Expertise
and insight
for the future

Aleksandr Tereshchenko

# Demand forecasting in the context of production planning system

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

27 October 2018

| | |
|---|---|
| Author | Aleksandr Tereshchenko |
| Title | Demand forecasting in the context of production planning system |
| Number of Pages | 50 pages + 4 appendices |
| Date | 27 October 2018 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information Technology |
| Professional Major | Software Engineering |
| Instructors | Peter Hjort, Senior Lecturer<br>Matti Rita-Kasari, CEO, Jubic Oy |

The thesis presents the research on the demand forecasting solution within the context of the production planning system. Since any production planning depends primarily on successful demand predictions, developing an accurate forecasting model is crucial for the success of the final solution. In turn, the production planning system will resolve multiple problems of the client company including the unjustified overproduction, the inefficiency of the production process and enable the personnel currently performing the planning manually to concentrate on other activities.

The theoretical background related to the forecasting solutions studied over the course of the research is described. Besides, the theory related to model ensembling is presented. Furthermore, six forecasting models were developed and described in the thesis. Among the models, the one-dimensional convolutional neural network is presented as an alternative to more traditional forecasting solutions. Finally, the benchmark, which uses the actual historical data obtained from the client, is described, and the techniques for model evaluation are discussed.

The benchmark findings demonstrate promising results for the majority of the products examined. Besides, it is pointed out that averaging the forecasts of multiple models resulted in more accurate predictions in the 4 cases out of the 9 analyzed. At the end of the thesis, the possible improvements are discussed, which will further increase the models' accuracies, thus enabling even more efficient operation of the production planning system.

| | |
|---|---|
| Keywords | time series forecasting, production planning, Prophet, ARIMA, dilated convolution |

## Contents

## List of Abbreviations

| | |
|---|---|
| AIC | Akaike information criterion |
| ANN | Artificial neural network |
| API | Application programming interface |
| AR | Autoregression |
| ARIMA | Autoregressive integrated moving average |
| BATS | Box-Cox transformation, ARMA errors, trend and seasonality |
| BIC | Bayesian information criterion |
| CNN | Convolutional neural network |
| CNTK | Cognitive Toolkit |
| CPU | Central processing unit |
| CSV | Comma-separated values |
| DB | Database |
| DNN | Deep neural network |
| ERP | Enterprise resource planning |
| GA | Genetic algorithm |
| GPU | Graphics processing unit |

| | |
|---|---|
| GRU | Gated recurrent unit |
| JDBC | Java Database Connectivity |
| JSON | JavaScript Object Notation |
| LSTM | Long short-term memory |
| MA | Moving average |
| MAE | Mean absolute error |
| MAPE | Mean absolute percentage error |
| MASE | Mean absolute scaled error |
| MILP | Mixed integer linear programming |
| MPS | Master production scheduling |
| NP | Non-deterministic polynomial time |
| R&D | Research and development |
| ReLU | Rectified linear unit |
| REST | Representational state transfer |
| RMSE | Root mean squared error |
| RNN | Recurrent neural network |
| SES | Simple exponential smoothing |
| SGD | Stochastic gradient descent |

| | |
|---|---|
| sMAPE | Symmetric mean absolute percentage error |
| SQL | Structured Query Language |
| STL | Seasonal and Trend decomposition using Loess |
| STLF | Forecasting using STL decomposition |
| TBATS | Trigonometric seasonal component, Box-Cox transformation, ARMA errors, trend and seasonality |
| TS | Time series |

# 1 Introduction

Production planning has long been one of the key aspects of any manufacturing industry. This is especially true nowadays since widespread digitalization and macroeconomic conditions cause businesses to rethink their processes reaching new heights in efficiency. However, any production planning technique - be it manual or automated - heavily depends on successful demand forecasting. Indeed, in order to plan production, it is required that future demand is predicted with high enough accuracy.

Due to the recent popularity of machine learning methods, an increasing number of businesses is searching for possibilities to incorporate it, thus improving their internal practices, replacing human work, where possible, and eventually, increasing profits. Demand forecasting is a long-studied field with significant research history in academia and numerous successful application cases. Still, as machine learning techniques progress rapidly, there is room for improvement over existing solutions.

