta de Amazon va a necesitar a largo plazo. Esta predicción se realiza tanto a nivel global europeo como para cada país concreto.

Antes de llevar a cabo la implementación de dichos algoritmos, se necesita la extracción de los datos de la base de datos de la empresa, la realización de un análisis exploratorio, la limpieza y transformación de los mismos para ser consumido por los diferentes modelos de aprendizaje automático.

Posteriormente, se ha analizado y comparado la precisión de los distintos modelos, confirmando que son capaces de predecir nuestra variable objetivo con el nivel de exactitud adecuada y siendo el modelo DeepAR el que presenta una mayor precisión. Finalmente, una vez comprobado el éxito de dichos modelos se han propuesto posibles líneas de trabajo e investigación futuras.

Abstract

Artificial intelligence, data analysis and machine learning are trending topics having an increasing application and impact in different sectors and industries. Specifically, in the logistics sector, it is present in numerous application cases such as demand prediction, route optimization or robotization of warehouses and logistics centers.

This Master Thesis compares and implements different machine learning algorithms in order to predict the number of trucks that a specific Amazon business unit will need in the long term. This prediction will be made both at a global European level and for each specific country.

Before carrying out the implementation of these algorithms, it is necessary to extract the data from the company's database, perform an exploratory analysis, clean and transform them to be consumed by the different machine learning models.

Subsequently, the accuracy of the different models has been analyzed and compared, confirming that they are capable of predicting our target variable with the appropriate level of accuracy, and surfacing the DeepAR model as the one with a highest accuracy. Finally, once confirmed the success of the studied models, future possible lines of work and research have been proposed.

INDEX

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Over the last few years, with the increasing of the number of clients ordering products from home, the logistic players (Amazon, DHL, Paak, Aliexpres) competing for the market and the available systems to capture client and logistic data (IoT, apps..), product demand has become a really challenging and critical problem for logistic/transportation companies.

I work as a Data Analyst within the Amazon Middle Mile Transportation Department. The Transportation organization is responsible of planning and managing the operations of the package transportation across the European Amazon Logistic Network. As a Data Analyst, my role consists in collecting, analyzing and transforming transportation data into data-driven business decisions and insights.

1.1 Amazon Logistic Network

Amazon Transportation Network (Figure 1) is divided in Middle Mile and Last Mile.

Middle Mile team develops routing solutions to move customer orders from its vendors and fulfillment centers to its network of sortation centers and delivery stations in the most efficient possible way**.** The middle mile department is sub-divided in three teams: 1) Inbound – responsible of moving the packages from the external vendors to Amazon Fulfillment Centers 2) Warehouse Transfers – Responsible of moving the packages between the different Fulfillment Centers and 3) Outbound – responsible of moving the packages downstream from the fulfillment centers to the Sort Centers and Delivery Station.

Last Mile team is responsible of moving the customer orders from the delivery station to your doorstep.

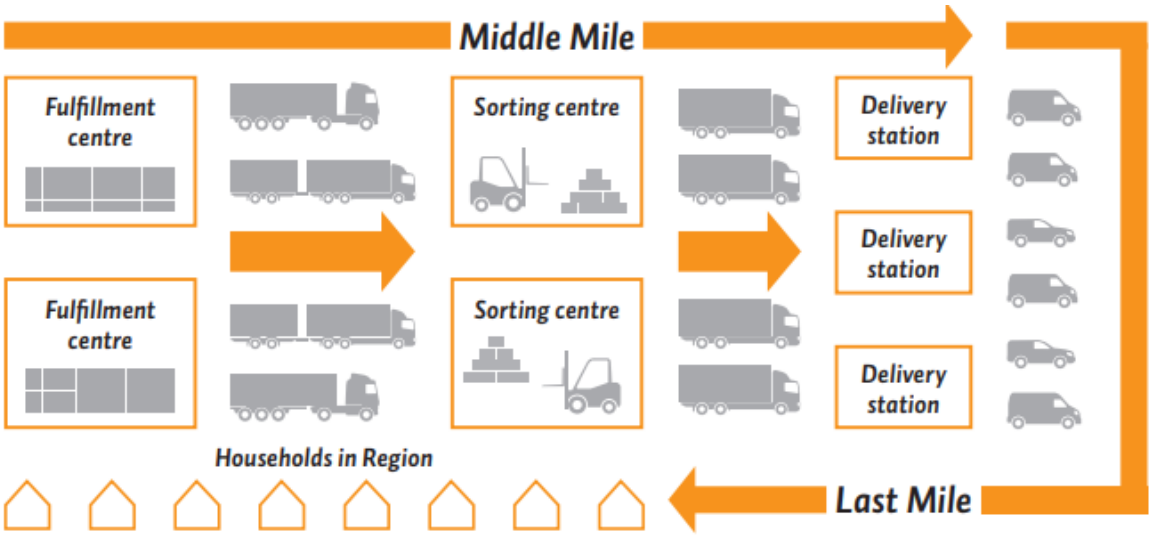


Figure 1 Amazon Logistic Network [1]

One of the key missions of this transportation teams is contract the required carriers (contractors providing drivers and trucks) to assure we can allocate and transport all the packages/volume. One of the key metrics required by our Senior executives to negotiate with the carriers is a forecast of the number of trucks we are going to need on a weekly basis.

So far Amazon has developed Statistical models to predict these number of trucks for the different Transportation operation for a three weeks Horizon (Operational/Tactical Level). However, it would be really interesting for our Senior Executives to have longer Horizon predictions, allowing them to go from operational/tactical decisions to long-term/strategical ones (Increasing/decreasing business with certain carriers, moving business from one country to others, preparing special planning for

1.2 General Description and Objectives

The Master Thesis was developed with the collaboration of the Applied Science Team (ATS) under the supervision of a Senior Machine Learning Researcher. The main objective is to produce a long-term number of truck loads forecast for the Amazon European Warehouse Transfers business (responsible of moving the packages between different Fulfillment Centers). The purpose is to predict 60 weeks in the future on a weekly granularity on different aggregations level (Europe, Country and Cluster). Specifications are summarized in Table 1.

Table 1 Project specifications (from internal Amazon document)

Forecast Target	Number of trucks that fall within the scope of ATS
Target Units	Count of truck loads (Count of VRIDs) **Note**: VRID is defined as unique identifier of a concrete truck going from a specific Origin Site to an specific Destination Site at a concrete timestamp.
Time Horizon	We expect the prediction to cover 60W
Time Granularity	Weekly granularity, so for a single time series we expect a 60 point prediction
Aggregation Level	Overall Europe/Country/ Cluster **Note**: Cluster is defined as a group of Amazon site centers located in a similar region of a country (Ex: ES-Barcelona cluster group all Amazon sites in Catalonia Area)
Scope	Cover all 3 major ATS flows: OB, WHT, ReLo
Publishing Frequency	N/A as this is an exploratory, strategic initiative
Evaluation Metrics	WAPE is a must, plus whatever other metric is deemed meaningful

The concrete objectives are:

1. Explore different statistical techniques for time series forecasting. The document will try to explain the theorical concepts and review the state-of-the-art of the main time series forecasting techniques
2. Develop a structured methodology to solve a Data Science problem. The document will cover the different phases to solve a ML problem:
    1) Explore Amazon Databases to find and extract the required Data.
    2) Clean the data (Nulls, Outliers, ...).
    3) Understand which variable are important for our analysis.
    4) Apply different statistical models.
    5) Apply methodologies to find the most efficient hyperparameters for each of our models.
    6) Once the model is working, productionalize it using AWS Sagemaker.

3. Compare the performance of the applied statistical models. Covering from old-school classical statistical methods (ARIMA, etc..) to modern state-of-the-art techniques (Prophet and Neuronal Networks approaches)
4. Learn and follow the standard methodology for an RD Applied Science project in Amazon.

1.3 Amazon Applied Science Project Structure

In Amazon per definition an Applied Science project are those that aim to solve a business need with an algorithm, machine learning model or similar science-based solution. This process aims to handle the specific peculiarities of this type of project, such as a dealing with a higher uncertainty, the data dependencies or the specific performance evaluation needs.

This Master Thesis has been developed following the standard Amazon Applied Science project structure, shown in Figure 2. This structure specifies a list of phases that the project will follow from the definition of the problem to the maintenance after the solution has been deployed in production.

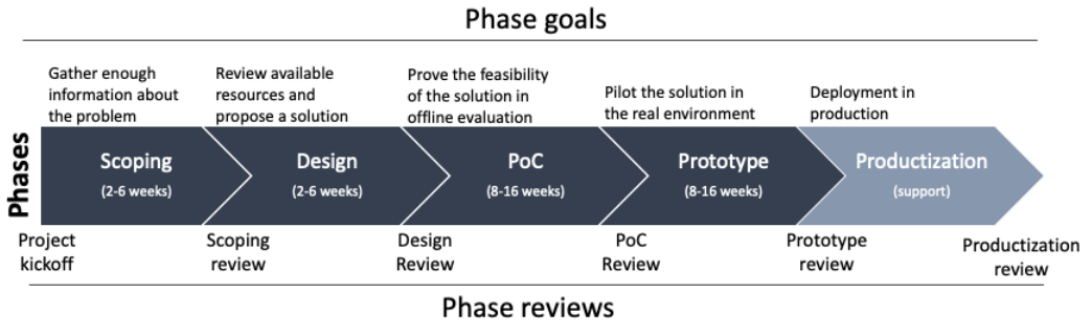


Figure 2 Amazon project structure (https://w.amazon.com/bin/view/GMMAS/ProjectStructure, internal Amazon)

The phases are defined by different goals:

- Scoping: Gather information about the problem and available resources
- Design: with the available resources/data the AS team proposes a solution
- PoC: A (possibly partial) solution is implemented and evaluated offline
- Prototype: A solution is implemented and piloted in the real environment
- Productization: The solution is deployed in production

This AS project was developed in a total of 22 weeks - Scoping (4 weeks), Design (4 weeks) and PoC (14 weeks). Also note that this Master Thesis will just cover until the end of the PoC phase. The Prototyping and Productization on the ML Time Series solution on Amazon systems needs still to be approved.

1.4 Memory Structure

The rest of the memory has been shaped with the following structure:

Chapter 2 covers the basics about time series. It will cover previous theorical concepts that are important to understand the following chapters.

Chapter 3 covers the different time series forecasting models used to predict the number of required trucks. We will try to use and cover on from a theorical point of view the basics from both the classical

statistical models and more modern/state-of-the-art techniques.

Chapter 4 focus on the Exploratory Data Analysis going from the extraction data of the different Amazon databases used to cleaning techniques and initial prescriptive analysis applied to the data.

Chapter 5 develop the practical implementation of the theorical models explained on chapter 3 to our clean data. It will showcase as well the results obtained comparing the performance/accuracy of all different models.

Chapter 6 and 7 wrap up the content of the thesis explaining the main conclusions and surfacing possible futures lines of investigations.

2 TIME SERIES

2.1 Introduction

This chapter is mainly based on the 2 months internal Amazon course to provide Amazon employees with the right tools to perform time series forecasting tasks [2].

Time series data is a collection of observations obtained through **repeated measurements** over **time**. That is:

$$\{y_1, y_2, \dots y_T\}$$

which can be written in a compact manner as:

$$\{y_t \mid t \in \{1, \dots, T\}\}$$

Time series are present in a large number of domains. Some examples include medicine (COVID cases, incidence of cancer), finance (stock market, PIB evolution), climate (average temperatures in a certain region per month, intra-day temperature change). The analysis of this time series leads to unique a concrete problem in statistical modelling and inference

Measurements are typically recorded with a regular frequency (monthly, weekly, daily, hourly...). The regular frequency is a special component in time series problems that is not common in other machine learning (ML) applications.

Thus, the most important difference between Time Series and other types of data is that Time Series Data follows a natural chronological order. The order matters and determines the type of relationship that might exist among observations.

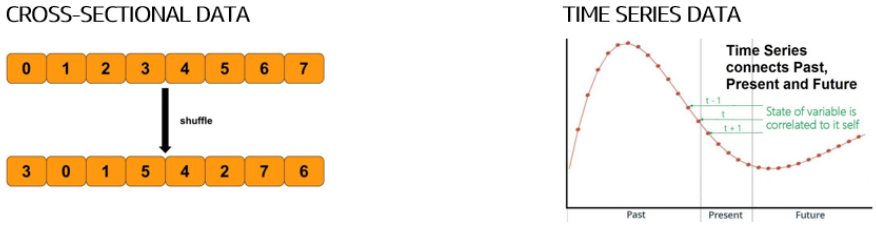


Figure 3 Cross-Sectional Data versus Time Series Data [2]

Figure 3 showcases this difference. In Cross-Sectional Data, the order of the observations is irrelevant. Models can be trained on shuffled data. Their predictions should not depend on that since data points are independent across observations. On the other hand, on time series the order of the observations is chronological and unique. It is reasonable to assume that there exists a serial correlation among observations. This dependence will be a key property to perform time series forecasts.

2.2 Time Series Decomposition

Time series data can exhibit a variety of patterns, and it is often helpful to split a time series into several components, each representing an underlying pattern category. There are three kinds of time series patterns:

Trend - A trend exists when there is a long-term increase or decrease in the data. It does not have to be linear. A general trend is a common component of most time series data. It will often be made up of one or a combination of three trends: an upward trend, downward trend, and/or a horizontal trend. Sometimes we will refer to a trend as "changing direction," when it might go from an increasing trend to a decreasing trend. Of course, it's also possible to see no general trend.

Seasonal - Pattern regularly occurring within that same time Interval. A *seasonal* pattern occurs when a time series is affected by seasonal factors such as the time of the year or the day of the week. Seasonality is always of a fixed and known period.

Cyclic - A cyclical *pattern* is often confused with seasonality, but there are important distinctions. A cycle occurs when the data exhibit rises and falls that are not of a fixed frequency. These fluctuations are usually due to economic conditions, and are often related to the nature of the business. The duration of these fluctuations is usually higher than 2 years. In summary, the main differences between seasonal and cyclic patters are 1) seasonal pattern constant length vs cyclic pattern variable length, 2) average length of cycle longer than length of seasonal pattern and 3) magnitude of cycle more variable than magnitude of seasonal pattern.

It is a common analysis to decompose the time series on their different patterns. Often this is done to help improve understanding of the time series, but it can also be used to improve forecast accuracy and it serves as starting points for some forecasting models. There are two Classical Time Series Decompositions:

$$\textbf{Additive:} \quad y_t = T_t + S_t + R_t$$

$$\textbf{Multiplicative:} \quad y_t = T_t \times S_t \times R_t$$

where y_t is the data, S_t is the seasonal component, T_t is the trend-cycle component, and R_t is the remainder component.

2.3 Time Series Fundamental Concepts

Stationarity

A **stationary** time series is one whose properties do not depend on the time at which the series is observed. This means: 1) Mean and variance are constant over time ($\mu_t = \mu$, $\sigma_t^2 = \sigma^2$) and 2) Covariance depends only on the **lag** and not on the actual time at which it is computed. Time series with trend, seasonality or change in the autocorrelation are not stationary. Figure 4 illustrates these concepts.

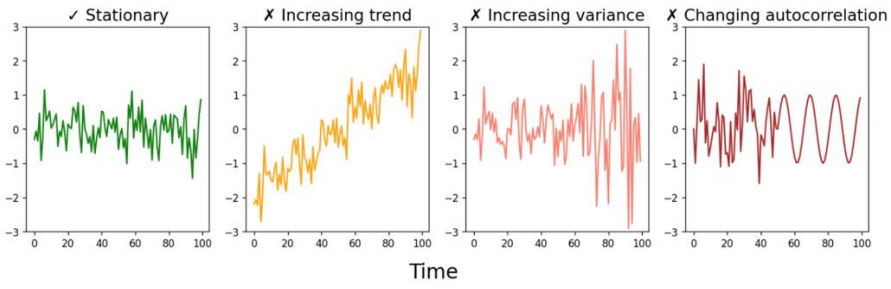


Figure 4 Stationary vs Non-Stationary time series [2]

Autocorrelation in Time series

The covariance is a measure of the joint variability of two random variables *X* and *Y*. If larger values of one variable mainly correspond with larger values of the other, and vice versa (the variables tend to show similar behaviour), the covariance is positive. In the opposite case, when the variables tend to show opposite behaviour, the covariance is negative. The sign of the covariance shows the tendency in the linear relationship between the variables.

$$cov(X,Y) = \mathbb{E}[(X-\mu_X)(Y-\mu_Y)]$$

The correlation is very similar to the covariance, but the latter is dimensionless. The correlation measures the degree to which a pair of variables are linearly related. It is useful because it can indicate a predictive relationship that can be exploited in practice.

$$corr(X,Y) = \frac{cov(X,\ Y)}{\sigma_X \sigma_Y} = \rho_{t\tau}$$

In time series, instead of comparing the relationship between two different variables (X,Y), it is interesting to understand the relationship between *lagged values* of the time series ($y_t, y_{t-\tau}$). That is, values that are tau time periods apart. This concept is illustrated in Figure 5.

Figure 5 Lagged time series

We can calculate the (auto)covariance and the (auto)correlation of lag τ as:

$$cov(y_t, y_{t-\tau}) = \mathbb{E}[(y_t-\mu)(y_{t-\tau}-\mu)] = \gamma_{t\tau} \quad corr(y_t, y_{t-\tau}) = \frac{cov(y_t,\ y_{t-\tau})}{\sigma_t \sigma_{t-\tau}} = \rho_{t\tau}$$

In order to understand and describe the autocorrelation in a time series a really common practice is to build and plot the autocorrelations functions: 1) Autocorrelation Function (ACF) and 2) Partial Autocorrelation Function (PACF)[10][12].

The ACF is an (complete) auto-correlation function which describes the auto-correlation between the present time series value (t) and its lagged values $(t-\tau)$. It is possible to plot it and add the confidence band, building an ACF plot (Figure 6). In simple terms, it describes how well the present value of the series is related.

Each vertical line on the autocorrelation plot represents the correlation between the series and its lag starting from lag 0. The blue shaded region in the plot is the significance level. The lags that lie above the blue line are the significant lags.

On the other hand, the PACF is a (partial) auto-correlation function which describes the auto-correlation between the present time series value (t) and its lagged values $(t-\tau)$, removing the correlation with the intermediate lagged values. On other words, the correlation between y_t and $y_{t-\tau}$

after removing the linear dependence of y_t on y_{t-1} through $y_{t-\tau+1}$. The plot layout is analogous to the ACF one.

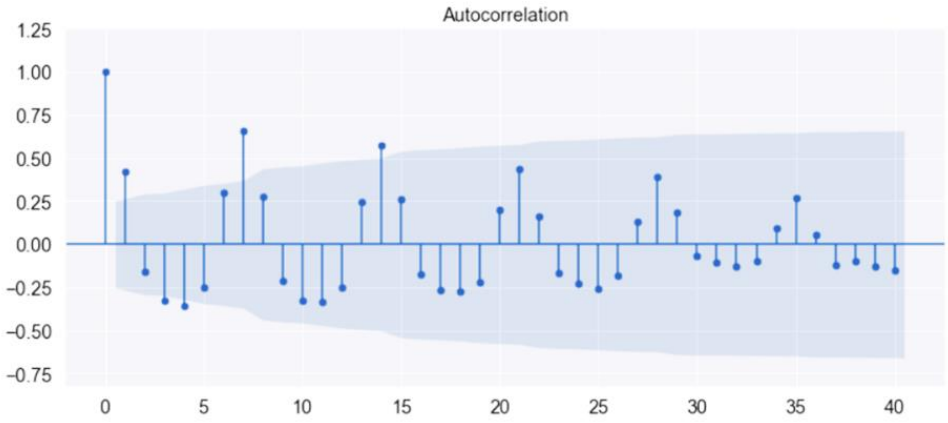


Figure 6 ACF plot

Check time Series Stationarity

There are different methods to identify if a time series is stationary:

- Data Visualization. It is the most basic method, but it is still effective in the majority of the cases.
- Ploting ACF. For stationary series, the ACF correlogram decreases quickly for increasing lags.
- Parametric Test. It is the most rigorous approach. The most popular statistical test is the Augmented Dickey- fuller Test (ADCF). In this method, we take a null hypothesis that the data is non-stationary. After executing this test, it will give some results comprised of test statistics and some other critical values that help to define the stationarity. If the test statistic is less than the critical value, then we can reject the null hypothesis and say that the series is stationary.

Time Series Differentiating

The majority of the classical time series forecasting methods are valid for stationary time series. Therefore, it is really important to understand the existing methodologies to transform a non-stationary time series into stationary.

The most popular method to make a non-stationary time series stationary is differencing: compute the differences between consecutive observations. Differencing can help stabilize the mean of a time series by removing changes in the level of a time series, and therefore eliminating (or reducing) trend and seasonality. It can be distinguished between simple and seasonal differencing.

In simple differencing, the first difference is calculated as the difference between the time series value and the previous one. It is possible to differentiate the first different function producing the second difference.

first difference $\nabla y_t = y'_t = y_t - y_{t-1}$

second difference: $\nabla^2 y_t = y''_t == y_t - 2y_{t-1} + y_{t-2}$

Seasonal differencing. A seasonal difference is the difference between an observation and the same value from the previous season:

$$\nabla_m y_t == y_t - y_{t-m}$$

2.4 Time Series Forecasting

Time series forecasting is making scientific predictions based on historical time stamped data. This means to use a model to analyze the trends and patterns in historical data to predict the future as shown in Figure 7. It involves building models through historical analysis and using them to make observations and drive future strategic decision-making.