The project has been carried out for Jubic Oy, a relatively small company located in Vaasa, Finland, which provides system and software development, consulting and cloud hosting services. In the case of the project, a client company, which operates in the manufacturing industry, ordered end-to-end development of the production planning system that would suit their custom needs.

The objective of the thesis is to research what solution or algorithm should be used for demand forecasting required by the mentioned system. More specifically, it will be researched whether a combination of forecasting solutions can produce similar or better results compared to any single model.

Finally, while the main focus of the thesis is on demand forecasting methodology, it is important that the context, namely the production planning software, is considered since it might influence certain decisions. Thus, the high-level architecture of the system and the interactions between its components are described in Section 5.3

# 2 Theoretical background

In this section, some key concepts of production planning and time series forecasting that are particularly relevant to the case of the client company are outlined. First, a typical production planning process is outlined, and the concept of a genetic algorithm is explained. Then, some of the most relevant forecasting models are introduced and shortly explained. Besides, time series decomposition is reviewed before the model descriptions to provide further theoretical background.

## 2.1 Production planning

Production planning can be defined as a procedure of determining what manufacturing units should perform what tasks at what times in the future with the purpose of maximizing or minimizing an objective, such as the number of production batches, total expenditure or amount of produced goods.[1] A production planning process typically consists of multiple stages that form a hierarchy, with higher-level ones acting as guidelines for more detailed lower-level stages. The stages include a business plan, an aggregate production plan, master production scheduling (MPS) and sequencing among others. [2 pp. 114 – 117] The previously mentioned stages were listed starting from the more general ones (see Figure 1).

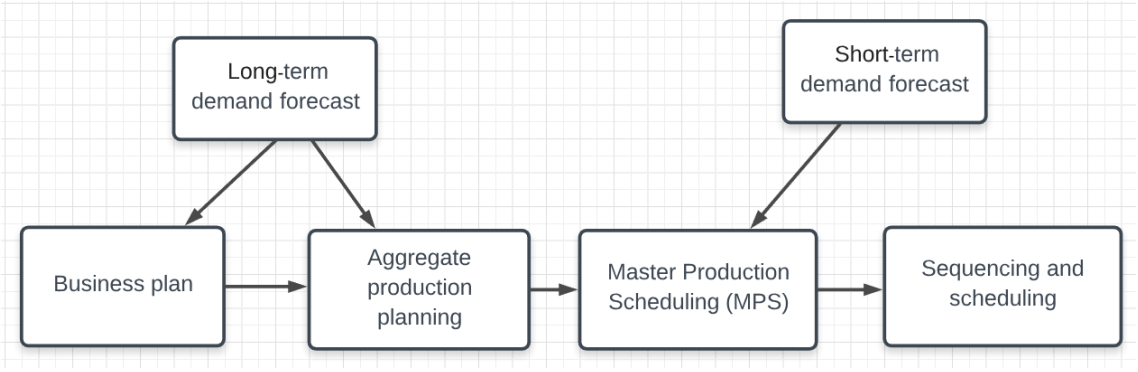



Figure 1. Production planning stages. Based on the information from Starr (2008) [2 pp. 114 – 117]

The listed stages will be briefly described in the following paragraphs. The information in the paragraphs is primarily based on Starr [2 pp. 114 – 117]. A business plan is the first

stage of a production planning process. It outlines general activity of a company for the following half to one and a half years. A business plan typically takes into consideration financial, marketing, production and R&D sections of a company, thus providing an agreement between all the parties as to what the company's focus is going to be for the next long-term period.