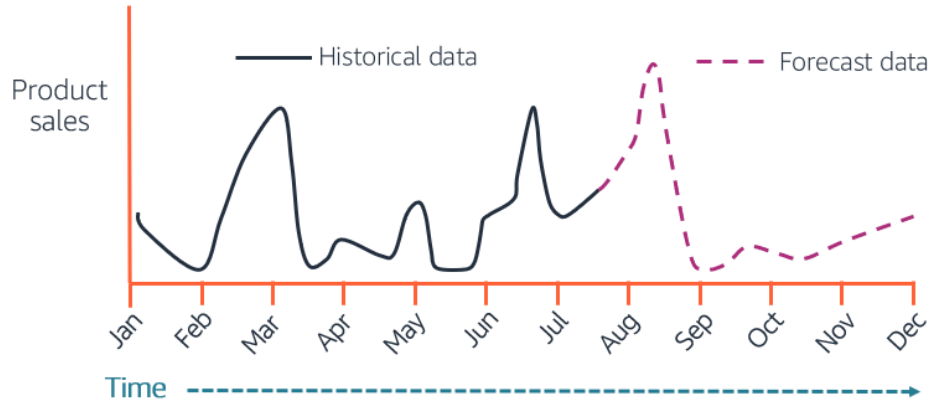


Figure 7 Time Series forecasting

An important distinction in forecasting, different to other areas of ML, is that at the time of the prediction, the future outcome is completely unavailable and can only be estimated through careful analysis and evidence-based priors. As it was commented on previous section, time series forecasting differs from a typical machine learning problem in that the order of the input data points matters.

Forecasting time series data can be posed as a supervised learning problem in Machine Learning. Given a sequence of numbers from a time series data set it is required to structure the data to look like a supervised learning problem as well as to use previous time steps as input variables and predict the next time step as the output variable.

Forecasting is an actual difficult task. The predictability of an event or a future quantity depends on several factor such as 1) how well the contributing factors to the phenomenon are understood, 2) how much past data is available, 3) how similar the future is to the past 4) how/whether the forecasts can affect the phenomenon to be forecast.

2.5 Taxonomy of Time Series Forecasting Problems

Applications in diverse fields (business, economics, astronomy, medicine, …) played a key role in the development of time series methodology. As a result, a variety of disciplines have contributed novel ways of thinking about time series data sets. This means that the way to approach forecasting problems can vary depending on the point of view. It is possible to classify forecasting problems and methods according to different dimensions of interest as shown in Figure 8.

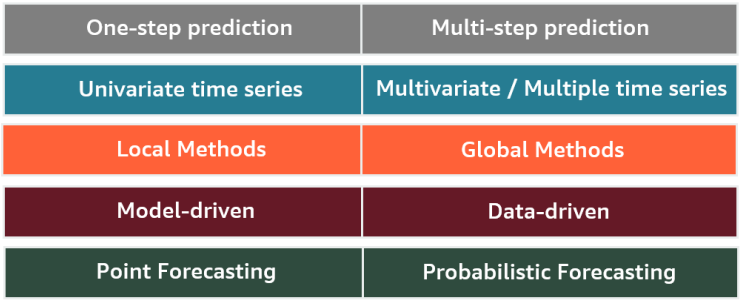


Figure 8 Forecasting approaches according to different criteria

One-step forecast vs Multi-step forecast

This category determines how far into the future are predictions made. In *one-step-ahead prediction*, you're using historical data to make a prediction for the next data point that will occur (Figure 9), For example, using historical sales data for a particular product to predict tomorrow's sales of the same product. The prediction is made using only observed values.

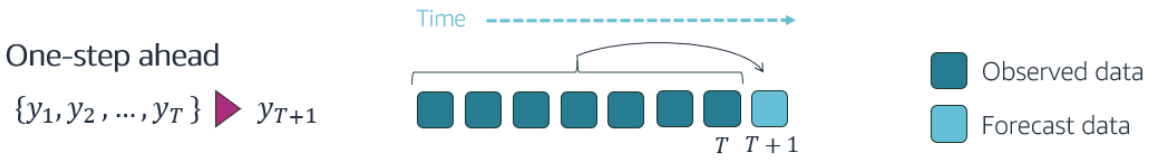


Figure 9 One step prediction

Multi-step-ahead prediction means that required prediction is farther into the future than just the immediately arriving data point (Figure 10). So now, for instance, instead of predicting only tomorrow's sales, you need to predict the product's sales for the entire month. The prediction for the time step *T*+*k* is made using forecast values for *T*+1, *T*+2, …until *T*+*k* - 1.

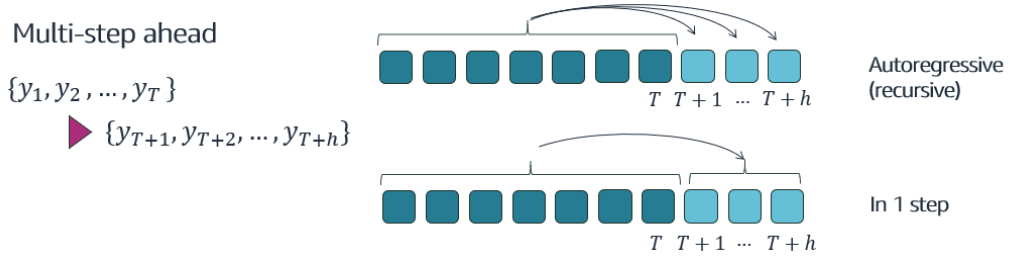


Figure 10 Multi-step ahead prediction

There are different strategies for multi-step forecasting:

Multiple Output in 1 step Forecast Strategy - The multiple output strategy involves developing one model that is capable of predicting the entire forecast sequence in a one-shot manner. In the case of predicting the number of trucks for the next two weeks, we would develop one model and use it to predict the next two weeks as one operation:

```
prediction(t+1), prediction(t+2) = model(obs(t-1), obs(t-2), ..., obs(t-n))
```

Recursive Multi-step Forecast Strategy - The recursive strategy involves using a one-step model multiple times where the prediction for the prior time step is used as an input for making a prediction on the following time step. The basic idea is to recursively compute one-step ahead forecasts until reaching the desired forecast horizon. In the case of predicting the number of trucks for the next two days, we would develop a one-step forecasting model. This model would then be used to predict week 1, then this prediction would be used as an observation input in order to predict week 2:

```
prediction(t+1) = model(obs(t-1), obs(t-2), ..., obs(t-n))
prediction(t+2) = model(prediction(t+1), obs(t-1), ..., obs(t-n))
```

Because predictions are used in place of observations, the recursive strategy allows prediction errors to accumulate such that performance can quickly degrade as the prediction time horizon increases.

Univariate vs Multivariate *Time* Series

This category defines the number of time series that the forecast models are working with as depicted in Figure 11.

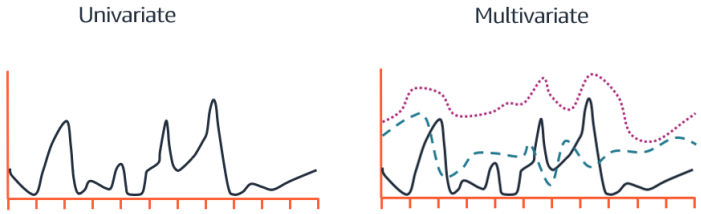


Figure 11 Univariate versus Multivariate Time Series

In a univariate time series, a single variable is measured over time and predictions for the single variable are made from one time series. For example, forecast number of passengers traveling from Madrid to Sevilla by AVE next year.

In a multivariate time series, multiple variables are measured over time. In a multivariate time series, the different time series are treated as a single observation. For example, forecast number of passengers traveling through CDG airport next year, their transit time, their luggage's weight, etc… Notice that you may have multiple time-dependent variables as input and be interested in predicting only one of the variables as output

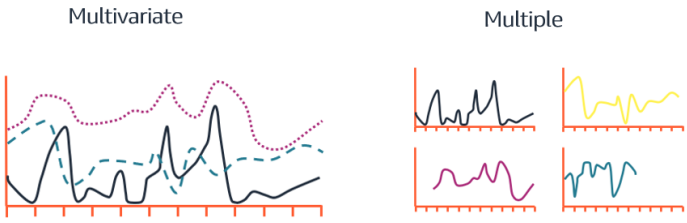


Figure 12 Multivariate Time Series versus Multiple Time Series

A multiple time series is similar to the notion of a multivariate time series in that it is making a prediction based on more than one time series. However, the multivariate treat the different time series as a single observed vector form, whereas the multiple treat the different time series as distinct

observations. Figure 12 iillustrates the differences. That is, using the previous example, a multivariate time series is treating the time series of passengers, their transit time, their luggage, etc. as a single observation in vector form. By contrast, an example of a multiple time series problem would be looking at product sales across an entire company, like Amazon.com. Each of the millions of products on Amazon.com would have its own distinct time series, which is used to make the forecast.

In these scenarios, a substantial amount of data on past behavior of similar, related time series can be leveraged for making a forecast for an individual time series. Using data from related time series not only allows fitting more complex (and hence potentially more accurate) models without overfitting, it can also alleviate the time and labor-intensive human selection and preparation of co-variates and model selection steps required by classical techniques.

Local vs Global Methods

Consider the problem of having to forecast many time series as a group. The safest way (making fewest assumptions) to approach this problem is to assume each of the time series in the set potentially comes from a different data generating process, so it should be modelled individually, as a separate regression problem of finding a function to predict future values from the observed part of the series. This is the standard univariate time series forecasting problem; we call it the local approach to the problem.

From a statistical/machine learning perspective, the local approach suffers from one shortcoming, which is sample size. Temporal dependence and the short length of the series makes time series forecasting a notoriously difficult problem. Individual time series cannot be modeled in a data-driven way because even basic models will suffer from over-fitting. To overcome over-fitting, an excessive burden is placed on manually expressing prior knowledge of the phenomena at hand (assuming seasonal patterns, looking for external explanatory variables, calendar effects, separate long/short term effects, etc.) and restricting the fitting. This limits accuracy, since finding appropriate simple models is not always possible; it also limits scalability, since modeling each time series requires human supervision. As a result, ensemble simple forecast models have been the most widespread approach to automatic time series forecasting (i.e. without prior information) for the last four decades.

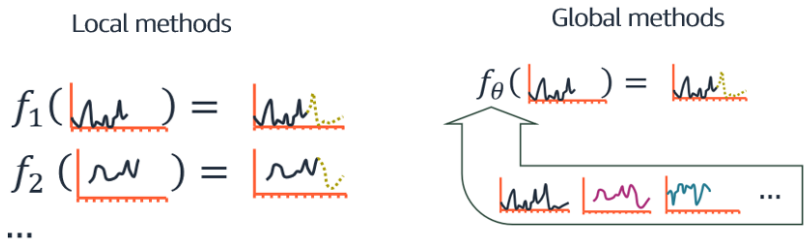


Figure 13 Local methods versus global methods

A univariate alternative to the local approach, called the global or cross-learning approach (Figure 13), has been introduced to exploit the natural scenario where all series in the set are “similar”, “analogous” or “related” (e.g. the demand for fiction books follows a similar pattern for all subgenres, stores or cover designs). The idea behind the global approach is to introduce the strong assumption of all time series in the set coming from the same process, because even when not true, it will pay off in terms of forecasting accuracy. Global methods pool the data of all series together and fit a single univariate forecasting function. The global approach overcomes the main limitation of the local

approach: it prevents over-fitting because a larger sample size is used for learning compared to a local counterpart. On the other hand, the same function must be used to forecast all-time series in the set, even when the series are different, which seems restrictive (and less general than local). Global models can increase model complexity compared to individual local models. Modern machine learning has developed an impressive array of techniques for increasing and controlling model complexity (e.g. deep networks, regularization) that can be applied to global models.

Model-Driven vs Data Driven

Model-Driven methods start with a solid idea of how the system works and generate a hypothesis about what aspects of that might have an influence in the task at hand (forecasting in our case). Then model those and collected data to confirm correlations between observations and prediction. Then apply those features in the real world, automatically.

Model-Driven approaches limit complexity. They are powerful because they rely on a deep understanding of the system they are trying to model. Models cannot accommodate infinite complexity and generally must be simplified. They have trouble accounting for noisy data and unincluded variables. Model-driven is expensive and takes time. Finding a suitable model and refining it until it produces the desired results is often a lengthy process.

Data-Driven is the new way of thinking, enabled by machine learning. Find an algorithm that can spot connections and correlations that you may not even know to suspect. Turn it loose on the data. Data-Driven approaches based on machine learning require a good bit of data to get decent results. They learn from examples, and there needs to be enough examples to cover the full range of expected variation and null cases.

***Point vs* Probabilistic forecasting**

This category pertains whether the prediction output is presented as a single value or as a range of values with an associated probability distribution as it is depicted in Figure 14.

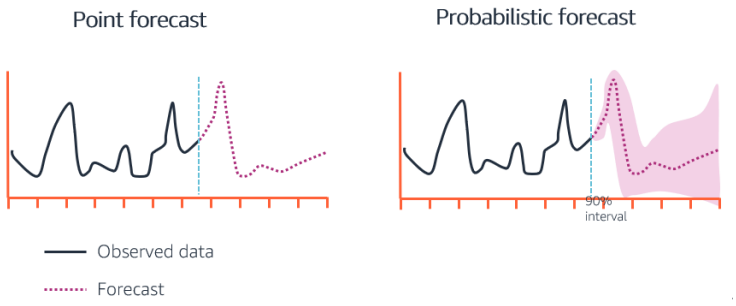


Figure 14 Point versus probabilistic forecasting

Point forecasting returns a single estimate for a time step. For instance, a model may predict $457 as tomorrow's stock price. Point forecasts are typically insufficient for decision making.

Probabilistic forecasting, on the other hand, returns a range of estimates for a time step with corresponding probabilities. So instead of just the one figure prediction, with probabilistic forecasting, the model output is a prediction range of $448-$461 with 90% confidence, with $457 being the most probabilistic price at a 50% likelihood. Logically, the uncertainty around this prediction increases as they move farther into the future.

2.6 Forecast Errors Measures and Evaluation Metrics

2.6.1 Training and test sets

It is important to evaluate forecast accuracy using genuine forecasts. The accuracy of forecasts can only be determined by considering how well a model performs on new data that were not used when fitting the model.

When choosing models, it is common practice to separate the available data into two pieces, training and test data, where the training data is used to estimate the parameters of the forecasting model and the test data is used to evaluate its accuracy. Because the test data is not used in determining the forecasts, it should provide a reliable indication of how well the model perform on new data.

There are some particularities when performing the training-test split on Time Series data. Typical methods used in other ML areas assume that all observations (data points) are independent and identically distributed (random variables). Methods that rely on this assumption are not appropriate for modelling time series Data. Observations close in time are likely to be correlated.

To evaluate a forecasting model on time series data, it is usual to train on data points up to a certain time and use subsequent data (”out-of-sample”) to test as it is shown in Figure 15. Typically, distribution between train and test data is 80-20%. Test set should be at least as large as the maximum forecast horizon required.

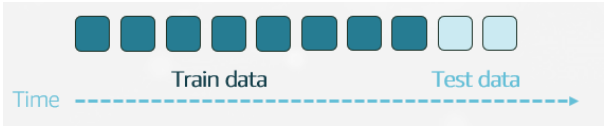


Figure 15 Training-test split on Time Series data

In the case of time series, the cross-validation is not trivial. We cannot choose random samples and assign them to either the test set or the train set because it makes no sense to use the values from the future to forecast values in the past. There is a temporal dependency between observations, and we must preserve that relation during testing.

2.6.2 Evaluation metrics for point models

The forecast errors are the differences between the actual values of the time series in the forecast horizon/test data (y) and the forecast values predicted by the model ($\hat{y}$) (Figure 16). The evaluation metrics look at the overall model errors, either taking these errors in absolute or relative values.

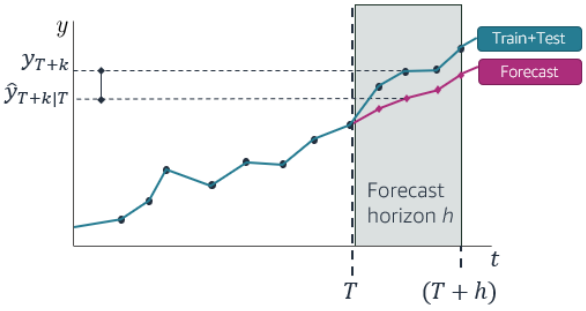


Figure 16 Forecasting error

Absolute Errors

Mean Squared Error (MSE). It is the average of the squared errors. As MSE squares the error differences, it penalizes even the smallest errors, leading to over-estimation of how bad the model is. MSE is defined as follows:

$$\boldsymbol{MSE = \frac{1}{h}\sum_{k=1}^{h}(y_{T+k} - \hat{y}_{T+k|T})^2}$$

Root Mean Square Error (RMSE). It is the square root of the average of squared errors:

$$\boldsymbol{RMSE = \sqrt{MSE}}$$

and is therefore more sensitive to outliers than other accuracy metrics. While MSE reports errors in units squared, the RMSE reports the errors in the original units, making it easier to interpreted the error.

Mean absolute error (MAE). It is the average of the absolute differences between the target values and the values predicted by the model:

$$\boldsymbol{MAE = \frac{1}{h}\sum_{k=1}^{h}|y_{T+k} - \hat{y}_{T+k|T}|}$$

Being a linear score, all the individual differences are weighted equally, hence the MAE is more robust to outliers and does not penalize the errors as extremely as MSE.

When comparing forecast methods applied to a single time series, or to several time series with the same units, the MAE is popular as it is easy to both understand and compute. A forecast method that minimizes the MAE will lead to forecasts of the median, while minimizing the RMSE will lead to forecasts of the mean. Consequently, the RMSE is also widely used, despite being more difficult to interpret.

Relative Errors

Percentage errors have the advantage of being unit-free, and so are frequently used to compare forecast performances between data sets:

Mean Absolute Percentage Error (MAPE). It takes the absolute value of the percentage error between observed and predicted values for each unit of time, then averages those values:

$$\boldsymbol{MAPE = \frac{100}{h}\sum_{k=1}^{h}\frac{|y_{T+k} - \hat{y}_{T+k|T}|}{|y_{T+k}|}}$$

A lower value indicates a more accurate model. MAPE is useful for cases where values differ significantly between time points and outliers have a significant impact.

Symmetric Mean Absolute Percentage Error (SMAPE). It is defined as follows**:**

$$\boldsymbol{SMAPE = \frac{100}{h}\sum_{k=1}^{h}\frac{|y_{T+k} - \hat{y}_{T+k|T}|}{\frac{1}{2}(|y_{T+k}| + |\hat{y}_{T+k|T}|)}}$$

The advantage of the symmetric version over the regular MAPE is that SMAPE avoids large errors when the actual values are close to zero.

The Weighted Absolute Percentage Error (WAPE). It is defined as:

$$\boldsymbol{WAPE} = \frac{\sum_{i,t}\left|\boldsymbol{y_{T+k}} - \hat{\boldsymbol{y}}_{\boldsymbol{T+k|T}}\right|}{\sum_{,}|\boldsymbol{y_{T+k}}|}$$

It is not so well-known in common literature, but it is the one used in Amazon. It measures the overall deviation of forecasted values from observed values.

2.6.3 Evaluation metrics for *probabilistic* models

A probabilistic time series forecast outputs the entire distribution of the forecasted values for a given time point, instead of just a mean or a point forecast. Therefore, the evaluation metrics are totally different from the previous ones and will be based on quantile concept.

The quantile α (α is a percentage, 0<α<1) of a random distribution is the value for which the probability for an occurrence of this distribution to be below this value is **α. A quantile forecast** is a forecast where instead of the median or the mean we forecast at various quantiles of the forecast value. Basically, you predict a quantile instead of predicting the real value. Figure 17 illustrates this concept.

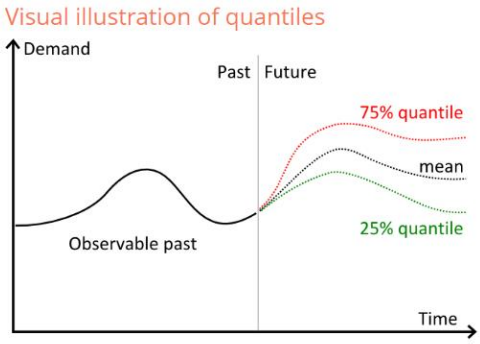


Figure 17 Quantile forecast

Evaluate the accuracy of a quantile forecast is a really complex problem. The metric used to evaluate the accuracy of a quantile forecast is the pinball loss function. The pinball loss function L_α is computed for a quantile α, the quantile forecast *f*, and the demand *d* as:

$$L\_\alpha(d,f) = \begin{matrix}(d-f)\alpha & if\ d \geq f \\ (d-f)(1-\alpha) & if\ f\ > d\end{matrix}$$

This is a complex concept, so it is interesting to cover a practical example on how this functions work in more detail. Figure 18 depicts an example with *d*(*y*)=100 and α=90% (we want to forecast the 90th quantile) where the pinball loss function L_α is calculated for the different values of our forecast value.

For the example above, forecasting 20 units too much will result in a loss of 10%|*f*-*d*|=2. But, on the other side, forecasting 20 units too little will result in a loss of 90%|*f*-*d*|=18. Basically, under forecasting is penalized by α*error, whereas over forecasting is penalized by (1-α) error. Under forecasting is penalized heavily compare to over forecasting on this case. If the model forecast the 90th quantile, the predicted value is likely to be higher than the observed value.