Aggregate production planning is concerned with how much of the goods specified in the business plan can and should be supplied over the next long-term period. An aggregate production plan usually specifies output levels on a weekly or monthly basis making sure the planned amounts are feasible given facilities of the organization. However, to provide further detail to the production plan, master production scheduling is used receiving medium-term and/or short-term demand forecasts and the aggregate production plan as its inputs. MPS is usually based on such factors as demand, production capacity, storage levels and storage capacities, costs of production and materials, etc. Results of an MPS stage are often presented as a set of tables each corresponding to a separate product the organization in question produces, where each column corresponds to a time unit in the future, e.g. weeks or days, while rows indicate required production amounts, demand, stock balances and other important factors. Table 1 presents a simple MPS result table for a single product.

Finally, the sequencing and scheduling stage is mainly concerned with what machinery should be run at what times to meet the plan provided by MPS. This stage specifies start and end running times for each machinery unit as well as what products are manufactured in these time intervals and, in some cases, what materials are to be used for it. Besides, this stage often manages workforce allocation as well.

Table 1. A sample MPS table. Based on Beasley [3].

| **Item N** | **17.04** | **18.04** | **19.04** | **20.04** |
|---|---|---|---|---|
| Demand (forecast) | 2098 | 2245 | 3056 | 1129 |
| Number of employees | 15 | 15 | 15 | 10 |
| Storage cost (per unit) | 0.5 | 0.5 | 0.5 | 1 |
| Production capacity (per hour) | 8900 | 8900 | 8900 | 8900 |
| Initial inventory | 11345 | | | |
| Salary per employee | 15 | 15 | 15 | 25 |
| Firm planned order | 2000 | | | |
| Planned order | ? | ? | ? | ? |
| Projected available balance | ? | ? | ? | ? |
| Available to promise | ? | ? | ? | ? |

Different techniques could be used to solve each of the beforementioned planning stages. For instance, MPS could be solved using a general linear programming model or a so-called transportation model [3]. However, the sequencing stage is often expressed as a mixed integer linear programming (MILP) problem [4-6]. And while explaining MILP is beyond the scope of this thesis, it should be noted that it is NP-hard, which means it often cannot be solved using “exact” optimization methods [4]. Due to this limitation, one of metaheuristic methods should be used. A commonly used option is an approach using a genetic algorithm (GA) [7], which is described in detail in the remaining part of the subsection.

**Genetic algorithms**

A genetic algorithm is a metaheuristic and a branch of a larger field of study called evolutionary algorithms. Genetic algorithms are inspired by natural evolution process, where a chance of survival of a given individual is determined by its fitness. Since GA can be useful for solving very complex otherwise non-feasible optimization problems, it has been successfully applied across various domains and industries. The information provided in this subsection is based on Mitchell [8 pp. 2 - 10].

Before proceeding further with the explanation, it is necessary to define several terms commonly used when dealing with GA. Firstly, an individual is a single proposed solution

which might or might not be sufficiently suitable. Secondly, a fitness function or an objective function is a function used to evaluate “fitness” of an individual. Very often, the purpose of a GA is to minimize or maximize an objective function. Furthermore, each individual contains one or multiple chromosomes, each of which, in turn, contains a number of genes. Each gene can be thought about as a particular trait of an individual. It has to be noted that sometimes an individual can only have a single chromosome by definition, whereas in other cases it is defined with multiple chromosomes. For simplicity, the case with a single chromosome is considered in this section. The genes generally represent parts of a particular solution to a problem. For instance, taking an example from production planning context, each gene may represent a time unit, during which a given machinery should or should not operate depending on the value of the gene. The previous sentence demonstrates how a real-life problem can be translated to suit GA context. Such a translation from an actual problem to its gene representation is defined by an algorithm’s author and in many cases can be challenging.

Besides the previously explained concepts, selection, crossover and mutation should be introduced as well. Firstly, selection is a process of selecting a subset of individuals that would be able to produce offspring i.e. the next generation. Generally, a certain number of the best individuals are selected. Secondly, crossover is a process of mixing two chromosomes of two individuals with the purpose of producing a new chromosome, which is used to form a new individual. There are multiple types of crossover, among which some of the most popular ones are single-point, multi-point and uniform. Finally, mutation is a process of randomly changing certain genes in a chromosome of a newly created individual. Mutation is important and allows next-generation individuals to have qualities that are not present in their parents’ generation.