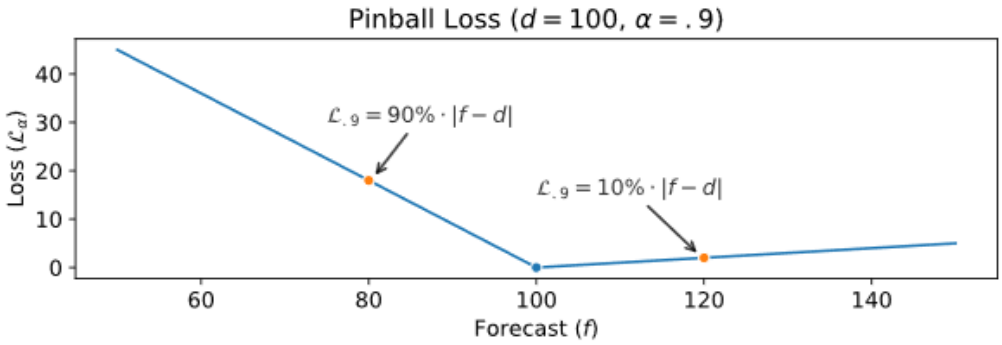


Figure 18 Example of evaluating accuracy using pinball function [7]

2.7 Model Complexity: Bias Variance Tradeoff

Whenever we discuss model prediction, it's important to understand prediction errors such as bias and variance. There is a tradeoff between a model's ability to minimize bias and variance. These concepts are key to avoid underfitting and overfitting in our models.

Bias measures the discrepancy between the average predictions of a model and the actual values due to systematic error that biases all predictions in the wrong direction. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

Variance measures the range of variability of the discrepancy between model predictions and actual values (how spread our predictions are), models with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.

Underfitting occurs when the model is not able to capture the underlying pattern of the data. This normally occurs when 1) we have a low amount of data or 2) the selected model is too simple for our problem (for example, linear models for not linear problems). Overfitting, on the other hand, occurs when the model learns too many details from the behavior of the training data (for example, learned noise will not generalize well to training data). Figure 19 illustrates this.

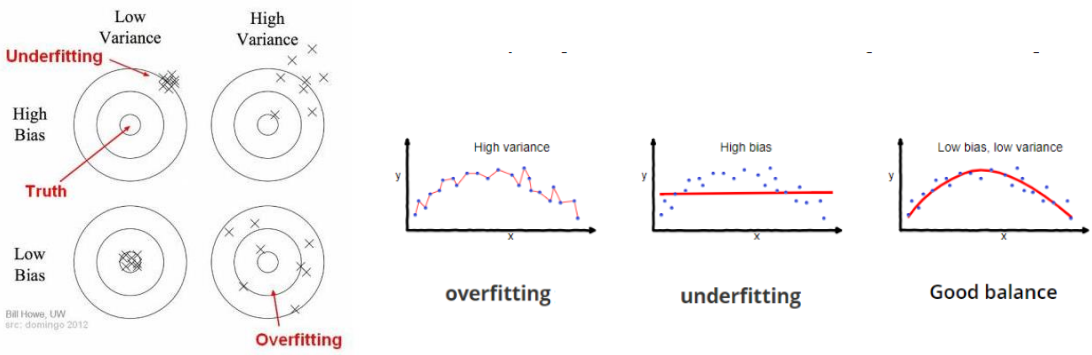


Figure 19 Bias and variance concepts [13]

Without entering in math details in ML de prediction error of a model can be decomposed as follows:

$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Irreducible error is the error that can't be reduced by creating good models. It is a measure of the amount of noise in our data. It is important to note that no matter how good the model is, it is a simplification the reality and the data will have some noise that the model will not be able to explain.

Then, in order to make good model it is important to minimize the total error. For that a trade-off between Bias and Variance is required as it can be seen in Figure 20. If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand, if our model has a large number of parameters, then it is going to have high variance and low bias. So, we need to find the right/good balance without overfitting and underfitting the data.

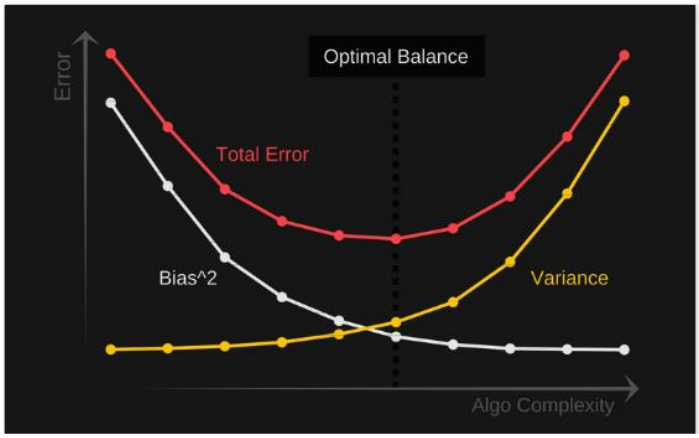
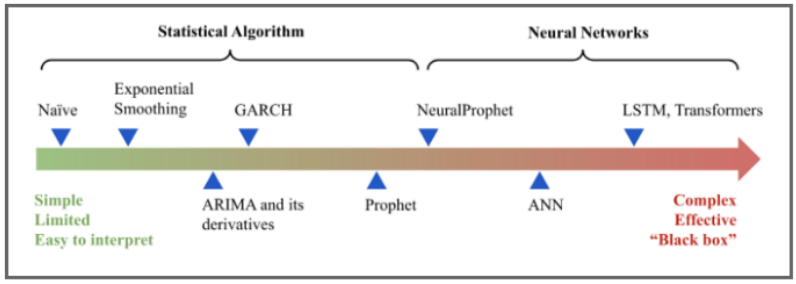


Figure 20 Total error versus model complexity [13]

3 MODELS

Most time series problems require forecasts that are **easily understandable**. At the same time, an efficient forecast is desired. These two aspirations lead to a trade-off between **interpretability and performance**.

This introduces an important distinction between Neuronal Networks and classic statistics, and ultimately a tradeoff between accuracy and explainability illustrated in Figure 21. The right model to choose will depend on whether the goal is to generate the most accurate prediction possible, or to understand the underlying generative processes in the data.



Streamlined examples of time series algorithms regarding their complexity and performance. Made for popularisation and explanatory purpose with probable adjustment. (Made by the author)

Figure 21 Time Series algorithms regarding interpretability and accuracy [9]

Machine learning (and deep learning especially) is the tool of choice to maximize accuracy and are but sacrificing some explainability. ANNs are a powerful forecasting tool, for example, but explaining how the network generated a prediction requires searching through the tangled mess of hidden layers that feed both forward and backward. If senior leadership at the company is weighing a major business decision based on these forecasts, they are unlikely to accept a model that they cannot understand.

A classical model can therefore be an attractive alternative, even given its lower predictive power. By concretely understanding how values in the time series relate to one another, it is much easier to build an intuition for the time series itself, such as how much today's values influence tomorrow, the cycle of seasonality, and more.

To sum up the right forecasting tool ultimately depends on the broader business context of the analysis/problem. If the nature of the problem is more strategic, that is, the analysis presents a small number of data series, abundant historical data and a forecaster with the required statistical knowledge to build a more Hand-crafted model is seek, the suitable approach would be to use "classical" model-driven models based on statistics/econometrics concepts. An example is predicting overall Amazon retail demand years into the future since time series has long history and exhibits clear patterns.

On the other hand, if the nature of the problem is more operational, that is, the analysis presents a large number of time series, with partial lack of historical data, short cycles/burstiness/sparsity and cold start for some time series, the suitable approach would be to use "modern", data-driven neural and deep learning-based models. A good example is the prediction of the demand for each product available at Amazon since time series are irregular and there is a lack of history.

This chapter is mainly based on [2] and [11].

3.1 Classical Models

3.1.1 Simple Forecasting Methods

Some forecasting methods are extremely simple and surprisingly effective. The next four simple methods that can be used as baseline/benchmarks to compare other models against.

Naïve Method

The naïve method of forecasting assigns the value of the last observation as the next forecast. This is a very simple approach that is mostly used for comparison with forecasts generated by other more sophisticated techniques. This method works remarkably good in many economic and financial time series. It is formulated as:

$$\hat{y}_{T+k|T} = y_T$$

Moving Average Method

This method forecast all future values of the series as the average (mean) of its past values. Alternatively, instead of using all past values it is possible to just use a given number of past observations:

$$\hat{y}_{T+k|T} = \bar{y} = \frac{1}{T}(y_1 + y_2 + \ldots + y_T)$$

Note that the method assumes that the weights between the in-sample observation are equal, i.e. the first observation has precisely the exact weight of 1/T as the last one. Suppose the series exhibits some changes in level over time. In that case, the global mean will not be suitable because it will produce the averaged out forecast, considering values for parts before and after the change.

Drift Method

In the case of the series with a trend, Naïve and Average will be inappropriate because they would be missing the essential component. This is a variation of the naive method that includes a constant change over time (drift). The drift is computed taking the slope of the train time series estimated from its first and last value:

$$\hat{y}_{T+k|T} = y_T + k\frac{y_T - y_1}{T-1}$$

Seasonal Naive Method

Similar to Naïve, Seasonal Naïve relies only on one observation, but instead of taking the most recent value, it uses the value from the same period a season ago. For data with a seasonal pattern, one can forecast all future values as the last observed values from the same season of the year. This is a surprisingly good method that can be used as baseline in most seasonal time series. It is formulated as:

$$\hat{y}_{T+k|T} = y_{T+k-m(p+1)}$$

where m is the seasonality cadence and p is the integer part of $(k-1)/m$.

Seasonal Naïve does not require estimation of any parameters and thus is considered one of the popular benchmarks in seasonal data.

3.1.2 Exponential Smoothing Methods

Exponential smoothing methods rely on the same idea than Naïve or Average Methods in which a prediction is a weighted sum of past observations, but the model explicitly uses an exponentially decreasing weight. These methods produce quick and reliable forecasts for a wide range of time series.

Using the naïve method, all forecasts for the future are equal to the last observed value of the series. The naïve method assumes that the most recent observation is the only important one, and all previous observations provide no information for the future. This can be thought of as a weighted average where all of the weight is given to the last observation. In the average method, all future forecasts are equal to a simple average of the observed data. It assumes that all observations are of equal importance, and gives them equal weights when generating forecasts. Exponential smoothing does something between these two extremes.

This section covers the basics of the main exponential smoothing methods and how to apply them to time series forecasting. The selection of the method is pretty manual and require understanding and recognizing the main components of the time series data (trend and seasonality) and how to adapt the smoothing methods to them.

Simple Exponential Smoothing (SES)

Simple Exponential Smoothing (SES) is a time series forecasting method for univariate data without a trend or seasonality. SES models a slowly but unsystematically changing mean in your series, by weighting (α). That is:

$$\hat{\boldsymbol{y}}_{\boldsymbol{t+1|t}} = \boldsymbol{\alpha y_t} + \boldsymbol{\alpha(1-\alpha)y_{t-1}} + \boldsymbol{\alpha(1-\alpha)^2 y_{t-2}} + \ldots$$

When α is closer to 0, it is considered slow learning because the algorithm gives historical data more weight. When α is closer to 1 it is considered fast learning because the algorithm gives more weight to the most recent observation. Therefore, recent changes in the data will have a larger impact on forecasted values. Simple exponential smoothing method can be rewritten as:

$$\hat{y}_{t+h|t} = l_t$$

$$l_t = \alpha y_t + (1-\alpha)\, l_{t-1}$$

The forecast equation shows that the forecast value at time t+1 is the estimated level at time t and that the level at time t is calculated using the level from previous time t-1. It is important to note that SES present a flat forecast function. On other words, all the forecast takes the same value:

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

Because these changes are assumed to be unsystematic, there is no good way to forecast how the mean will change in the future since it could go up or down. If there are systematic changes in your mean, these could come from seasonality or trend. If so, methods that model these changes are more appropriate, such as double or triple exponential smoothing, which will be covered next. Forecasts based on these models will not be flat.

Double Exponential Smoothing

Double Exponential Smoothing is an extension to Exponential Smoothing that explicitly adds support for trends in the univariate time series. This method is classically referred to as Holt's linear trend model, named for the developer of the method Charles Holt. It can be written as:

$$\hat{y}_{t+h|t} = l_t + hb_t$$

$$l_t = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1})$$

$$b_t = \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1}$$

where $0 \leq \alpha \leq 1$ is the smoothing parameter for the trend and $0 \leq \beta^* \leq 1$ is the smoothing parameter for the level. In addition to the α parameter for controlling smoothing factor for the level, an additional smoothing factor is added to control the decay of the influence of the change in trend called b_t.

As with simple exponential smoothing, l_t is a weighted average of observation y_t and the one-step-ahead training forecast for time t, here given by $l_{t-1} + b_{t-1}$. The third equation shows that b_t is a weighted average of the estimated trend at time t based on $l_t - l_{t-1}$ and the previous estimate of the trend, b_{t-1}.

The forecast function is no longer flat but trending. The h -step-ahead forecast is equal to the last estimated level plus h times the last estimated trend value. Hence the forecasts are a linear function of h.

Damped Double Exponential Smoothing

One of the main disadvantages of the constant linear trend is that it increases or decreases forever. So, it tends to over-predict. Thus, damped double exponential smoothing introduces a term that dampens the trend to flatten out. It is formulated:

$$\hat{y}_{t+h|t} = l_t + (\phi + \phi^2 + \cdots + \phi^h)b_t$$

$$l_t = \alpha y_t + (1-\alpha)(l_{t-1} + \phi b_{t-1})$$

$$b_t = \beta^*(l_t - l_{t-1}) + (1-\beta^*)\phi b_{t-1}$$

with $0 < \phi < 1$ is the damping factor. In practice, ϕ is rarely less than 0.8 as the damping has a very strong effect for smaller values. When ϕ is close to 1 it means that a damped model is not able to be distinguished from a non-damped model. For these reasons, we usually restrict ϕ to a minimum of 0.8 and a maximum of 0.98.

Triple Exponential Smoothing: Holt-Winter's method

Triple Exponential Smoothing is an extension of Exponential Smoothing that explicitly adds support for seasonality to the univariate time series. This method is sometimes called Holt-Winters Exponential Smoothing, named for two contributors to the method, Charles Holt and Peter Winters.

There are two variations to this method that differ in the nature of the seasonal component. The additive method is preferred when the seasonal variations are roughly constant through the series, while the multiplicative method is preferred when the seasonal variations are changing proportional to the level of the series.

With the additive method, the seasonal component is expressed in absolute terms in the scale of the observed series, and in the level equation the series is seasonally adjusted by subtracting the seasonal component. Within each year, the seasonal component will add up to approximately zero. With the multiplicative method, the seasonal component is expressed in relative terms (percentages), and the series is seasonally adjusted by dividing through by the seasonal component.

$$\hat{y}_{t+h|t} = l_t + hb_t + s_{t+h-m(k+1)}$$

$$l_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1})$$

$$b_t = \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1}$$

$$s_t = \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$$

Where *m* to denote the frequency of the seasonality. *k* is the integer part of (*h*−1)/*m*.

The multiplicative method is formulated:

$$\hat{y}_{t+h|t} = (l_t + hb_t)s_{t+h-m(k+1)}$$

$$l_t = \alpha\frac{y_t}{s_{t-m}} + (1-\alpha)(l_{t-1} + b_{t-1})$$

$$b_t = \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1}$$

$$s_t = \gamma\frac{y_t}{(l_{t-1} - b_{t-1})} + (1-\gamma)s_{t-m}$$

The level equation for $l_t$shows a weighted average between the seasonally adjusted observation ($\frac{y_t}{s_{t-m}}$) and the non-seasonal forecast ($l_{t-1} + b_{t-1}$) for time t. The trend equation for b_t is identical to Holt's linear method. The seasonal equation for s_t shows a weighted average between the current seasonal index, ($\frac{y_t}{(l_{t-1}-b_{t-1})}$), and the seasonal index of the same season last year (*m* time periods ago).

Exponential smoothing methods are not restricted to those we have presented so far. By considering variations in the combinations of the trend and seasonal components, nine exponential smoothing methods are possible.

3.1.3 ARIMA and its Derivatives Models

The main underlaying idea of ARIMA models is to find the autocorrelations in the data. They are among the most widely used methods for univariate time series forecasting.

Autoregressive models (AR)

The concept behind is really similar to a multiple regression model, in which the target variable is forecasted using a linear combination of the predictors. In the AR model, on the other hand, the target variable is forecasted using a linear combination of the past values. The autoregression term indicates that this method is basically a regression of the variable against itself. This is referred as an AR(*p*) model, or autoregressive model of order *p*:

AR(1) → $\boldsymbol{y_t = c + \phi_1 y_{t-1} + z_t}$

AR(2) → $\boldsymbol{y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + z_t}$

AR(p) → $\boldsymbol{y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + z_t}$

where z_t is the error or white noise.

Moving average models (MA)

Instead of using a linear combination of the past values of the variable, the moving average model

(MA) uses a linear combination of the past forecast errors. This is referred as an MA(*p*) model or moving average model of order *p*:

MA(1) → $\boldsymbol{y_t = c + z_t + \theta_1 z_{t-1}}$,

MA(2) → $\boldsymbol{y_t = c + z_t + \theta_1 z_{t-1} + \theta_2 z_{t-2}}$

MA(q) → $\boldsymbol{y_t = c + z_t + \theta_1 z_{t-1} + \theta_2 z_{t-2} + \cdots + \theta_q z_{t-q}}$

Integrated model (I)

Non-stationary time series that can be converted into stationary by means of successive differences:

I(0) → non-integrated stationary process $u_t \sim I(0)$

I(1) → $y_t \sim I(1)$ if $\nabla y_t = y'_t = u_t$

I(q) → $y_t \sim I(q)$ if $\nabla^q y_t = (1-L)^q (y_t) = u_t$

where u_t is the covariance stationary.

Usually, the order of integration is either I(0), I(1), or I(2). It is rare to see values for q > 2.

Non-seasonal ARIMA models (ARIMA)

ARIMA is an acronym for Autoregressive Integrated Moving Average (in this context, "integration" is the reverse of differencing). ARIMA models combine Autoregressive with Integrated with Moving Average components:

$$\boldsymbol{y_t^* = c + \phi_1 y_{t-1}^* + \cdots + \phi_p y_{t-p}^* + \theta_1 z_{t-1} + \cdots + \theta_q z_{t-q} + z_t}$$

where $y_t^* = \nabla^d y_t$ is the *d*-differenced series. *p* is the order of the autoregressive part (AR), *d* is the degree of differentiating and *q* is the order of the moving average part (MA).

One of the main challenges when applying ARIMA forecasting models is to identify the parameters (*p*,*q*,*d*). There are different methodologies to properly identify these parameters:

- Analyse the PACF and ACF correlograms to understand the existing autocorrelation on the time series and infer the parameters. There are some more or less general rules:
  - If the ACF plot declines gradually and the PACF drops instantly, use Auto Regressive model.
  - If the ACF plot drops instantly and the PACF declines gradually, use Moving Average model.
  - If both ACF and PACF decline gradually, combine Auto Regressive and Moving Average models (ARMA).
  - If both ACF and PACF drop instantly (no significant lags), it's likely you won't be able to model the time series.

- Reading ACF and PACF plots can be both challenging and unprecise, a really good option is using grid search to find the optimal parameter values combination that minimize the error (MAPE/WAPE).

Seasonal ARIMA models (SARIMA)

So far, we have restricted our attention to non-seasonal data and non-seasonal ARIMA models. However, ARIMA models are also capable of modelling a wide range of seasonal data. A seasonal ARIMA model is formed by including additional seasonal terms in the ARIMA models we have seen so far. It is written as:

$$(p, q, d) \times (P, Q, D)_m$$

The seasonal ARIMA model incorporates both non-seasonal (*p*, *q*, *d*) and seasonal factors (*P*, *Q*, *D*) in a multiplicative model. The seasonal part of the model consists of terms that are similar to the non-seasonal components of the model, but involve backshifts of the seasonal period. seasonal AR(*P*) and MA(*Q*) terms predict y_t using data values and errors at times with lags that are multiples of *m*. SARIMA increment in complexity makes it a quite large model compared to ARIMA.

3.2 Prophet Model

As we already commented, classical time series methods present some limitations: 1) their restrictive assumptions and parametric nature limit their performance in real world more operational applications and 2) they require a Senior forecaster with deep domain knowledge in the application itself and in classical time series modelling.

On the other hand, Neuronal Network based model presents a problem of explainability which remains mostly an open research problem in the forecasting. Further, they often require substantial engineering efforts to preprocess data and fine-tune hyperparameters.

Forecasting is a data science task that is central to many activities within an organization. Analysts who can produce high quality forecasts are thus quite rare because forecasting is a specialized skill requiring substantial experience. The result is that the demand for high quality forecasts often far outstrips the pace at which they can be produced. With all these ingredients on mind, Facebook created a Hybrid Methods: called Facebook Prophet [3]. This forecasting method proposes a modular regression model with interpretable parameters that can be intuitively adjusted by analysts with domain knowledge about the time series. The aim of Facebook when designing Prophet was double. On the one hand, to design a simple enough model suitable for a large number of people making forecasts, possibly without training in time series methods; and on the other hand, to tackle a large variety of forecasting problems from different domains and features.

Prophet is a decomposable time series model. In the component combination models, each component is estimated separately and the function that combines them is applied:

$$\boldsymbol{y(t) = g(t) + s(t) + h(t) + e_t}$$

where $\boldsymbol{g(t)}$ is the trend function which models non-periodic changes in the value of the time series, $\boldsymbol{s(t)}$ represents periodic changes, and $\boldsymbol{h(t)}$ represents the effects of holidays or special events. The error term e_t represents any change which are not interpreted by the model.

Trend component may be linear or logistic as depicted in Figure 22. The selection of one or the other will depend on the characteristics of the data. If the growth of the series saturates when it reaches some level, maximum and/or minimum, the trend of the model will be logistic. In practice, we encounter this situation, for example, when training a model of population growth model where, in general, this growth is limited by the environment and will reach a maximum. When we use Prophet, we must indicate the parameters cap and floor to set these saturation limits.