A simple genetic algorithm is usually executed according to the following steps. First, a population of $n$ individuals is initialized. The initialization is generally performed using randomly generated individuals, but it can also be specified manually if needed. Then, each individual is evaluated by applying a fitness function to it. After this, a so-called survivor selection is performed selecting a subset of individuals based on their fitness values. This is followed by a crossover process producing offspring which is going to become the next generation. The described selection and crossover processes are repeated multiple times until a total number of offspring individuals reaches $n$. The new individuals then become a new generation, and the algorithm described in this paragraph

is repeated. The execution is terminated after several generations based on one or multiple conditions. The conditions commonly used for genetic algorithm termination are reaching a certain specified number of generations, having a certain number of generations in a row during which a fitness value of the best individual does not change, achieving a specified fitness value by at least one individual, etc.

## 2.1. Time series forecasting

The rest of the section will describe the most important aspects of time series forecasting, its decomposition and list some of the most popular forecasting models that are used nowadays. If not stated otherwise, the information presented in this section is based on chapters 6-8, 11.1, 11.3 and 12.4 of Hyndman and Athanasopoulos [9].

### 2.1.1. Overview

According to Mahalakshmi et al., time series forecasting is a type of temporal data mining [10]. Time series data is defined as a number of certain values each associated with a timestamp distributed over a time interval [10]. Frequency of the timestamps can range from milliseconds to months to decades depending on a specific situation and needs.

Time series analysis in general and forecasting specifically are used across a wide range of industries and fields. Some of the areas of application include medical domains, stock markets, astronomy, demand prediction, energy demand forecasting and management, predictive maintenance in multiple industries, etc. [11] Moreover, certain fields, such as already mentioned meteorology, almost exclusively rely on time series forecasting in their operations. Finally, an increasing number of businesses that are new to the technique are readily adopting it nowadays as a way to stay competitive in today's world where digitalization and process automation makes a significant impact on the market.

The length of the period for which a forecast is executed is often called forecasting horizon. Depending on the length of a horizon, time series forecasting can be classified into three categories, namely short-term forecasting, medium-term forecasting and long-term forecasting. Although the boundaries for the terms are not strictly defined, it is generally considered that forecasts with a horizon of up to two to three months are short-term,

while those with a horizon longer than a year are long-term. [10] This thesis focuses primarily on short-term forecasting, which is justified by needs to provide predictions with daily resolution.

In addition to this, the notions of a prediction interval and a point forecast should be explained briefly. Since any forecast is never a guarantee of the way some future events will unfold, it is important to understand how much uncertainty exists when a variable is forecasted. For example, when forecasting a daily demand that for the previous five years has never exceeded the value of 80 units, it is extremely improbable that in the following seven days the daily demand happens to be equal to 9000 units. On the other hand, values between 30 and 60 are considerably more feasible in this example. This loosely represents the notion of probability interval. Specifically, when it is said that a 95% probability interval at time ***T*** for a variable ***y*** is between values ***a*** and ***b***, it is meant that the variable ***y*** will occupy a value within this range with the probability of 95%. Furthermore, a tighter interval can be specified, for which the percentage probability value would typically be smaller e.g. 80%. Thus, a forecasting probability interval represents a range of values that a variable of interest is expected to occupy with relatively high probability.

In contrast to this, a point forecast is a single value, which can be thought about as a middle point of corresponding prediction intervals. To clarify it further, one can think about a prediction interval, which, by becoming increasingly tighter, eventually converges to a single point. It can be argued that a single point forecast does not carry significant meaning without uncertainty intervals associated with it since it is impossible to understand how accurate the forecasting model of interest is. Nevertheless, when evaluating a model, it is common to execute a benchmark to produce multiple point forecasts and calculate related errors based solely on them. Thus, while prediction intervals will be featured in the final implementation of the system, the scope of the thesis is limited to point forecasts only.