Prophet's algorithm estimates the trend differently if the growth is linear or logistic. The logistic growth model in its most basic form follows the equation as a function of time:

$$\boldsymbol{g(t) = \frac{C}{1 + exp(-k(t - m))}}$$

where *C* is the capacity limit, *k* is the growth rate and *m* is an offset parameter.

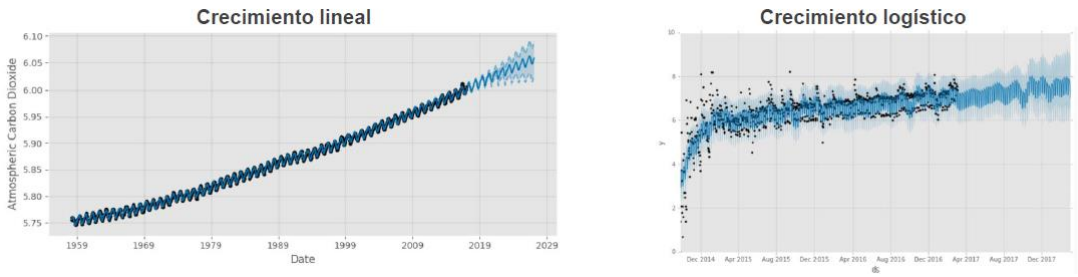


Figure 22 Linear vs logistic growth

In the basic linear growth model, the trend component is:

$$\boldsymbol{g(t) = kt + m}$$

In practice, *C* can be a function of time, *C*(*t*). Also, the growth rate *k* can vary. In order to take the later into account, changepoints in time are considered. Changepoints are “abrupt" changes in the trend, caused by known or unknown events. Different growth rates and offsets are used for the different time intervals defined by the position of the change points. These results in the case of the linear model in a piece-wise linear trend component.

Thus trend setting is determined by the number of changepoints chosen. Prophet automatically detects these changepoints. However, it is possible to specify them manually or intervene in the way Prophet detects them by adjusting parameters that allow us to do so. So, we are able, for example, to solve overfitting (or underfitting) problems. There are ways to intervene in the detection of changepoints by modifying the change point_prior_scale and changepoints parameters of the model.

To estimate seasonality, Prophet makes use of Fourier series:

$$\boldsymbol{s(t) = \sum_{n=1}^{N} (a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right)}$$

P is the period of the expected seasionality.

One of the key parameters to choose, therefore, will be the order of the series (*N*). Prophet recommends taking order 10 and 3 for the cases of annual and weekly seasonality respectively.

The **Holidays and Extra Regressors** information are added to the model using the holidays parameter and the add_regressor method. For the holidays, the input must be a *dataframe* with the columns ds (date) and holiday (holiday name). For the regressors, a column is added to the original *dataframe* with the value of that regressor. This information (holidays and regressors) must be available for the future forecasted values as well.

Overall Prophet is a really used forecasting techniques with a good balance between explainability (much more intuitive than Neuronal Network), accuracy (in general perform better than classical models) and implementability (simple model with easy to configure parameters).

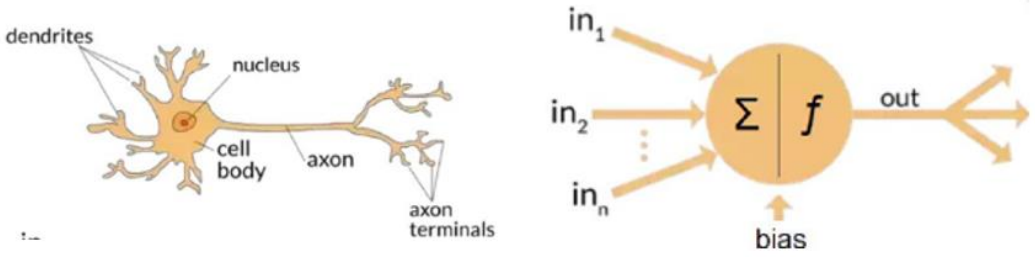
3.3 Artificial Neuronal Network Models (ANN)

Due to the high complexity and lack of examinability, most scientist in 2011 still regarded ANN models as dead ends in Machine Learning. In 2012, a Deep Neural Network architecture AlexNet won the ImageNet challenge (large photos dataset with more than 14 million images), beating handcrafted parametric models, making the hype back. On this new era, the world is becoming really data-driven,

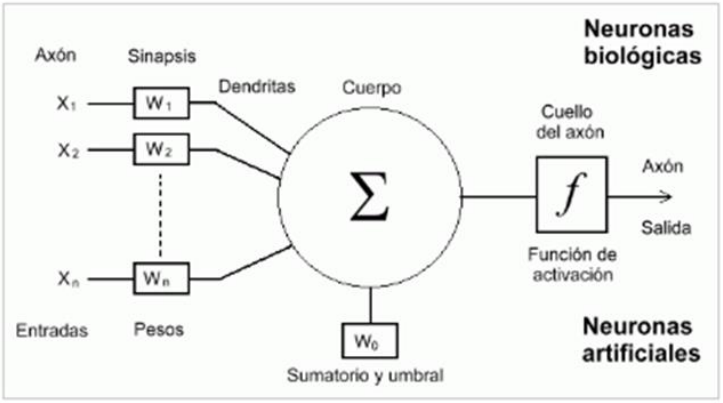
with a huge increase on amount of generated data (Apps, IOT, Robotic, sensors, ...). Neuronal Networks, are Data-centric models, which can outperform more parametric statistical approaches with the right amount of Data. This type of algorithms can spot connections and correlations that you may not even know to suspect. As it was already commented, they learn from examples, and there needs to be enough examples to cover the full range of expected variation and null cases.

3.3.1 Introduction: Neuronal Network Intuition

An Artificial Neural Network (ANN) is a mathematical model inspired by the biological behavior of neurons and how they are organized to form the structure of the brain. The brain can be considered a highly complex system, where it is estimated that there are approximately 100 billion neurons in the cerebral cortex (human) and that they form a network of more than 500 trillion neuronal connections (a neuron can have 100 thousand connections, although the average is between 5000 and 10000 connections).



(a)



(b)

Figure 23 Biological inspiration and basic structure of an ANN

The main function of neurons is the transmission of nerve impulses. These travel throughout the neuron, starting with the dendrites until they reach the axon endings, where they pass to another neuron by means of the synaptic connections. The way we respond to stimuli from the outside world, and the learning we can do from it, is directly related to the neural connections in the brain, and ANNs are an attempt to emulate this fact. Figure 23 illustrates the relationship between the biological neuron and the artificial one or perceptron [4].

A single-layer perceptron is the basic unit of a neural network. A perceptron consists of input values(x), weights (w) and a bias (b), a weighted sum (Σ) and activation function (f).

The inputs are the stimulus that the artificial neuron receives from the surrounding environment, and the output is the response to that stimulus. The neuron can adapt to the surrounding environment and learn from it by modifying the value of its synaptic weights, and therefore they are known as the free

parameters of the model, since they can be modified and adapted to perform a given task. In this model, the neuronal output Y is given by:

$$Y = f(\sum_{i=1}^{n} w_i x_i)$$

One of the most important elements in the construction of an ANN is the determination of the activation function. The activation function transforms the inputs of a neuron into the signal that propagates through the network. The activation function is chosen according to the task performed by the neuron. Among the most common we can highlight: step function, sigmoid function, tanh function or Relu Function.

The Multilayer Perceptron

Individual perceptrons can be grouped together to form networks of neurons connected to each other, so that the output of one becomes the input of another. Although the way in which they can be grouped is completely free and varied, the most common case of grouping corresponds to an organization in ordered layers such that the outputs of the neurons of one layer are the inputs of the neurons of the next layer, as it is depicted in Figure 24.

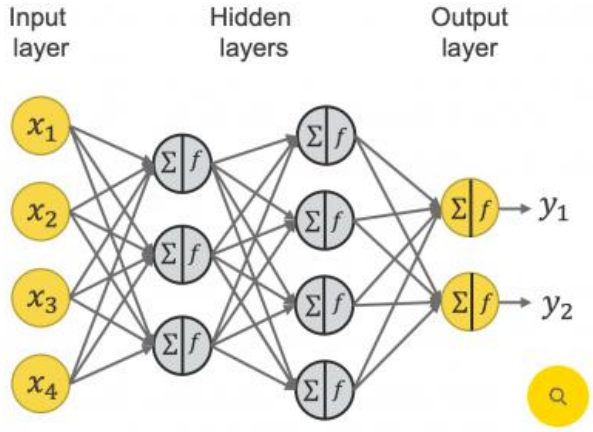


Figure 24 Structure of a simple Multilayer perceptron

Thus, we have the input layer formed by the inputs to the network (which are not really neurons), the output layer formed by the neurons that constitute the final output of the network, and the hidden layers formed by the neurons that lie between the input and output nodes. An ANN can have several hidden layers or none at all. The synaptic connections (the connections to and from neurons) indicate the signal flow through the network, and have the corresponding weight (w_{ij}) associated with them.

Learning

Once a particular neural network is fixed, it can be seen that the only parameters we can modify to obtain different behaviors are the synaptic weights between neurons in different layers and the bias. Therefore, the usual problem is that of, given a set of data for which the desired output is known, finding the appropriate weights of the network so that a correct approximation of the outputs is obtained if the network receives the input data. That is, find the weights that make the function that computes the network the best possible approximation for the data in our set.

Note that this problem (like most problems associated with machine learning) can be posed naturally

as an optimization problem, where the objective is to find the weights of the network that minimizes the error made by the network on a set of data. The error function is commonly known as cost function (C).

The main method to find the weights *w* and bias b that minimize the cost functions is called Backpropagation [5]. The main goal of this methodology is to compute the partial derivatives $\partial C/\partial w$ and $\partial C/\partial b$ of the cost function C with respect to any weight w or bias b in the network. It uses the Gradient Descend method to iteratively move the model weights toward the opposite gradient direction (minimizing the cost function). The amount that controls how much we move the weights in this direction is called learning rate and it can be a predefined constant.

It is known that one of the main problems of neural networks is overfitting. Numerous problems arise when this phenomenon appears and it is not easy to solve analytically. There are different methods to help reduce or limit this overfitting.

One known option is to simply remove connections from the network randomly, known as dropout. It is illustrated in Figure 25. In practice, the method includes a parameter that calibrates the probability that each node is affected. It is important to note that this method is only applied in the training phase of the model and never in the test phase. Experience shows that it is an effective technique

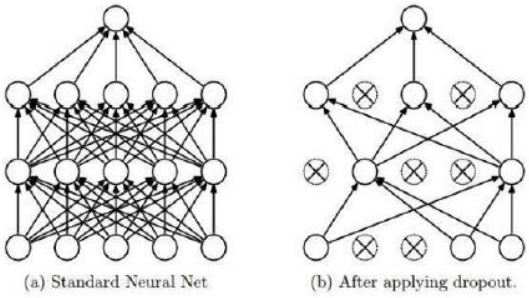


Figure 25 Standard Neural Net vs Neural Net after applying dropout

Another important aspect on ANN is the network topology. It must be defined a priori and will have a significant impact on the quality of the estimates. Intuitively, larger networks will provide better results, although in practice this is often not the case due to the increased likelihood of overfitting. Very complex structures can be generated which can make it difficult to solve problems encountered during training.

It is suggested to start with simple structures and increase the degree of complexity of the network topology. Some of the most used Neuronal Network structures are shown in Figure 26. They include:

1) Markov Chain. It is widely used for probabilistic modeling and state changes.
2) Recurrent Neuronal Network (RNN). It is used for Natural Language Processing.
3) Long/Short Term Memory (LTSM). It is useful when there is a correlation between separated elements of a sequence.
4) Generative Adversarial Network (GAN). It has application, for example, in used on computer vision.

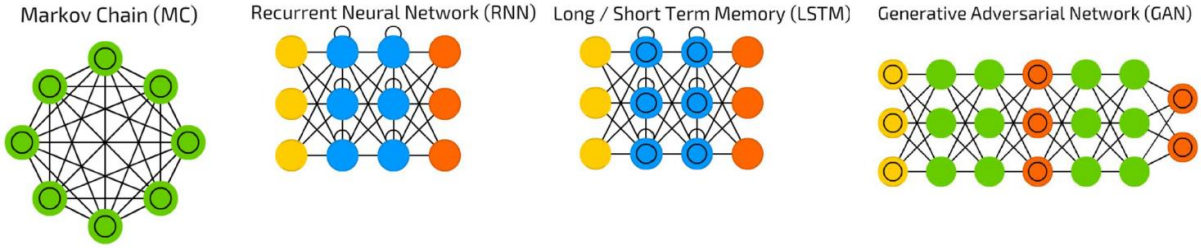


Figure 26 Examples of Neural Network topologies

3.3.2 Neuronal Prophet

NeuralProphet [6] [14] is a successor to Facebook Prophet, which set an industry standard for explainable, scalable, and user-friendly forecasting framework. As already commented hybrid solutions are needed to bridge the gap between interpretable classical methods and scalable deep learning models. NeuralProphet is a hybrid forecasting framework based on PyTorch [15] and trained with standard deep learning methods. NeuralProphet retains the design philosophy of Prophet and provides the same basic model components and incorporate an auto-regression components implemented through a Neuronal Network. It can be formulated as:

$$\boldsymbol{y(t) = T(t) + S(t) + E(t) + F(t) + A(t) + L(t)}$$

where we identify the components trend (*T*), seasonality (*S*) and events (*E*) as in original Prophet. In addition, other components are added: regression effects for future-known exogenous variables (*F*), auto-regression effects based on past observations (*A*) and regression effects for lagged observations of exogenous variables (*L*). All model component modules can be individually configured and combined to compose the model.

3.3.3 Deep AR

DeepAR it is a time series forecasting deep learning-based algorithm develop by Amazon scientist. DeepAR is specifically useful when your dataset contains hundreds of related timeseries (Ex: forecast for different countries). It is also possible to train the model to generate forecasts for new time series that are similar to the ones it has been already trained on. We are not going in detail on the theory under DeepAR models first because of its complexity and second one because is a Deep Learning Architecture build in house by Amazon and having some confidential details. Despite this there is a couple of characteristic worth to be mentioned.

Figure 27 depicts DeepAr fundamentals. DeepAR model architecture is based on Recurrent Neuronal Networks (RNNs). When comparing RNNs with simple Feed forward Neuronal Networks we can spot a slight difference. The hidden units (h) inside RNN have feedback loop, enabling the information to be passed back to the same node multiple times. These hidden units are commonly called recurrent units. On other words, a recurrent neural network, keep a memory of what it has processed in previous steps as the output of a preceding steps is used as an input for the current process step. This "memory" is especially convenient for sequential problems such as time series.

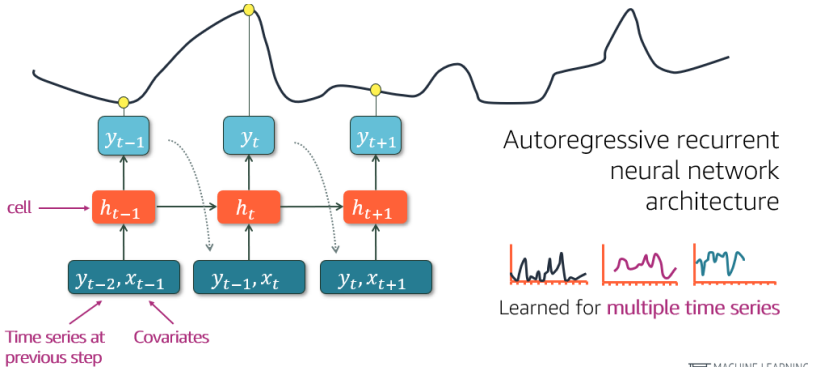


Figure 27 DeepAR algorithm fundamentals

Figure 28 illustrates another interesting characteristic of DeepAR algorithm is that it does not forecast the variable itself but the whole distribution. On other words: the output of the model won't be the number of trucks for time t but the probabilistic distribution of the number of trucks variable for time t. In the case of Gaussian, the likelihood is fully described by two parameters, the mean, μ, and the standard deviation, σ.

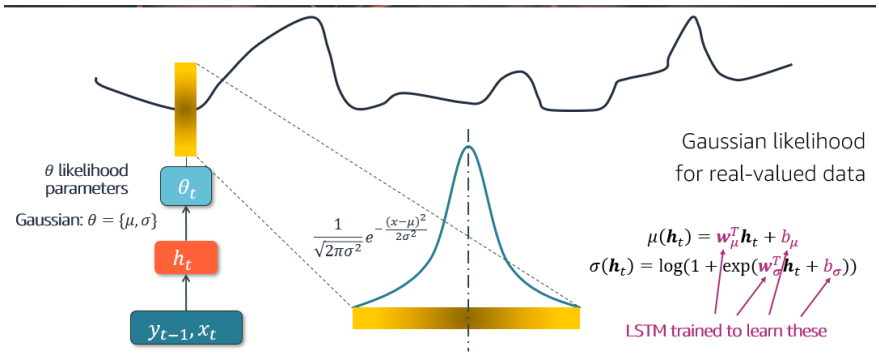


Figure 28 Probabilistic nature of DeepAR forecasting model

After we have covered the main concepts around Time Series Forecasting from a theorical point of view, Chapters 4 and 5 looks to present all the practical work performing a time series forecasting analysis to our concrete dataset/problem.

4 DATA PREPARATION

The practical side of this thesis has been implemented using Python. Python is a high-level, interpreted, general-purpose programming language. Nowadays python is one of the most popular programing languages due to its really simple/clear code readability and the numerous developed specialized libraries for really different areas (Statistic, Data Wrangling, Machine Learning, Deep Learning, Biostatistics, …).

This chapter will cover how we extract and clean the data that we will impute to the time series forecasting models. The second part of the chapter explain the initial exploratory analysis applied to the data. All this Data preparation have been implemented using Pandas, (data manipulation), matplotlib and seaborn (Data Visualization) python libraries.

One of the main challenges of any Data Science/ ML project inside a company is having accurate input data for the model. Even the most advanced model will poorly perform if it is feed with low quality Data ("Crap in, Crap out" principle). In fact, different studies suggest that a data scientist spend around 80 % of his time finding decent quality data and cleaning it. This means that only 20 % time will be used to actually build the ML model.

4.1 Data Sources

In Amazon there are three different Information system/ Databases containing Middle Mile truck Transportation Data (Perfectmile, ORR-Schedule and Reveal). First four weeks of the project were dedicated to 1) engage with the Data Engineers owning these systems, 2) understand how many years of good quality truck related data there was on each of the databases, 3) get the required access permissions and 4) write the SQL queries and logic to extract the required data.

ORR- Schedule is the Database with the highest quality data and the one used for the purpose of this Master Thesis. It contains good truck level transportation data from the 5th April 2015 until the 3rd May 2022.

Initial Data Cleaning

The objective of this second data exploration is to understand what are the max-min dates of good quality data that we have for each country**,** those with missing data and/or low quality will be removed from the analysis. These are the conclusions and transformations performed in the data:

1) Italy, Spain, France, England, Poland and Germany present enough number of truck loads from the 5th April 2015 until the 3rd May 2022.

2) Belgium, Luxembourg, Slovakia, Rumania, Ireland, Norway, Denmark present a really low number of trucks loads, especially for 2015-2018 and have been therefore removed from the analysis.

3) There are some duplicates VRIDS (Vehicle route ID). We have applied a logic in our SQL query remove those duplicates: count (distinct VRIDS).

4.2 Exploratory Data Analysis (EDA)

Exploratory Data Analysis is a key process in every Data Science project. The idea is to investigate the data to find out anomalies and patterns and verify your hypothesis using descriptive statistic and data visualization techniques. We are going to start our exploratory analysis on the aggregated data for the whole European Network and after we will dive deep into each specific country.

4.2.1 Overall Europe EDA

A common approach when performing exploratory data analysis is to formulate some basics hypothesis around the data before perform the actual Exploratory data Analysis and later on perform the actual statistical analysis in order to validate or refuse the initial hypothesis. Our two main Hypothesis previous to the analysis are:

1) There is a general overall growth of Amazon business across the years. Therefore, the time series is expected to have a continuous increasing trend over the years. The expectation would be having the highest relative growth during the first years (2015-2017) and during COVID pandemic (2020-2021).

2) Around seasonality, the expectation would be to have a strong yearly seasonality, with peak periods close to Christmas and Amazon prime week (mid-end December) and valley periods at the beginning of the year and during summer.

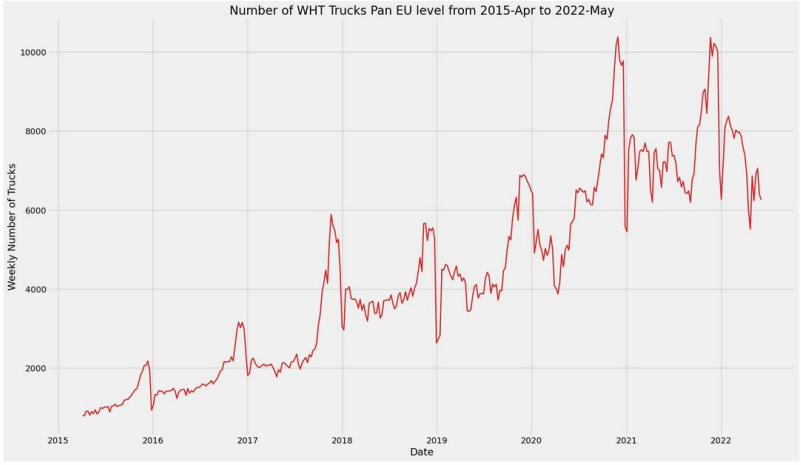


Figure 29 Weekly Number of trucks Apr 2015 – May 2022

As we can observe on Figure 29, as a general comment, there is a clear increasing trend over the years - specially spike from 2017 to 2018 and from 2020 to 2021 On the other hand, the time series present a clear seasonality Off-peak (Jan-Oct) vs Peak (Oct-Dec). Now we will analyses in more detail each of the main components of this time series.

Trend - A boxplot is a really powerful tool to visualize the distribution of a numerical variable in a really simple way. It shows the median, quartiles (Q1-Q3) and outliers on a really visual and straight way. We have built a boxplot to understand the distribution of the weekly number of truck loads per year.

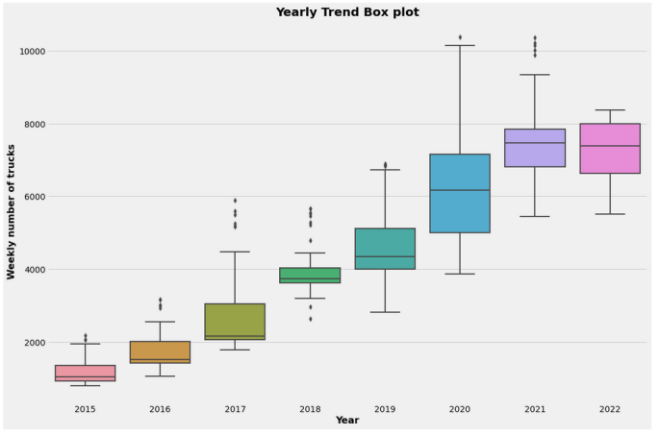


Figure 30 Boxplot for weekly number of trucks per year

As you can observe on the boxplot on Figure 30, there is a consistent increasing Year-over-year (YoY) trend across all the years except for the 2021-2022. This is due to the fact that there is only data until May 2022, so the months with the highest number of trucks loads, November and December, were not computed for 2022. Taking the median as a reference, 2018 and 2020 seem to have the highest YoY increase. In volatility terms, 2020 seems to be the year with the highest variance, having an absolute difference of more than 20000 loads between Q1 and Q3 quantiles.

With the boxplot you get a general idea of the statistical distribution. In order to get a deeper understanding of the data, a YoY monthly delta (%) has been calculated as:

(Number trucks month x(year) – Number trucks month x(year-1))/ Number trucks month x(year-1)

This metric will give an idea of how much the time series grew compared with the same month from the previous year. Results are depicted in Figure 31.

	2016	2017	2018	2019	2020	2021	2022	Total Average
January		64.5%	84.6%	13.2%	14.6%	44.3%	4.8%	37.7%
February		40.8%	81.1%	17.3%	15.1%	43.5%	10.5%	34.7%
March		47.9%	67.1%	21.1%	10.7%	61.6%	4.7%	35.5%
April	64.0%	30.5%	87.9%	11.1%	15.4%	58.2%	-10.5%	36.7%
May	65.4%	51.8%	65.1%	13.4%	25.5%	37.0%	-2.7%	36.5%
June	53.2%	42.2%	71.5%	4.1%	62.4%	21.8%		42.6%
July	50.0%	38.0%	72.7%	17.4%	53.8%	4.4%		39.4%
August	64.6%	34.7%	70.5%	2.0%	54.9%	4.7%		38.6%
September	61.9%	33.8%	43.0%	27.6%	49.4%	2.1%		36.3%
October	53.8%	87.1%	12.0%	36.4%	32.3%	4.6%		37.7%
November	52.8%	91.4%	1.1%	22.5%	43.5%	0.9%		35.4%
December	52.4%	66.9%	2.8%	22.0%	54.0%	6.3%		34.1%
Total Average	57.6%	52.5%	55.0%	17.3%	36.0%	24.1%	1.4%	34.8%

Figure 31 YoY monthly delta

There is a clear Overall YoY increasing trend since 2015. The average YoY monthly increase since 2015 is 34.8%. Additionally, it can be observed:

- COVID has a clear growing impact with a 50.7 % average monthly YoY increase between June 2020 and April 2021. This was the main lockdown period causing our customers to buy more products from home.

- 2016 YoY increase is stable and consistent across the year. It is mainly due to the fact that the European Logistic Network was almost new in 2015 and suffer a huge relative expansion from 2015 to 2016.

- 2017 (52.5% Average YoY increase) and 2018 (55 %) is mainly drive by Oct/2018 – Aug/2019 time period. It is mainly due to the expansion of the German Network with the launch of 4 new stations. See Figure 8.

- 2019 (17.3 %) and 2022 (1.4%) were relatively flat years in terms of YoY growth.

- A really soft or even decreasing YoY monthly delta (%) for the first months of 2022.

Seasonality **–** there are 2 useful data visualizations to help understanding the time series seasonality: 1) line chart with the number of weekly trucks comparing different years and 2) a boxplot to better understand the distribution of the weekly number of trucks across the different month of the years. These two representations are shown in Figure 32

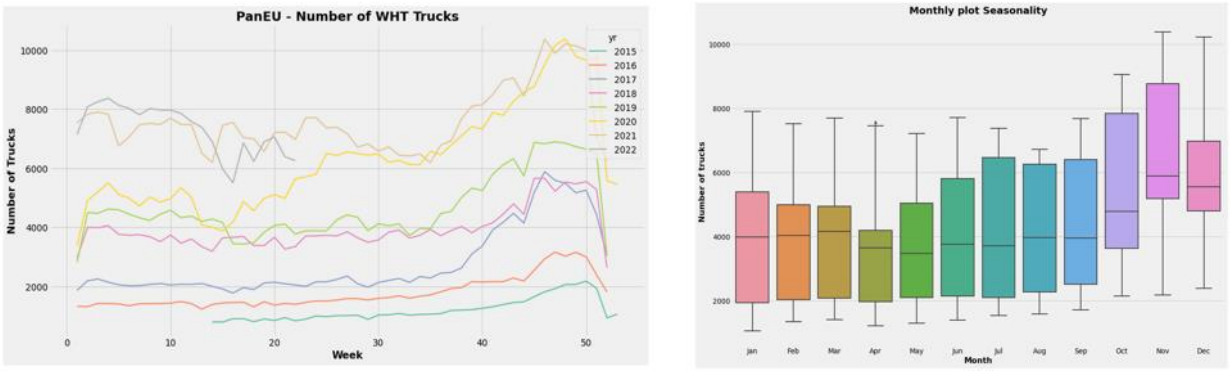


Figure 32 Line chart and Boxplot for weekly number of trucks per month

Again, in order to dive deep in the data and get a better quantify detail the seasonality, we have calculated the Month-over-Month (MoM) delta (%):

(Number of trucks month x– Number of trucks month x-1)/ Number of trucks month x-1

This metric will give and idea on how much the number of trucks grew when compared with the previous month of the same year and therefore will indicate where the big increase/decrease take place during the year. Results are depicted in Figure 33

	2015	2016	2017	2018	2019	2020	2021	2022	Monthly Average
January		-26.3%	-20.4%	-12.0%	-3.0%	-8.9%	-14.7%	-15.9%	-14.5%
February		3.5%	-11.4%	-13.1%	-10.0%	-9.6%	-10.1%	-5.2%	-8.0%
March		8.3%	13.7%	4.9%	8.3%	4.2%	17.3%	11.2%	9.7%
April		-3.6%	-14.9%	-4.3%	-12.2%	-8.5%	-10.4%	-23.4%	-11.0%
May	-0.1%	0.8%	17.2%	3.0%	5.1%	14.4%	-1.0%	7.6%	5.9%
June	14.0%	5.6%	-1.1%	2.7%	-5.7%	22.0%	8.5%		6.6%
July	6.7%	4.4%	1.4%	2.0%	15.0%	8.9%	-6.7%		4.5%
August	0.6%	10.4%	7.7%	6.4%	-7.6%	-6.9%	-6.6%		0.6%
September	14.3%	12.4%	11.7%	-6.3%	17.2%	13.1%	10.3%		10.4%
October	18.2%	12.3%	57.1%	23.0%	31.5%	16.4%	19.2%		25.4%
November	29.0%	28.1%	31.1%	18.2%	6.2%	15.2%	11.1%		19.9%
December	-3.3%	-3.5%	-15.9%	-14.4%	-14.8%	-8.6%	-3.7%		-9.2%
Total Average	9.9%	4.4%	6.3%	0.8%	2.5%	4.3%	1.1%	-5.1%	3.4%

Figure 33 Month-over-Month (MoM) delta

It can be observed that the series presents a clear seasonality Off-Peak Period (Jan-Sept) vs Peak Period (October-December). The Number of trucks is pretty stable for the off-Peak period with a smooth valley in April (-11.0 % average MoM decrease). We can see how this number start increasing in October (+25.4 % average MoM Increase) with the highest number of trucks loads in November and December. The explanation is clear. There are 2 clearly differentiated periods for the majority of e-commerce retailers:

1) Peak Period: November and December where the majority of events/promotions take place. It is important to note that the distribution of this truck is higher during November (vs December) because main events (Black Friday, Prime day) take place during this month.

2) Off-Peak Period: rest of the year with a more or less stable demand.

4.2.2 Country Level EDA

After a general analysis on the overall European Network, this section deep dive on drill down to the country level granularity. Figure 34 shows weekly number of trucks per country.

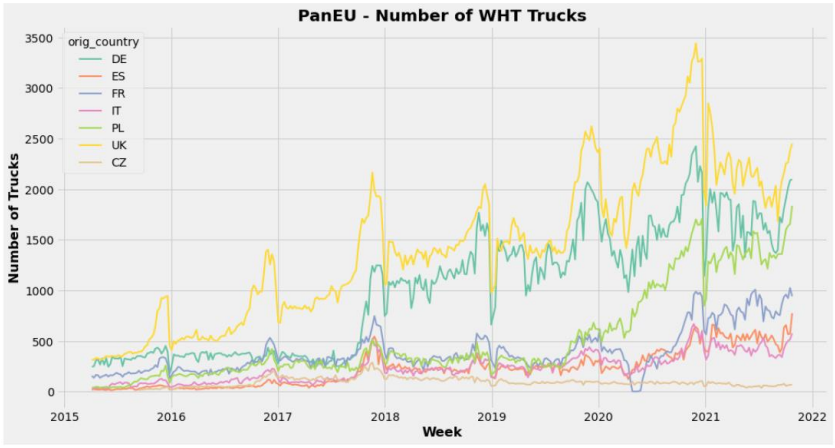


Figure 34 Weekly number of trucks per country

Trend - In general lines, there is a positive YoY trend for the majority of the countries as it can be observed in Figure 34. Some interesting points to be commented when detailing on the country granularity:

- United Kingdom (UK) is the oldest country in the network. The number of Amazon Stations did not grow much since 2015 making it the country with the more stable/consistent trend.

- Spain (ES) and Germany (DE) and Poland (PL) are younger countries in the Network. There was a strategic decision to launch new Stations on those territories on 2017 (ES and GE) and 2020 (PL) resulting in a huge YoY number of truck loads increase - Spain (+307 % average YoY increase from October 2017-May 2018), Germany (+231% YoY average increase November 2017- September 2018) and Poland (+185,2 % YoY average increase April 2020-April 2021)

- Check Republic (CZ) is one of the most complex countries in the Network. There was an initial business decision of expand the number of stations on 2016 (+306.9 % increase Sept 2016- March 2017). However, in 2016 due to political reasons it was decided to smoothly remove business from Check Republic (-17.7 % average YoY decrease since 2019).

Seasonality - In general lines all the countries follow the same seasonality pattern previously described. The number of trucks remains pretty stable from January to September (off-peak) with a small valley in April (month with a negative MoM trend for all the countries) and it starts to peak up at the end of September, being October and November the months with the highest MoM relative increase.

Outliers – As we can observe on Figure 35, the data shows a clear subsequence global outlier for France time series. It can be observed that there is a set of 4 consecutive weeks without any truck loads transported from a French Amazon site (Week 17 2020- Week 21 2020). This is due to an Amazon driver's strike that took place in France during these weeks.

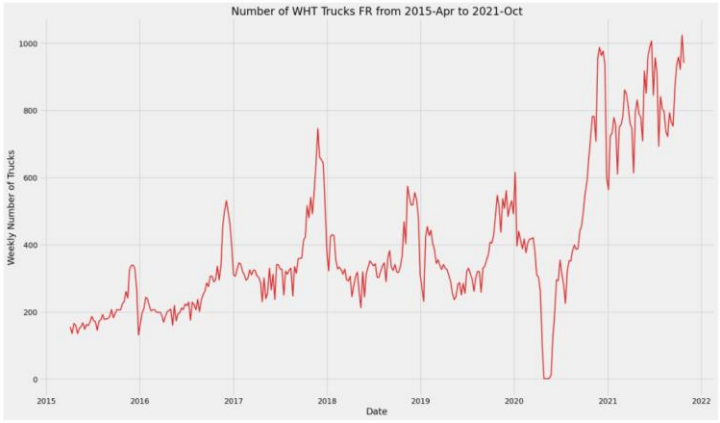


Figure 35 Weekly number of trucks per year in France

As commented before, these observations are inconsistent with the remainder of the series and can drastically influence our time series forecast for France. There are different techniques for treating these outliers: 1) Remove the outliers; 2) Mean/Median Imputation; and 3) Interpolation using previous and next values. For this specific use case, the interpolation approach was selected to calculate and substitute the missing 4 points:

$$y_{INT1} = y_{PREV} + \frac{1}{4}(y_{POST} - y_{PREV})$$

4.3 Features

Features are extra variables or events that can impact time series behavior. It is therefore important to try to properly estimate them and build models that take those into account. This small section describes the work done on feature engineering and modelling choices. To avoid making the problem to complex, we have considered the following features in our models:

- **Peak and Amazon Events -**In order to model the high volumes associated to end of year holidays, we added a feature to model the ramp up before the event and romp down once the event has finished. Another important event included in the model is Prime Day, a one to two-day event, with high sales on many products for Amazon Prime members. It usually takes place in July (except for 2020 because of COVID). The discounts Amazon offers are thus limited in time, creating a strong, bursty demand peak.

- **COVID 19 -** During the first months of COVID it is noticed a strong drop in demand. For the model not to assume any seasonal or trend effect we introduce a COVID flag feature with a period arbitrarily fixed at 2020-02-15 to 2020-06-01.

There are other Features that would be interesting to explore but would add a higher complexity to the models and are not therefore object of this work:

- **Site Openings** - when a new site opens, it follows a gradual volume increase pattern until it reaches full throughput. As we are forecasting at cluster level, the model might not correctly account for site openings, which is why would be interesting adding a site opening feature. The idea would be to try to flag/estimate the ramp up period of a site.

- **Number of actives clusters** -The cluster size feature is trivial, it models the number of sites included in the forecast. This could be as well an interesting variable to add into the model.

5 RESULTS AND MODEL COMPARISON

After we have covered the main Time Series forecasting models from a theorical point of view, this chapter presents all the practical work performed to implement these models to our concrete clean dataset, reports concrete results for each of them and performs a comparison among them. The chapter includes as well some explanation around the different ML python libraries used and how to configure their main parameters and hyperparameters to maximize the performance of the model.

The chapter is divided into two parts. The first one covers an explanation on how to implement each kind of models using python packages. This section will detail how to vary the models hyperparameters, include the different features and try to find the most performing model for each of the model types (most performing ARIMA model, most performing Prophet model…). This top performing model will be used to perform the Country level forecasting. Please, note that the comparison will be established by comparing model WAPE (%) for three different training/test splits

- Period 1 (P1) - Training: April 2015 - April 2021 | Test: May 2021 - May 2022
- Period 2 (P2) - Training: April 2015 - Jan 2021 | Test: Feb 2021 - Feb 2022
- Period 3 (P3) - Training: April 2015 – April 2020 | Test: May 2020 – May 2021

The second part of the chapter covers a more detailed comparison of the performance of the top performer models selected on the basis of the first part.

All the Practical implementation was done in Sagemaker - Amazon SageMaker is a managed service in the AWS (Amazon Web Service) cloud. It provides the tools to build, train and deploy machine learning (ML) models for predictive analytics applications. We have implemented everything in Jupyter notebooks supported by Sagemaker. The main advantages are that 1) we have access to the majority of Python libraries without installing them and 2) we can run our models on the cloud instead of in our local machine resulting in an efficiency and time consumption improvement. For future stages, it offers also the possibility of Productionalize and integrate our models with other AWS components.

5.1 Implementation and Results

5.1.1 Simple Methods

As we know, there are some forecasting methods extremely simple that can be used as baseline/benchmarks to compare other models against: 1) Naïve Method, 2) Moving Average Method, 3) Drift Method and 4) Seasonal Naive Method. There is not specific statistical library in python to directly implement these methods. We have implemented them in Python with the only support of NumPy library. Figure 36 depicts obtained results.

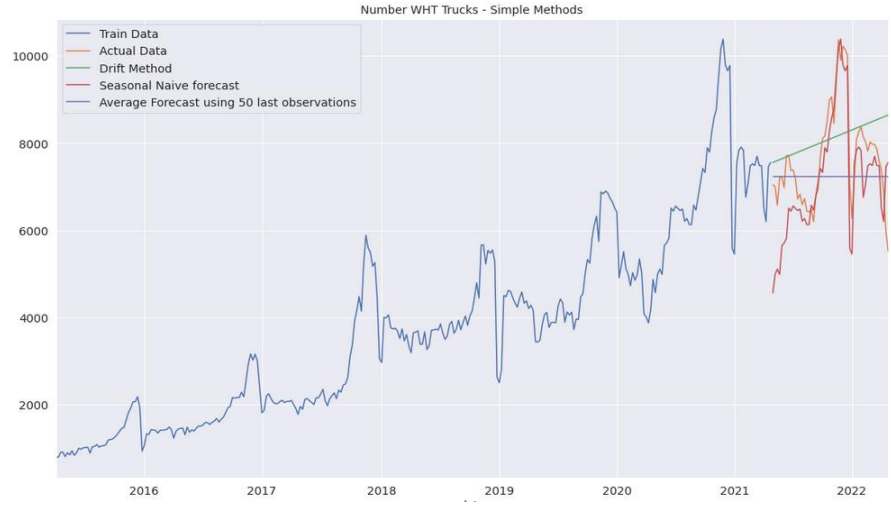
We have evaluated them using the WAPE Evaluation metric for the three already specified periods (P1, P2, P3). Results are reported in Table 2.

Table 2 WAPE results for simple methods

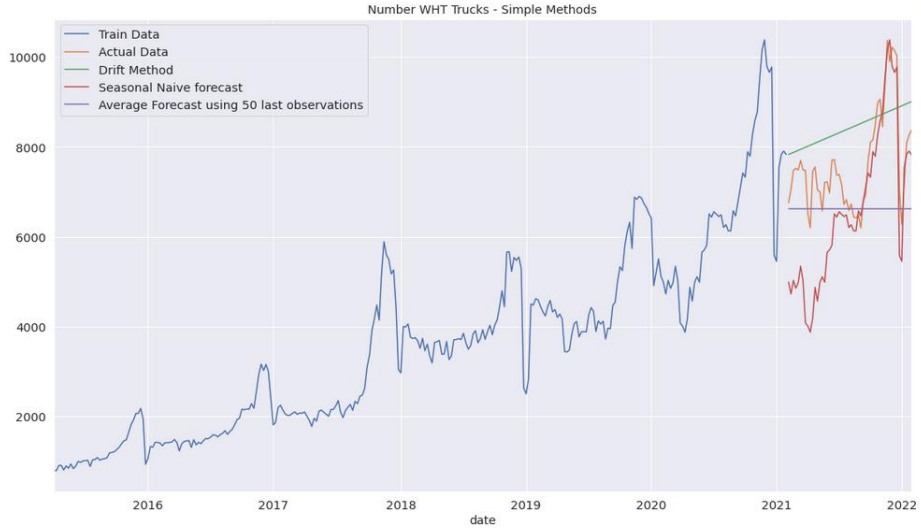
Method	P1 WAPE (%)	P2 WAPE (%)	P3 WAPE (%)
Naïve Method	11.54	12.08	31.98
Moving Average Method (50 previous weeks)	11.67	14.07	30.98
Drift Method	12.65	14.39	32.60
Seasonal Naive Method	9.94	14.95	26.48

On Table 2, we can observe how these methods seems to work relatively fine for Period 1 and Period 2. Despite this, it is important to note that these kinds of methods are rarely used on real applications as they are too simplistic and only make some sense when forecasting really stables time series in which previous year is really similar to this one. On this regard, we can observe that the second semester of 2020 is really similar to the second semester of 2021 and the first semester of 2021 is really similar to the first semester of 2022 - That's why these methods seem to perform fine (in terms of WAPE) for P1 and P2.

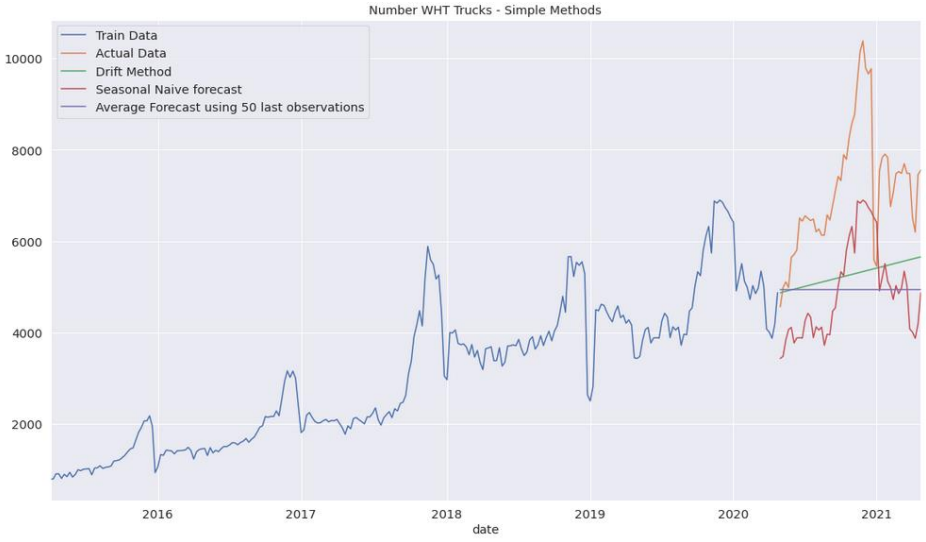
An interesting analysis in order to understand the limitation of these methods is to observe the really poor forecast (WAPE) produced for Period 3. We can observe that the difference between 2019 and 2021 is much more significant than between 2020 and 2021. Because of this, the forecasted values starting on April 2021 (Period 3) are really poor.