### 2.1.2. Decomposition of time series

Time series data can be represented as three separate components, namely a trend, a seasonality and a so-called remainder component. This is usually helpful for analyzing

data. For example, having a trend extracted might provide a better insight into how successful the last year of sales was. Or, on the other hand, having the data stripped of its seasonal and trend components helps to understand whether the time series is volatile. While there are multiple methods available for performing time series decomposition, Seasonal and Trend decomposition using Loess (STL) [12] is among the most popular ones and has a number of advantages compared to other methods.

Figure 2 demonstrates an example of such decomposition. The first frame depicts the partial demand data for one of the products of the client company. Natural to sales data, the demand contains both yearly and weekly seasonal components, which can also be seen in Figure 2. Finally, although not obvious form the original data, the demand time series contains a positive trend. The values of the y axes were removed from the figure due to the confidentiality policy of the client company.

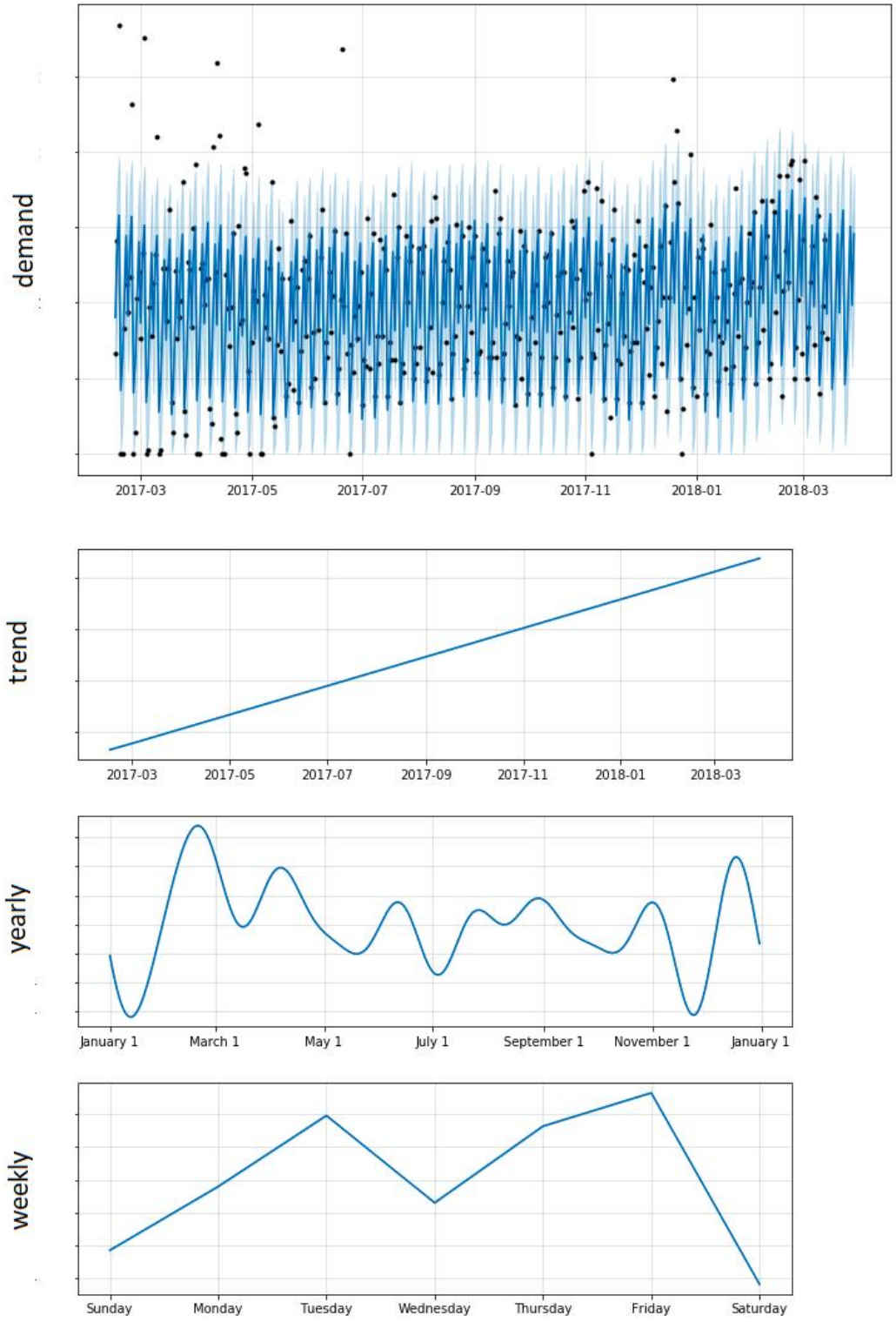



Figure 2. Decomposition of time series. The demand with yearly and weekly seasonal components