(a)



(b)



(c)

Figure 36 Simple methods results for (a) P1, (b) P2 and (c) P3

5.1.2 ETS Models

Statsmodel is one of the most well-known python libraries that provides classes and functions to implement various statistical models and tests. Python allows to implement exponential smoothing models by using the *ETSModel* module inside the *statsmodel* library. Similarly, to the majority of ML libraries, ETSModel module allows to both train the model and use that model to make the required predictions

The *ETSModel* function allows to train the model by specifying the training data (train), the trend (None, Additive, Multiplicative), the seasonality (None, Additive, Multiplicative) and the error (Additive, Multiplicative) as you can see on script 1 in Appendix.

The *get_prediction* method is used to make the forecast by simply specifying the start and end date. As we already detailed, ETS models generate not only a point forecast but an entire probability distribution for the future forecasted horizon. As we can observe on the script 2 in Appendix, in order to calculate the WAPE, we can simply take the mean of this probability distribution:

We have implemented a total of 18 ETS models using the different combinations of Error, Trend and Seasonality parameters configuration, and compared them based on point forecasting evaluation metric (WAPE) for the 3 already introduced periods. Table 3 and Table 4 depict WAPE obtained results. The first one corresponds to the nine models with additive error and the second one to the other nine models which use multiplicative error. Best and worse results for each period are shown in orange and red respectively.

Table 3 WAPE (%) results for ETS models with additive error P1/P2/P3

Seasonal Trend	N	A	M
N	**11.54** /12.08 / 31.8	15.12 / 16.67 / **32.32**	15.41 / 7.82 / 19.81
A	12.86 / 14.54 / 26.31	15.12 /17.86 / 24.86	**17.71** / 11.96 / **14.76**
Ad	**11.54** / 12.09 / 31.98	15.01 / **7.81** / 30.57	15.08 / **7.81** / 19.83

Table 4 WAPE (%) results for ETS models with multiplicative error P1/P2/P3

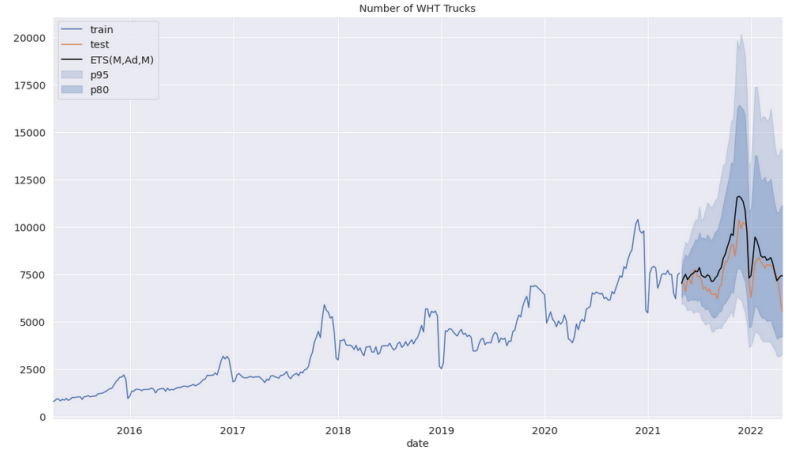
Seasonal Trend	N	A	M
N	11.56 / 12.08 / 32.33	11.70 / 13.32 / 31.43	9.07 / 9.98 / **33.11**
A	11.30 / 12.53 / 31.98	**17.04** / **22.59** / 26.38	14.23 / **6.94** / **21.78**
Ad	11.53 /17.64 / 31.93	16.26 / 17.10 / 31.92	**9.04** / 7.44 / 27.68

From this tables, it can be observed that overall the Multiplicative Damped Holt Winters with Multiplicative error (ETS (M, Ad, M)) is the top performing model with a 9.04% and 7.44% WAPE respectively for P1 and P2, but does not perform so well for P3 (27.68%).

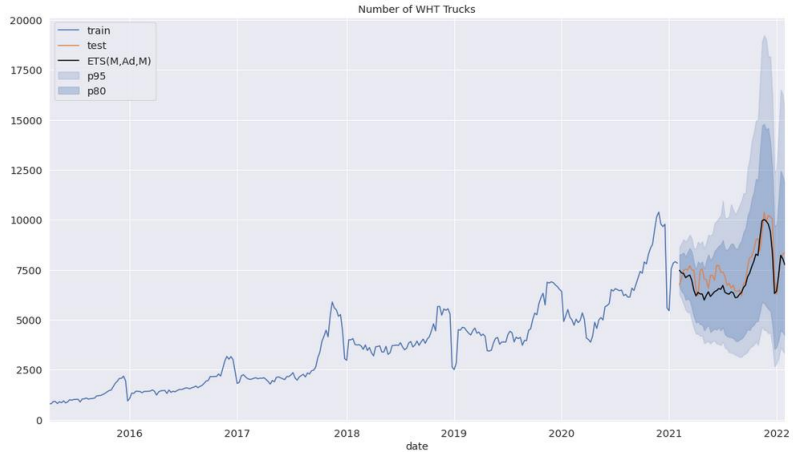
Figure 37 depicts the performance of this model for P1, P2 and P3. We can clearly graphically confirm how this model works for P1 and P2 and does not for P3.

Effectively, despite the effect of the damped trend parameter, the model tends to underpredict the Number of Trucks for P3 (May 2020 – May 2021). As we commented on Chapter 4, COVID has a clear growing impact with a 50.7% average monthly YoY increment between June 2020 and April 2021. For P3, models use data until April 2020 (before COVID started) in order to forecast the number of trucks from May 2020 to May 2021. In a nutshell, as the models were not trained using data during COVID, they are not able to integrate the effect of it, and therefore, they will tend to underestimate

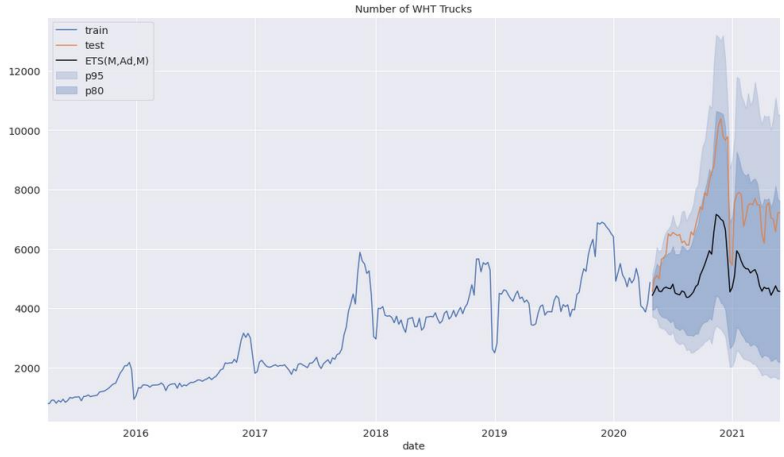
the number of trucks. As we commented in Chapter 4, it is possible to add a COVID Feature to help the model properly understand, integrate and forecast the effect of COVID. However, ETS models do not allow to integrate external regressors/features. This functionality will be therefore explored in next models.



(a)



(b)



(c)

Figure 37 ETS(M,Ad,M) results for (a) P1, (b) P2 and (c) P3

Country Analysis

As we already mentioned, the top performing model (ETS(M,Ad,M)) will be used to perform the Country level forecasting. We will limit our forecasting simulations to Period 1 (Training: April 2015 - April 2021 | Test: May 2021 - May 2022). Table 5 shows the WAPE for each specific country.

Table 5 WAPE (%) results of ETS(M,Ad,M) model for period P1 by country

Country	WAPE (%)
UK	7.85
ES	15.77
FR	16.31
IT	20.06
DE	9.04
PL	11.41
CZ	20.52

We can observe that the country level accuracy is lower than the Global Europe aggregated one. On general lines, aggregated forecasting has a smaller standard deviation of error relative to the mean than disaggregated one and therefore a higher accuracy. More aggregated data is inherently less noisy than low-level data because noise cancels itself out in the process of aggregation. On this specific case aggregation will help to mitigate and compensate specific countries singularities/outliers.

On other side, we can observe how the model works better for UK and DE than for the rest of the countries. The explanation rest on the same concept, UK and DE transportation Network is bigger and contains a higher number of Fulfillment center and volume of trucks than the other countries – this cause having aggregated data from a higher number of sources cancelling noise, reducing the standard deviation and improving accuracy.

5.1.3 ARIMA Models

Python allows to implement ARIMA models by using *tsa* module inside the *statsmodel* library. The *tsa.arima.model* module is used to generate ARIMA models and the *tsa.statespace.sarimax* is used to generate Seasonal Arima modes. We can observe how to import these libraries on Script 3 in Appendix.

For this concrete time series analysis, we will just deploy Seasonal Arima Models. As Script 4 in Appendix depict,s Tsa ARIMAX module allows to train the model by specifying the training data (train), and the combination of Seasonal/non-Seasonal AR order, differencing and MA order.

The *get_prediction* method is used to make the forecast by simply specifying the start and end dates. *Tsa.statasmodel* generates not only a point forecast but an entire probability distribution for the future forecast horizon. Again, as we can see on Script 5 in Appendix, in order to calculate the WAPE ARIMA allows to simply use the mean of the probability distribution.

There is an almost unlimited combination of parameters for ARIMA model, so, we cannot deploy them all. As we can observe on Script 6 in Appendix, we have created a manual 'grid search' to iterate through different combinations of model parameters and select the top performing using the WAPE

as criteria.

The top performing model was the SARIMAX (1,1,3) x (1,1,2). Table 6 shows WAPE results for this model. As ETS models, ARIMA gives a really solid performance for P1 (6.91%) and P2 (8.74%) and tents to under-predict on P3 (20.54%).

Table 6 WAPE (%) results for SARIMAX (1,1,3) x (1,1,2)

Seasonal Trend	WAPE (%)
P1	6.91
P2	8.74
P3	20.54

Figure 38 depicts the output forecast of this model for P1, P2 and P3. We can clearly graphically confirm how this model works for P1 and P2 and does not for P3. As explained on ETS models' section, on P3 ARIMA model does not integrate COVID effect and it tends to underpredict the number of trucks for P3.

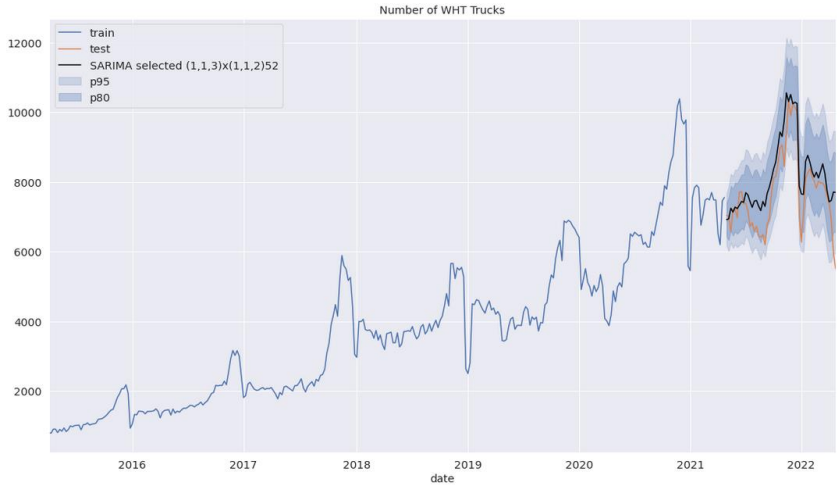
One of the main advantages of ARIMA libraries compared to ETS is that ARIMA allows you to take into consideration and incorporate extra external regressors to our models. As Script 7 in Appendix shows Python SARIMAX Library allows you to incorporate these by using the *exog* parameter. *Exog* parameter requires to provide exactly enough out-of-train values for the external variables. On other words, it is necessary to know the value of these external regressors for the test or the forecasting data.

We have incorporated both Amazon Events and COVID as external regressors to our baseline top performing model and evaluated the WAPE for the three periods to understand if these external variables are helping to improve the model performance.

Amazon Events - The following Amazon events have been incorporated to the model: 1) Cybermonday, 2) Primeday and 3) BlackFriday. Opposite of what it was expected, it can be observed in Table 7 that the inclusion of these amazon events feature it is reducing model accuracy (WAPE%) for P1 (-4.33%) and P3 (-4.05%).

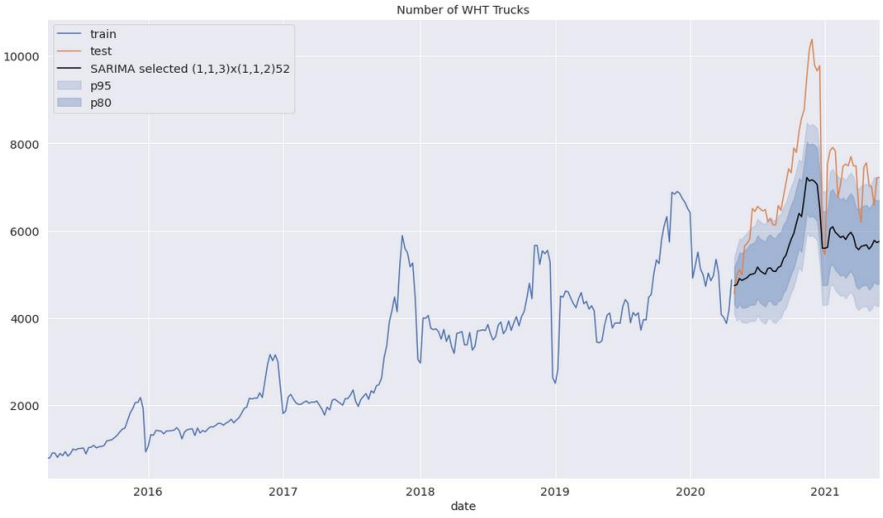
Table 7 Comparison of WAPE (%) results with and without Peak and Amazon Events

Seasonal Trend	Without Peak and Amazon Events	With Peak and Amazon Events
P1	6.91	11.24
P2	8.74	8.74
P3	20.54	24.55



(a)

(b)



(c)

Figure 38 SARIMAX (1,1,3) x (1,1,2) performance for (a) P1, (b) P2 and (c) P3

The original SARIMAX model without the influence of any external regressors is already really good into understanding the time series seasonality. As we can observe on Figure 39, Amazon events are normally aligned with peaks on the number of trucks. However, sometimes these events are not aligned with time series peak (for example, Primeday 2018-2021 or Cybermonday 2018). These signals are confusing the model and reducing its ability to predict the time series seasonality leading to lower performance/accuracy results.

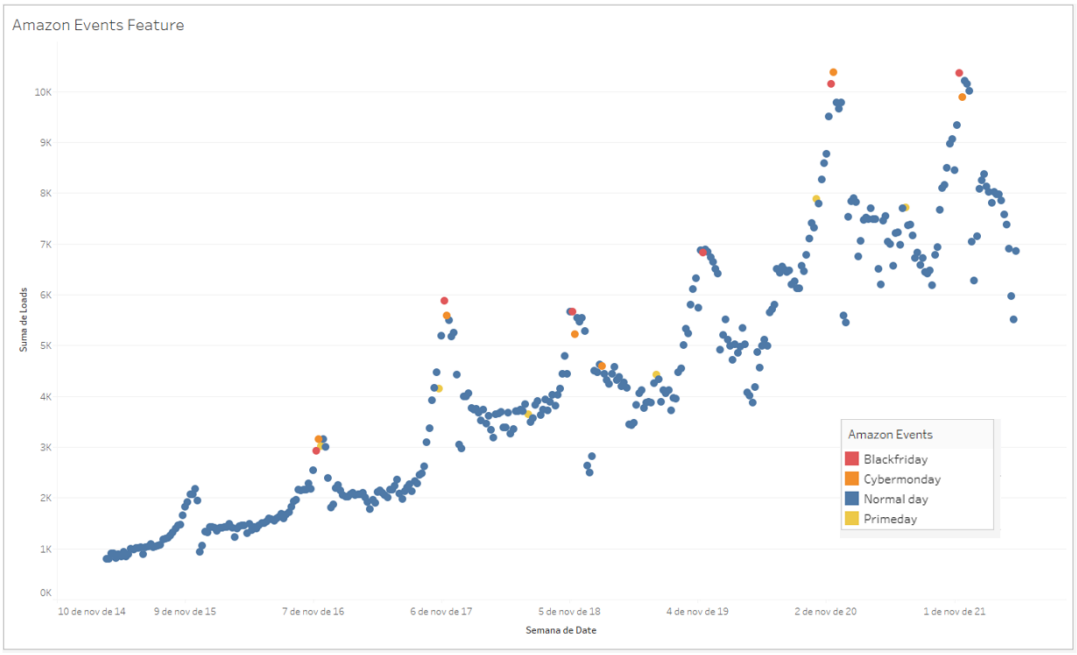


Figure 39 Number of trucks in time. Amazon events pointed out.

COVID Feature

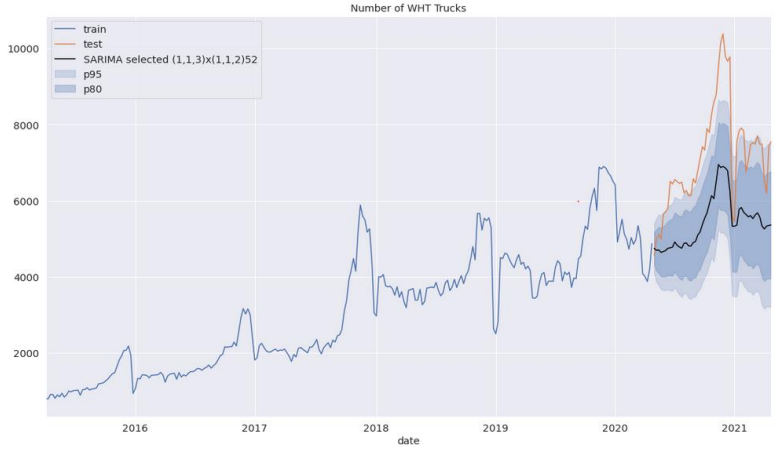
As we already commented several times our model is having some difficulties to predict the high increasing trend caused by the effect of COVID between mid-2020 and mid-2021. On this regard, we have tried to incorporate the COVID as an external regressor. In order to simplify we have considered only the period of highest COVID impact in Europe (2020-05-01 – 2021-05-01) and we have modeled this COVID regressor as a binary flag variable (1 or 0).

Table 8 Comparison of WAPE (%) results with and without COVID feature

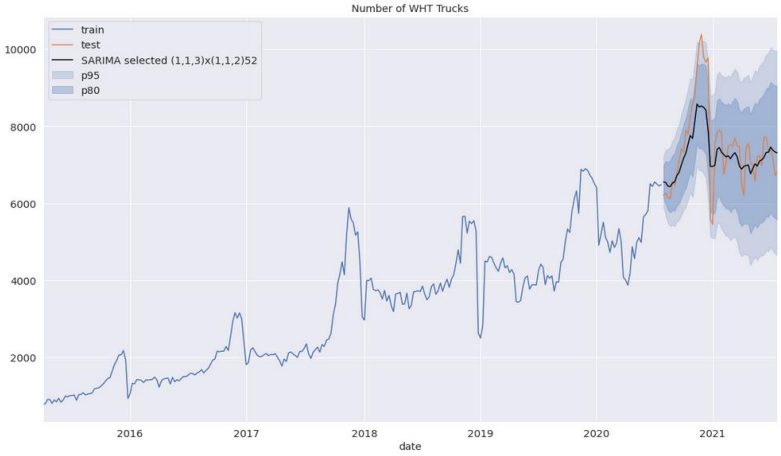
Seasonal Trend	Without COVID feature	With COVID feature
P1	6.91	6.30
P2	8.74	7.73
P3	20.54	20.56

As we can observe on Table 8, on the one hand, the inclusion of these amazon events feature is improving model accuracy (WAPE%) for P1 (+0.61%) and P2 (+1.01%). The inclusion of this variable is helping the model to understand that the big YoY increasing trend during the second part of 2020 and first part of 2021 is something mainly driven by Covid and not extensible to the second part of 2021 and 2022. On the other hand, the inclusion of this feature does not have not have any impact on the accuracy of the model for predicting P3 as the model was not trained with COVID data for P3.

An interesting experiment is to extend the training period in a few months so the model is trained with some COVID data. We have performed a new experiment expanding the training end date (P3': Training: April 2015 – Aug 2020 | Test: Aug 2020 – Aug 2021). As we can observe on Figure 40, when we train the model with some COVID data, ARIMA is now able to properly integrate COVID effect, avoiding underprediction and critically improving the accuracy (WAPE) from 24.18% to 7.07%.



(a) P3 - Training: April 2015 – April 2020 | Test: May 2020 – May 2021



(b) P3' -Training: April 2015 – Aug 2020 | Test: Aug 2020 – Aug 2021

Figure 40 SARIMAX(1,1,3) x (1,1,2) COVID feature performance for P3 and P3'

Country Analysis

We have performed a country level experiment, finding the best model parameters (based on the WAPE) for each of the countries and comparing the performance with the same model incorporating the COVID external Regressors for P1.

As we can observe in Table 9, and we commented previously, the accuracy of the country level forecast is lower than for aggregated Europe data. On other side, this accuracy of the models is higher for UK and DE and reasonably solid across the rest of the countries with the exception of CZ. Finally, it is important to note that in general lines the model incorporating the COVID regressors possess a better performance – specially for ES (+0.81%) and CZ (+1.57%).

Table 9 Comparison of WAPE (%) results with and without COVID feature for P1 by country

Country	WAPE (%)	With COVID feature
UK	8.25	8.18
ES	14.77	13.96
FR	14.25	14.14
IT	13.75	13.67
DE	9.36	9.36
PL	11.23	11.05
CZ	22.35	20.78

5.1.4 Prophet Models

As showed in Script 8 in Appendix, Python allows to implement Prophet models by using *Prophet* module inside the *fbprophet* library.For this concrete time series analysis, we will first deploy a simple Prophet Model. As in previous models, firstly you need to specify models' parameters and train the model using the training data as detailed on Script 9 in Appendix.

In order to make predictions, it is first required to create a *dataframe* with a column *ds* containing the dates for which a prediction is to be made. As we can observe on Script 10 in Appendix, in order to do so, it is possible to 1) use the *Prophet.make_future_dataframe method* and 2) used the predict method to make the forecast. As previous models, Prophet generate not only a point forecast but an entire probability distribution for the future forecast horizon. In order to calculate the WAPE we will simply use the mean of this probability distribution.

Similarly, to ARIMA and as we can observe on Table 10, Prophet models present a large number of parameters that can be configured.

Table 10 Prophet Hyperparameters

NN Hyperparameter	Description
Changepoint Prior Scale	It determines the flexibility of the trend, and in particular how much the trend changes at the trend changepoints. if it is too small, the trend will be underfit and it is too big it will overfit.
Changepoints Range	This is the proportion of the history in which the trend is allowed to change. This defaults to 0.8, 80% of the history, meaning the model will not fit any trend changes in the last 20% of the time series.
Seasonality Prior Scale	This parameter controls the flexibility of the seasonality. Similarly, a large value allows the seasonality to fit large fluctuations, a small value shrinks the magnitude of the seasonality.
Holidays Prior Scale	This controls flexibility to fit holiday effects.
Seasonality mode	Options are ['additive', 'multiplicative']. Default is 'additive', but many business time series will have multiplicative seasonality.

We have therefore created again a manual 'grid search' to iterate through different combination of this parameters and select the top performing using again WAPE as criteria as we can observe on Script 11 in Appendix.

The top performing model was the Prophet with *changepoint range*= 0.7, *changepoint_prior_scale* =0.004, *seasonality_prior_scale* =0.1 *seasonality_mode*= 'additive') Table 11 shows WAPE results for this model. Again, we can observe really solid performance for P1 (6.45%) and P2 (6.95%) and tends to under-predict on P3 (18.75 %).

Table 11 WAPE (%) results for top performing Prophet model

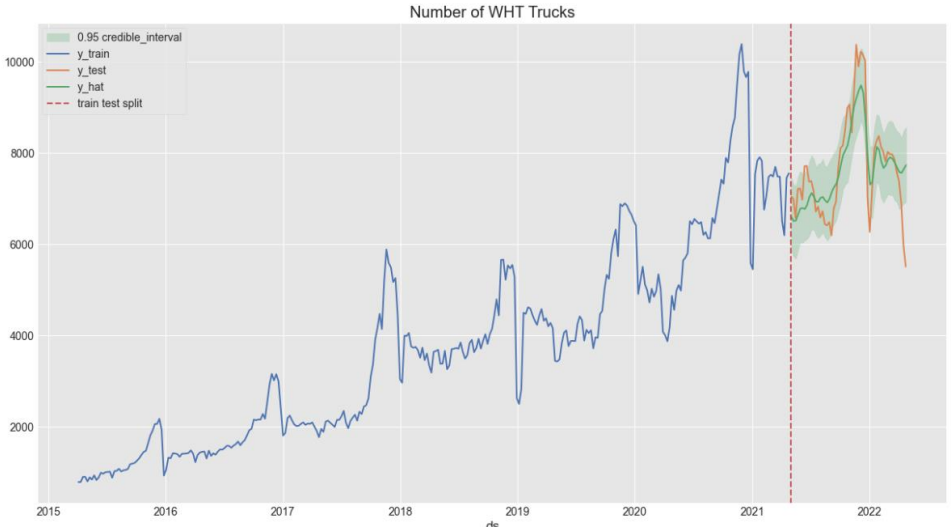
Seasonal Trend	WAPE (%)
P1	6.45
P2	6.95
P3	18.75

Figure 41 depicts the output forecast of this model for P1, P2 and P3. We can clearly graphically confirm how this model works for P1 and P2 and does not for P3.

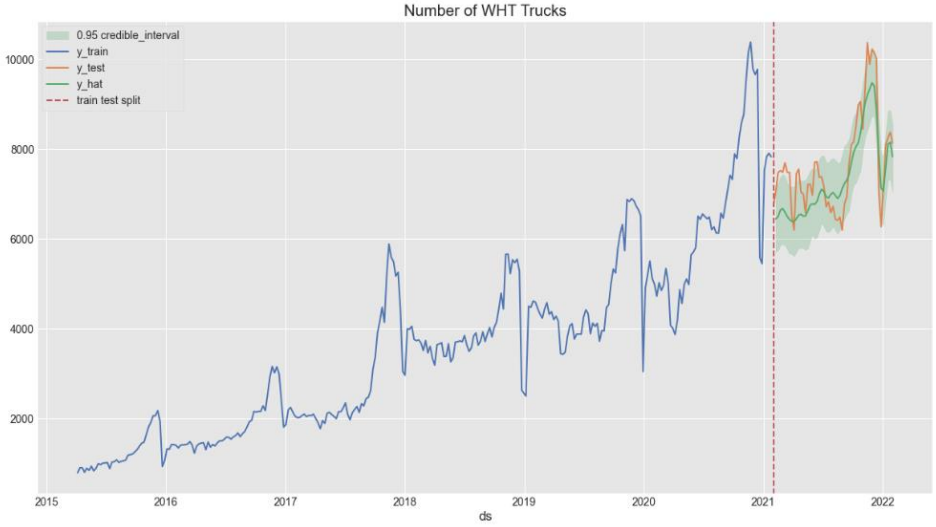
Features

Prophet library also offer the possibility of incorporating external regressors to our model using a holiday parameter.The model required a holiday *dataframe* with a specific format as an input. This dataframe must have two columns (*holiday* and *ds*) and a row for each occurrence of the holiday. It must include all occurrences of the holiday, both in the past (back as far as the historical data go) and in the future (out as far as the forecast is being made). It is also possible to extend the holiday impact to the future some periods before or after the actual event by using the *lower_window* and *upper_window* parameters. As for ARIMA models, we have incorporated the effect of Amazon events and COVID effect to the model. By including these external regressors in our model we can clearly observe that this regressors are not having any impact on the model prediction.

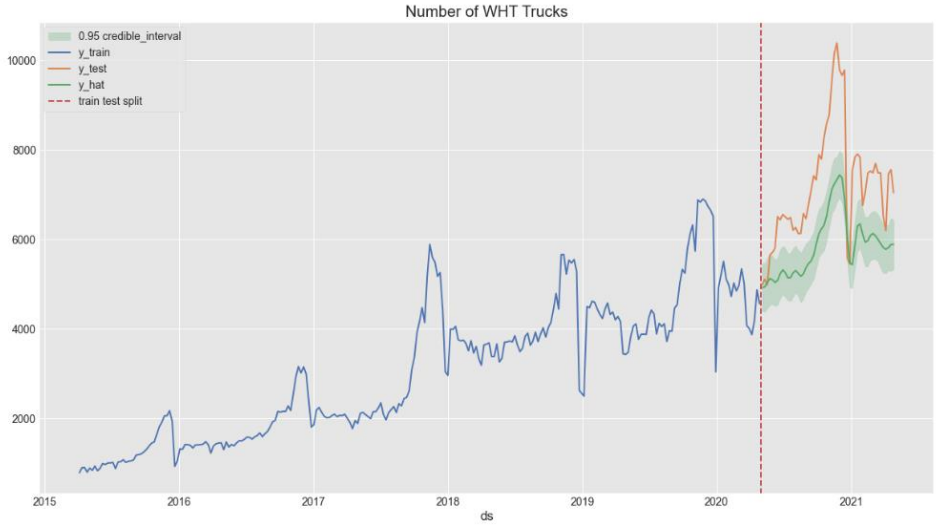
Effectively, Prophet library has a bug/defect that does not allow to include holiday effect in the model when the frequency of the data is weekly. From the research done, Facebook engineers are working on fixing this, it will take some time and will be included in futures versions of the library. A possible workaround would be to desegregate the data to a daily level, perform the forecast on that daily granularity (for which prophet model is able to include the holiday effect) and after that aggregate the forecasted data on a daily level. This analysis is not on the scope of this work.



(a)



(b)



(c)

Figure 41 Prophet forecast for (a) P1, (b) P2 and (c) P3

Country level Analysis

We have again found the best model parameters based on WAPE for each of the countries and P1. Table 12 depicts obtained results. In concordance with previous models, Prophet performance is relatively solid across all the countries with exception of CZ (32.71% WAPE). It is also interesting to note that the performance on country level is consistently lower when compared with the performance of the model applied on the overall Europe data (6.45% WAPE).

Table 12 Top performing Prophet model and WAPE (%) for P1 by country

Country	WAPE P1 (%)
UK	8.65
ES	10.17
FR	12.64
IT	9.19
DE	9.33
PL	9.26
CZ	32.71

Figure 42 depicts the model performance for CZ.

Low forecasting WAPE accuracy for CZ country is a common problem for our models: ETS (20.45%), ARIMA (22.35%). The time series trend is fairly stable and the model fails to anticipate the big descend suffered from 2020 to 2021. It is important to note that this descend is driven by a strategic decision of removing certain fulfillment centers from CZ.

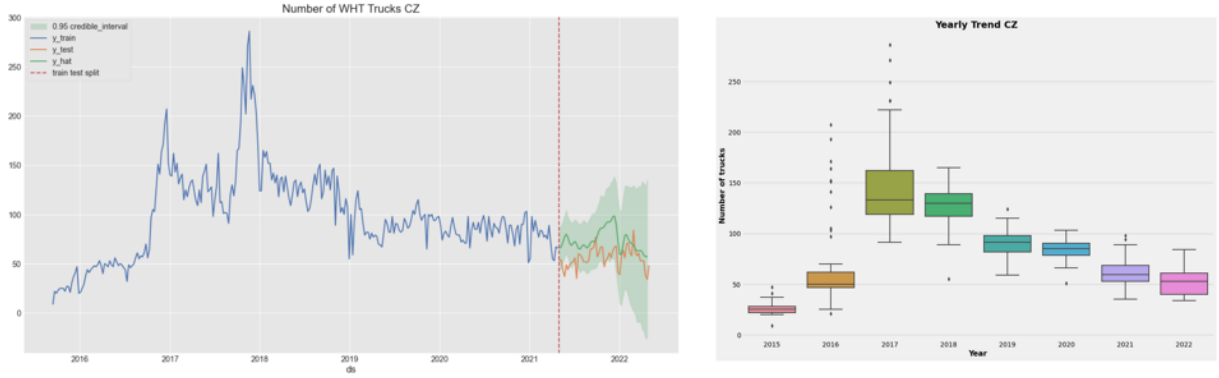


Figure 42 Best performing Prophet model for CZ P1 forecast and weekly number of trucks boxplot

5.1.5 Neuronal Prophet

As explained on Chapter 3 of this document, Neuronal Prophet is a variation of the original Prophet model that mainly incorporates an Auto-regressive component to the model equation. Auto-regression is handled using an implementation of AR-Net, an Auto-Regressive Feed-Forward Neural Network for time series. Python allows to implement Neuronal Prophet models by using *NeuralProphet* module inside the *neuralprophet* library as Script 12 in Appendix shows.

As in previous models, firstly you need to specify models' parameters and train the model using the

training data. It is important to note the addition of the AR-Net will incorporate some new Neuronal Network related parameters that needs to be configured (see Table 13 for more detail).

Table 13 Neural Prophet Hyperparameters

NN Hyperparameter	Description
Number of hidden layers	Hidden layers are the layers between input layer and output layer. Number of them
Dimension of hidden layers	Number of Cells on each hidden layer
Epochs	Number of epochs is the number of times the whole training data is shown to the network while training.
Loss Function	The loss function captures the difference between the actual and predicted values. During NN training the idea is to minimize this loss function.
Learning Rate	When training a NN, the learning rate defines how quickly a network updates its parameters.
Changepoints Range	This is the proportion of the history in which the trend is allowed to change. This defaults to 0.8, 80% of the history, meaning the model will not fit any trend changes in the last 20% of the time series.
Number of Changepoints	This is the number of automatically placed changepoints

As we can observe NeuronalProphet models present even a higher combination of parameters than Prophet model. We have therefore created again a manual 'grid search' to iterate through different combinations of these parameters as showed on Script 13 in Appendix.

The script for training the model and create the forecast it is identical to the one previously detailed for simple Prophet model.

The top performing model is the Neural Prophet Table 14 shows WAPE results for this model. Again, we can observe really solid performance for P1 (6%) and P2 (6.72%) and tends to under-predict on P3 (18.88 %). We can observe how the Auto-regressive component of Neuronal Prophet slightly helps to improve the accuracy of the model when compared with the simple prophet on P2 and P3

Table 14 WAPE (%) results for Top performing NeuralProphet model

Seasonal Trend	Top performing Model
P1	6.00
P2	6.72
P3	18.88

Features

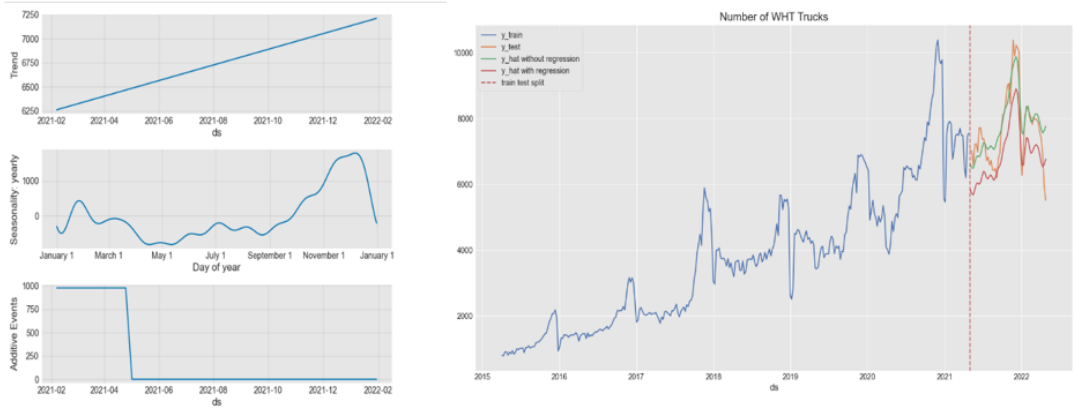
NeuronalProphet library offer the possibility of incorporate external regressors to our model using the *event_df* parameter. Table 15 shows the accuracy of the model when incorporating Amazon Events and COVID Regressors.

Table 15 Comparison of WAPE (%) results with and without features for NeuralProphet

Seasonal Trend	Without features	With Amazon Events	With COVID
P1	6.00	5.98	10.92
P2	6.72	6.70	9.86
P3	18.88	18.88	22.03

Neuronal Prophet model without regressors was already able to detect seasonality and peak event impacts and therefore the inclusion of these External Regressors is not improving a lot the accuracy of the predictions.

The additive way Neural Prophet model estimates the COVID Event is too simplistic, penalizing the accuracy. As we can observe on Figure 43 (a), Neuronal prophet will simply add a constant additive event component for the COVID flagged months. The model is overestimating the effect of COVID and therefor as we can observe on Figure 43 (b), for months in which the COVID component value is null (out of COVID Period) our model tends to under predict the number of trucks when compared with the simple model without external regressor.



(a) (b

Figure 43 Neural Prophet forecast with COVID regressor

Country Analysis

We have again found the best model parameters (based on WAPE) for each of the countries. Again, as we can observe on Table 16 the performance is solid across all the countries with the usual exception of CZ (24.71% WAPE). Despite this, more than on the global data the main benefit of Neural Prophet model when compared with Simple Prophet is the accuracy improvement for CZ (8,18 WAPE (%) Improvement). Effectively Autoregressive NN helps the model to better adjust to that descend trend suffered by CZ on 2021-2022

Table 16 WAPE (%) for Top performing Neural Prophet model for P1 by country

Country	WAPE (%)
UK	8.22
ES	10.13
FR	11.34
IT	9.16
DE	9.33
PL	8.00
CZ	24. 53

5.1.6 DeepAR

In order to implement Neuronal Network time series models we have used *GLuonTS library*. Gluon Time Series (*GluonTS*) is a toolkit developed by Amazon Scientist for probabilistic time series modelling, focusing on deep learning-based models. *GluonTS* is built around *MXNet*, an opensource deep learning software framework, used to train and deploy deep neural networks. Scrip 14 in Appendix shows how to import such framework

This tool simplifies the development and experimentation of TS data, providing component to quickly build new models, run experiments and evaluate model accuracy. *GluonTS* comes pre-equiped with a series of models that use already well-known time series related Neuronal Network architectures (Simple Feedforward, DeepAR, WaveNet..). For the purpose of this Master Thesis, we will just experiment with Deep-AR models to predict the number of Amazon trucks.

One of the main challenges with *GluonTS* is data manipulation. Custom time series data should satisfy some minimum format requirements to be compatible with *GluonTS*. The main requisites are 1) the dataset should be an iterable collection of data entries (dictionaries) and 2) each entry should have at least a target field, which contains the actual values of the time series and a start field, which denotes the starting date of the time series. This is implemented on Script 15 in Appendix.

As we already commented, *GluonTS* comes with a number of pre-built models. All the user needs to do is to select the desired model and hyperparameters values and train the selected model using our train data. DeepAR will also incorporate some Neuronal Network related parameters that needs to be configured: number of hidden layers, number of cells, epochs, batch size, number batches per epoch and learning rate. We have therefore created again a manual 'grid search' to iterate through different combinations of these parameters and select the top performing one using again WAPE as criteria.

We can now create our estimator, training it on our data (predictor) and use it to predict on our forecast horizon by using the make_evaluation_predictions function that automates the process of prediction and model evaluation. Roughly, this function performs the following steps: 1) Removes the final window of length prediction_length of the dataset that we want to predict, 2) The estimator uses the remaining data to predict (in the form of sample paths) the "future" window that was just removed and 3) The module outputs the prediction. This is done as showed on script 16 in Appendix.

The top performing model is the DeepAR (*learning_rate=1e-4, epochs=100, num_batches_per_epoch=50, num_cells=40, num_layers = 3, batch_sixe =32*). Table 17 depicts WAPE results for P1, P2 and P3.

Table 17 WAPE (%) results for top performing DeepAR model

Seasonal Trend	WAPE (%)
P1	5.64
P2	5.92
P3	13.65

Figure 44 depicts the out forecast of this model for P1, P2 and P3. We can graphically confirm how DeepAR model is able to improve the prediction for P3 when compared with previous models.

Probabilistic forecast it is not in the scope of this work. Despite this we have calculated and display in our graphs the confidence interval (95 and 80). The confident interval is a range of values that's likely to include a population value with a certain degree of confidence. As an example, the 95% confidence interval is a range of values that you can be 95% confident contains the true mean of the population. One of the main advantages of DeepAR models when compared with previous ones is that, as you can observe on Figure 44, it provides narrower and therefore more precise confident intervals.

Features

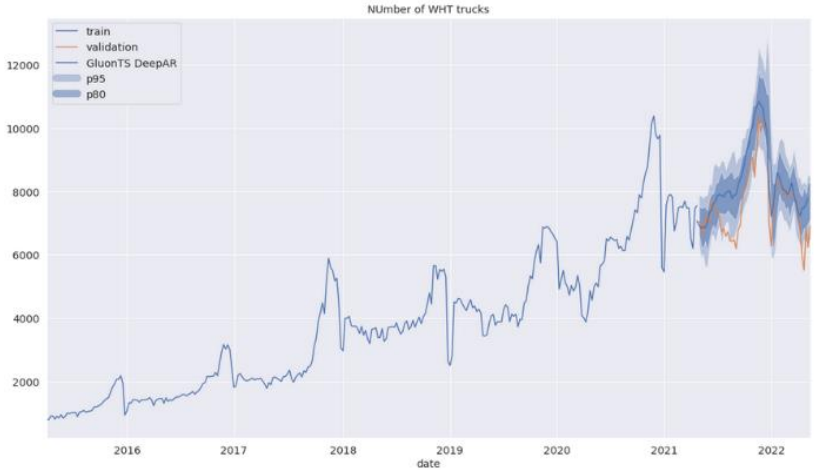
One of the main disadvantages of DeepAR Models is that is very difficult from the technical point of view to incorporate extra features/external regressors to the model. We have therefore excluded this analysis from the scope of this work.

Country Analysis

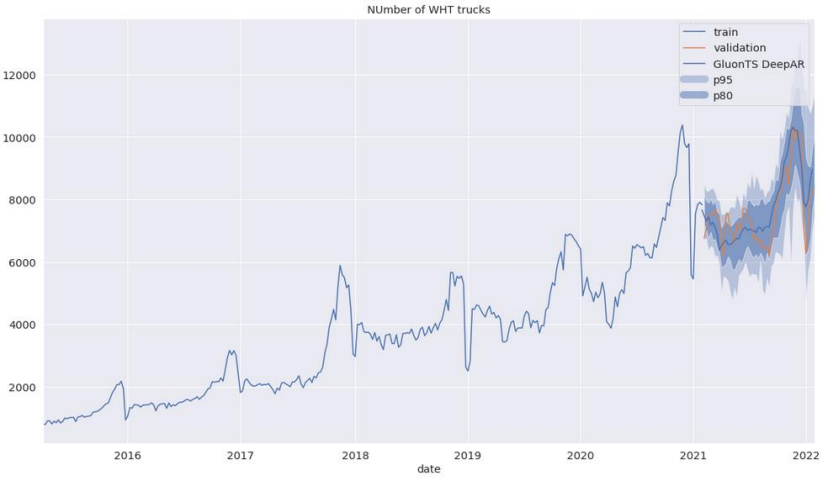
We have again found the best model parameters (based on WAPE) for each of the countries on Table 18. The performance of DeepAR model is solid across all the countries with exception of CZ (22.67% WAPE). We can observe the model accuracy is especially high for UK (8.05% WAPE) and PL (7.89% WAPE).

Table 18 WAPE (%) for country top performing DeepAR model for P1

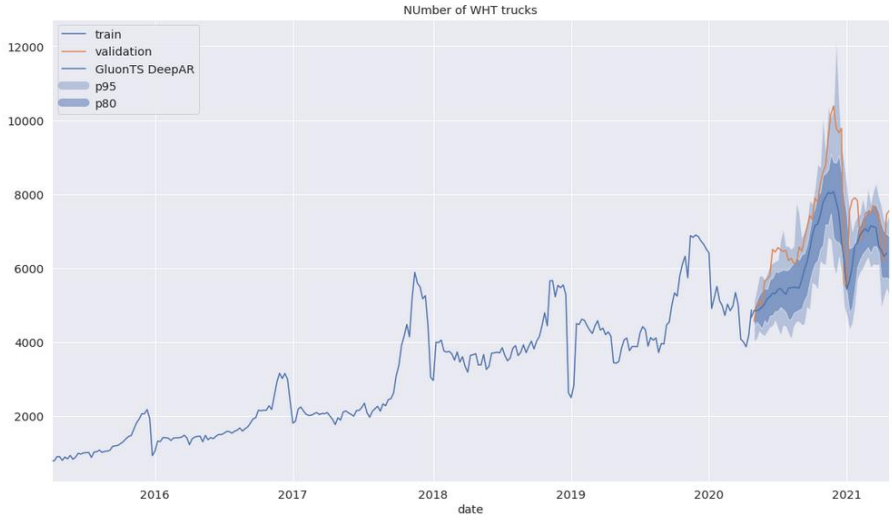
Country	WAPE (%)
UK	8.05
ES	9.58
FR	10.86
IT	9.16
DE	9.07
PL	7.89
CZ	22.67



(a)



(b)



(c)

Figure 44 DeepAR top performing model forecast for (a) P1, (b) P2 and (c) P3

Global Model

One of the main advantages of deep Learning based models is that it allows to pool and predict the data of all time series together allowing the model to apply the learnings from the training data of one specific country to other countries (global model). This is specifically useful when we want to forecast the number of trucks for new countries or countries with low amount of data. On this regard, we perform an experiment to use DeepAR model to try to find a global model able to predict the number of trucks for all the country instead of trying to find separately the best model for each of the countries. From a coding perspective, the implementation of all the countries under the same global model in *GluonTS* required some interesting data manipulation pre-works that can be summarized basically in two steps: 1) create a dataframe containing each country time series in a different row and 2) specify the Country as the variable category when creating the train and test by using the FEAT_STATIC_CAT parameter.

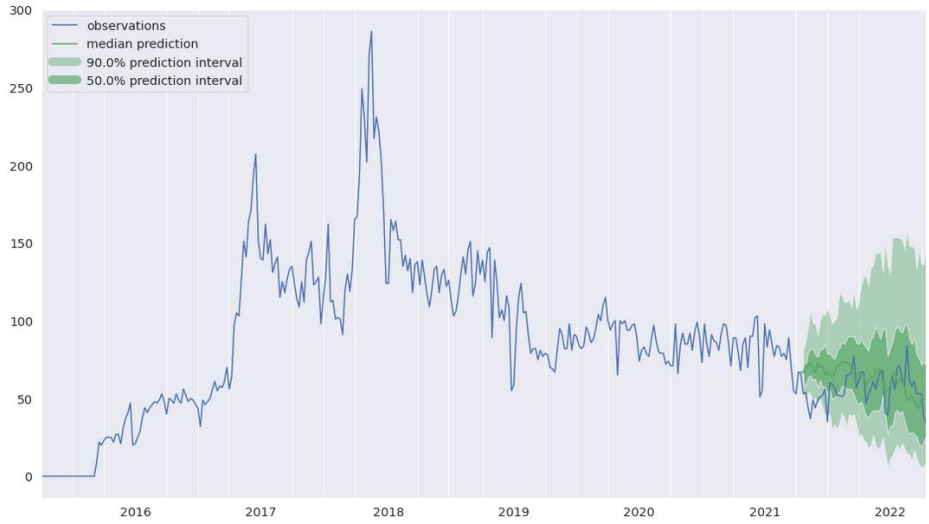
We have calculated the WAPE performance of the model on each of the countries. It is important to understand that it is normal that this country-level performance is lower than previous models as previously we were implementing a specific model for each of the countries, whereas on this case we are using a global model for all the countries. Results are depicted in Table 19.

Table 19 WAPE (%) for Global DeepAR model for P1

Country	WAPE (%)
UK	22.42
ES	10.25
FR	12.85
IT	10.06
DE	12.46
PL	8.75
CZ	23.45

The performance is solid across all the countries with exception of UK (22.42 %) and CZ (23.45%). Effectively, as you can observe on Figure 45, UK and CZ are the only 2 markets with a descending trend for 2022 and therefore the global model tends to over-estimated the number of forecasted trucks.

(a)



(b)

Figure 45 DeepAR global model forecast for (a) UK, (b) CZ

5.2 Model Results Comparison

After the detailed explanation an analysis of the different models by separate, this section showcases a comparative analysis on model's performance. Table 20 shows the accuracy (WAPE%) of each kind of model for the three already specified period. It is important to note that for each model type we have selected the top performing one.

Table 20 WAPE (%) for all models and periods

Model/Period	P1	P2	P3
Simple Methods	9.94	14.95	26.48
ETS Models	9.04	7.44	27.68
ARIMA	6.30	7.73	24.18
Prophet	6.45	6.95	18.75
Neuronal Prophet	6.00	6.72	18.88
DeepAR	5.64	5.92	13.65

Firstly, we can confirm how both Classical and Deep Learning models outperform Simple Methods. On other words, we confirmed that ours models are able to actually learn from the data and work better than simple and really basic models used as benchmark (Simple Methods). We can also observe that both Neuronal Network model (DeepAR) and Prophet models (Prophet, Neural Prophet) present a higher accuracy than Classical Parametric methods (ARIMA, ETS). Finally, even if DeepAR showed a decent WAPE accuracy we can highlight that all the models encounter difficulties to predict the number of trucks for Period 3. Effectively in P3 where training set does not contain almost any COVID data, all the models fail to predict that spike on the number of trucks variable caused by the COVID Situation.

Table 21 shows the accuracy (WAPE%) of each kind of model for each specific country. Again, it is important to note that for each model type we have selected the top performing model.

Table 21 WAPE (%) for all models by country

Model/Country	UK	ES	FR	IT	DE	PL	CZ
ETS Models	7.85	15.77	16.31	20.06	9.04	11.41	20.52
ARIMA	8.25	14.77	14.25	13.75	9.36	11.23	22.35
Prophet	8.65	10.17	12.64	9.19	9.33	9.26	32.71
Neuronal Prophet	8.22	10.13	11.34	9.16	9.33	8.00	24. 53
DeepAR	8.05	9.58	10.86	9.16	9.07	7.89	22.67
DeepAR (Single model for all countries)	22.42	12.25	12.85	10.06	12.46	8.75	23.45

We can observe again how both Neuronal Network models and Prophet outperform Classical models also on country level forecasting. We can observe how all model present an accuracy drop for CZ country. Effectively, due to strategic decision Amazon business in CZ were reduced from 2021 and therefore all models tend to overpredict P1 forecasted values for CZ. This strategic decision is something that obviously our ML/Statistical techniques are not able to model.

6 CONCLUSIONS AND FUTURE WORK

1. Firstly, it is confirmed that this forecasting project was a success inside the company Applied Science department, since we were able to find and extract clean data and build different models able to forecast the number of required trucks for a one-year horizon with a good accuracy both for global network in the third studied period and for the specific country level. The output of these models will be a key data/insight that seniors transport managers will be able to use in the future for operational decision-making.

2. The top performing model both on global and country specific level is the DeepAR. This Deep Learning based model was able to forecast the number of trucks with an accuracy around 6% (WAPE) for global data and around 9% (average WAPE off all countries excluding CZ) for country-level data. On other words, our model is able to predict what the number of trucks required by Amazon will be for the whole year horizon with only an error of 6% for the global EU Network and 9% for each specific country.

3. Overall, we have confirmed that modern techniques outperform classical methods for our particular use case. Despite this, it is important to note that methods based on neuronal network have two main shortfalls: 1) they required greater computational capacity and running time and 2) they required more extensive data manipulation as the structure of the inputs for the model is, in general, more complex.

4. On other side, the results obtained confirmed that a single DeepAR global model is able to forecast the number of trucks for each specific country with robust accuracy. This global model is able to apply cross-learning between the behavior of each country. Such model will be really useful for the transportation department, since it will allow us to leverage of data from other countries to make forecast on new countries where Amazon is expanding and for which we have really limited data.

5. Finally, this analysis shows that even the most complex model does not allow the inclusion of the COVID effect, unless the model is trained with specific data from that period. This finding suggests that the accuracy of long-term forecast problems will always be limited by unexpected events (macroeconomic crisis, pandemics, wars, etc.).

Future Research

The work done for this thesis served to confirm that we are able to predict the number of required trucks for the long term with a good accuracy. Once this have been confirmed there are some follows up lines of study and work to continue this original thesis and that would be helpful for the Applied Science department.

First of all, it would make sense to explore other models that were not in the scope of that thesis but could potentially have a good performance/accuracy (Decision Trees based models or other RNN/LTSM Neuronal Networks architectures build from scratch and different from DeepAR).

Another path to explore would be the Hierarchical time series Forecasting. When you have hierarchical data, it is the process of generating coherent forecasts allowing individual time series to be forecast

individually, but preserving the relationships within the hierarchy. It could be a Bottom-up approach (Forecast firstly the most granular level of the hierarchy (each of the countries) and then aggregating up to create estimates for the higher level (Global EU Network) or Top-down approach (you first forecast the highest level of the hierarchy (Global EU Network) then split up the forecasts to get estimates for each specific country (typically using historical proportions).

One important area that could be also worthy to explore would be correlations. On more detail, try to correlate the number of trucks variable with other external quantitative economic variables, such as the Gross Domestic Product (GDP) of each country or the average stock price of Logistic competitors

On other side, another area that could be interesting to explore is trying to go on a deeper granularity on the forecasting. Instead of only predict until country level, being able to predict on Cluster level. As an example, create model that not only predict the number of future trucks in Spain but also the specific number of trucks required for example for the South Cluster (Andalucia and Extremadura). On this regard, once we have the cluster level information it would make sense to include the number of active cluster and Site opening as a Feature.

All the accuracy metrics used during this work were based on a point forecasting approach. It would be valuable to extend our analysis on the probabilistic field. At the end, we won't be able to predict the exact number of trucks for one specific week one year ahead. However, for Amazon leaders it would be a really powerful decision-making supporting tool understand what it is the probability of having more or less than a certain number of trucks in one specific week.

Finally, there will be some good future work invested on productionizing the model. At the moment all the experiments were run using a Simple Notebook. Once we confirmed the model is working it will be required to 1) Automate the pipelines sourcing the model with data from Amazon Databases, 2) implement the model in Sagemaker and schedule it to run on a certain cadence and 3) store the output of the model on a Redshift Database table and use it to source a Dashboard that Amazon Senior Leaders are able to leverage.

REFERENCES

[1] "*Amazon's* Last Mile" based on paper by Jörn Boewe, Tina Morgenroth and Johanne Schultenhttps (in german). Rosa-Luxemburg-Stiftung. Online Available: http://www.rosalux.de/fileadmin/images/EnglishWS/AmazonLastMile/RLS_Amazons_last_mile_DIN-A-4_20211007_Print.pdf.

[2] Amazon Machine Learning University - Time Series Forecasting course.

[3] Taylor, S.J, Letham, B.: "Forecasting at scale", *The American Statistician*, *72*(1), 37–45. 2018. https://doi.org/10.1080/00031305.2017.1380080.

[4] Rosenblatt, F., "The perceptron: A probabilistic model for information storage and organization in the brain", *Psychological Review,* *65*(6), 386-408. 1958. Online Available: https://doi.org/10.1037/h0042519

[5] Rumelhart, D. E., Hinton, G. E. *and* Williams, R. J.. "Learning Internal Representations by Error Propagation". *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations. 1986.* ISBN 0-262-18120-7.

[6] Triebe, O., Hewamalage, H., Pilyugina, P., Laptev, P., Bergmeir, C. and Rajagopal, R., "NeuralProphet: Explainable Forecasting at Scale", 2021. Online Available: https://arxiv.org/abs/2111.15397

[7] Vandeput, N.: "Probabilistic Forecasts: Pinball Loss Function", *Towards Data Science Inc.* 2021. Online Available: https://towardsdatascience.com/probabilistic-forecasts-pinball-loss-function-baf86a5a14d0

[8] Nduati, E.: "A Data Cleaning Journey", *Analytics Vidhya*, Online Available: https://medium.com/analytics-vidhya/a-data-cleaning-journey-2b0146407e44

[9] Catherine, V., "In-Depth Understanding of NeuralProphet through a Complete Example", *Towards Data Science Inc.* 2021. Online Available: https://towardsdatascience.com/in-depth-understanding-of-neuralprophet-through-a-complete-example-2474f675bc96

[10] Radecic, D.: "Time Series From Scratch — Autocorrelation and Partial Autocorrelation Explained", *Towards Data Science Inc.* 2021. Online Available: https://towardsdatascience.com/time-series-from-scratch-autocorrelation-and-partial-autocorrelation-explained-1dd641e3076f

[11] Hyndman, R.J., and Athanasopoulos, G: Forecasting: principles and practice, 2nd edition, OTexts. 2018. Online Available: Forecasting: Principles and Practice (2nd ed) (otexts.com)

[12] Masum, M. Time Series Analysis: "Identifying AR and MA using ACF and PACF Plots", *Towards Data Science Inc.* 2021. Online Available: https://towardsdatascience.com/identifying-ar-and-ma-terms-using-acf-and-pacf-plots-in-time-series-forecasting-ccb9fd073db8

[13] Singh, S.: "Understanding the Bias-Variance Tradeoff, *Towards Data Science Inc.* 2019. Online Available: https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229

[14] Alizadeh, E. 2020, "NeuralProphet: A Time-Series Modeling Library based on Neural-Networks", *Towards Data Science Inc.* 2019. Online Available: https://towardsdatascience.com/neural-prophet-a-time-series-modeling-library-based-on-neural-networks-dd02dc8d868d

[15] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-libr

Appendix

Script 1:

```
etsaaa = ETSModel(train, error="add", trend="add", seasonal="add", seasonal_periods=52)
etsaaa_fit = etsaaa.fit()
```

Script 2:

```
etsaaa_get_pred = etsaaa_fit.get_prediction(
    start=test.index.min().date(), end=test.index.max().date()
)
etsaaa_pred = etsaaa_get_pred.predicted_mean
```

Script 3:

```
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

Script 4:

```
sarima = sm.tsa.statespace.SARIMAX(
    train, order=(3, 1, 3), seasonal_order=(1, 1, 1, 52)).fit()
```

Script 5:

```
sarima_get_pred = sarima.get_prediction(
    start=test.index.min().date(), end=test.index.max().date() )
```

Script 6:

```
param_grid = {
                'p': [1,2,3,4,5],
                'd': [1],
                'q':[1,2,3,4,5],
                'P':[1,2,3],
                'D':[1],
                'Q':[1,2,3],
                'm': [52]

              }

# Generate all combinations of parameters
all_params = [dict(zip(param_grid.keys(), v)) for v in itertools.product(*param_grid.values())]
wapes = []  # Store the MAPEs for each params here


# Evaluate all parameters in
for params in all_params:
    print(params)
    print(wapes)
    sarima = sm.tsa.statespace.SARIMAX(train, order=(params['p'], params['d'], params['q']),
                                       seasonal_order=(params['P'], params['D'], params['Q'], 52)).fit()
    sarima_get_pred = sarima.get_prediction(
    start=test.index.min().date(), end=test.index.max().date())
    sarima_pred = sarima_get_pred.predicted_mean
    wapes.append((test - sarima_pred).abs().sum() / test.sum()*100)
```

Script 7:

```
# Let's add External regressor to the SARIMAX Model
sarima = sm.tsa.statespace.SARIMAX(
    train, order=(1, 1, 3), seasonal_order=(1, 1, 2, 52), exog = train_holiday ).fit()
```

```
# Add external regressors values for test data
sarima_get_pred = sarima.get_prediction(
    start=test.index.min().date(), end=test.index.max().date(),exog = test_holiday )
```

Script 8:

```
from fbprophet import Prophet
from fbprophet.plot import plot_plotly
```

Script 9:

```
my_model = Prophet(interval_width=0.95, yearly_seasonality=True,
                   changepoint_range=0.75, changepoint_prior_scale=0.045, holidays = holiday)

my_model.fit(df_train_pf)
```

Script 10:

```
future_prophet=my_model.make_future_dataframe(periods=52, freq='W-SUN')
```

```
forecast=my_model.predict(future_prophet)
```

Script 11:

```
param_grid = {
                'changepoint_prior_scale': [0.001,0.002,0.003,0.004,0.005, 0.05, 0.5, 5],
                'changepoint_range': [0.6,0.7,0.8, 0.9],
                'seasonality_prior_scale':[0.1, 1, 10.0],
                'holidays_prior_scale':[0.1, 1, 10.0],
                'seasonality_mode': ['additive']
              }

# Generate all combinations of parameters
all_params = [dict(zip(param_grid.keys(), v)) for v in itertools.product(*param_grid.values())]
wapes = []  # Store the WAPEs for each params here
y = df_test_pf.y.reset_index(drop=True)

# Evaluate all parameters in
for params in all_params:
    print(params)

    mod = Prophet(**params).fit(df_train_pf)
    future_prophet=mod.make_future_dataframe(periods=60, freq='W-SUN')
    forecast=mod.predict(future_prophet)
    ypred = forecast[(forecast['ds'] < '2022-05-01') & (forecast['ds'] > '2021-04-30')]
    ypred = ypred.reset_index(drop=True)
    ypred = ypred.yhat
    wape = (y - ypred).abs().sum() / y.sum()*100
    wapes.append((y - ypred).abs().sum() / y.sum()*100)
    print(wapes)
```

Script 12:

```
from neuralprophet import NeuralProphet
```

Script 13:

```
param_grid = {
                'num_hidden_layers': [4,5,6,7],
                'changepoints_range': [0.5,0.6,0.7,0.9],
                'n_changepoints': [2,5,6,7,10],
                'loss_func':["huber", "smoothl1", "smoothl1loss","mse", "mseloss", "l2", "l2loss"],
                'num_hidden_layers': [4,7,8,10],
                'd_hidden': [4,5,7],
                'learning_rate': [0.01,0.1,0.5],
                'epochs': [100]
              }

# Generate all combinations of parameters
all_params = [dict(zip(param_grid.keys(), v)) for v in itertools.product(*param_grid.values())]
wapes = []  # Store the WAPEs for each params here
y = df_test_pf.y.reset_index(drop=True)

# Evaluate all parameters in
for params in all_params:
    print(params)
    my_model = NeuralProphet(**params)
    my_model.fit(df_train_pf, freq="W")
    future_prophet=my_model.make_future_dataframe(df_train_pf, periods=60, n_historic_predictions=len(df))
    forecast=my_model.predict(future_prophet)
    ypred = forecast[(forecast['ds'] < '2022-05-01') & (forecast['ds'] > '2021-04-30')]
    ypred = ypred.reset_index(drop=True)
    ypred = ypred.yhat1
    wape = (y - ypred).abs().sum() / y.sum()*100
    wapes.append((y - ypred).abs().sum() / y.sum()*100)
    print(wapes)
```

Script 14:

```
import mxnet as mx
from mxnet import gluon

from gluonts.evaluation.backtest import make_evaluation_predictions
from gluonts.evaluation import Evaluator
```

Script 15:

```
from gluonts.dataset.common import ListDataset

# train dataset: cut the last window of length "prediction_length", add "target" and "start" fields
train_ds = ListDataset(
    data_iter=[{"target": x, "start": start} for x in target[:, :-prediction_length]],
    freq=freq,
)

# validation dataset: use the whole dataset, add "target" and "start" fields
val_ds = ListDataset(
    data_iter=[{"target": x, "start": start} for x in target], freq=freq
)
```

Script 16:

```
estimator = DeepAREstimator(
    prediction_length=prediction_length,
    freq=freq,
    trainer=Trainer(
        ctx="cpu",
        learning_rate=1e-4,
        epochs=100,
        num_batches_per_epoch=50,
        num_cells=40,
        num_layers = 3,
        batch_sixe =32
    ),
)

predictor = estimator.train(train_ds)

forecast_iter, ts_iter = make_evaluation_predictions(
    dataset=val_ds,  # val dataset
    predictor=predictor,  # predictor
    num_samples=100,  # number of sample paths we want for evaluation
)
```