Any time series can be represented as a combination of its components in two ways – as a sum of the trend, seasonality and the remainder, or as a product of the components. The most relevant way of decomposition in a given situation depends on the trend variation and the seasonal component's volatility.

Apart from being used for understanding data deeper and manually analyzing its patterns, time series decomposition can also be applied when forecasting future values. For example, in a simple case, when a seasonal component is cyclic and does not change over time, and a trend component increases linearly, forecasting the data at future timestamps can be relatively simple by first applying one of non-seasonal forecasting methods to the remainder component, and then adding the trend and the seasonal components to the predicted results. Further information on using decomposition in forecasting is provided in the Section 2.2.3.

### 2.2.3 Forecasting models

This section will shortly describe several major existing forecasting models. In many cases, a detailed and highly technical explanation is required to fully understand the forecasting models. While this section provides certain level of technical detail, a reader is encouraged to seek further information by following the provided references.

#### Exponential smoothing

Exponential smoothing is one of the most popular forecasting techniques and considered to be a catalyst for numerous other widely used forecasting methods. Since the time it was proposed in the middle of the 20th century, it was successfully applied across a wide range of industries.

The simplest version of the technique, called simple exponential smoothing (SES), is based on the modified naïve forecasting methods. One of the naïve forecasting methods is to assume that the value of a variable at time T depends only on the value at time T – 1 e.g. simply is equal to it. Another naïve approach is to sum all the available data points prior to T and divide the result by their number. These approaches either assign a weight of 1 to the previous value, as in the former case, or assign equal weights to all the history

data, as in the latter case. In the case of SES, however, weights assigned to the history data are exponentially decaying, thus making more recent data points have a larger impact on the predicted values compared to more distant data points.

However, the described technique works well only for time series data without clear seasonality and trend components. A modified version of the algorithm described uses three sets of decaying weights, which define a trend, a seasonality and level components of a currently predicted value. The values are then combined to produce the forecasted value.

Multiple versions of exponential smoothing exist including additive and multiplicative in respect to a seasonality, and the ones using a dumped version of a trend component. However, it is outside the scope of the thesis to describe the methods in detail.

ARIMA

Autoregressive integrated moving average (ARIMA) is another model for time series prediction commonly used across various fields. The method can be described as a combination of three other techniques, namely differencing, autoregressive models (AR) and moving average models (MA). The techniques are briefly outlined below.

Differencing can be defined as transforming a sequence of time series values into a sequence of values representing differences between consecutive data points of the initial sequence. Essentially, the operation extracts information about how a variable changes its value over time, as opposed to what exact values the variable is equal to. This is often useful to make time series data stationary, that is to remove or reduce its trend and seasonality components. Differencing can be applied multiple times to the same data sequence, which is usually referred to as second-, third- or, more generally, nth-order differencing.

An autoregressive model represents a currently forecasted variable $y_t$ as a linear combination of its $p$ previous values.

$$y_t = c + \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \dots + \varphi_p y_{t-p} + \varepsilon_t \quad (1)$$

In Formula 1, $y_t$ is a currently forecasted variable, c is a constant, $\varepsilon_t$ is white noise and $\varphi_1 - \varphi_p$ are regression parameters that define the model. A particular autoregressive model is defined by its parameter $p$, which indicated how many previous values of the predicted variable should be used for forecasting its current value.

A moving average model, on the other hand, represents a current value of a variable $y$ as a linear combination of $q$ previous forecast errors (Formula 2).

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \dots + \theta_q \varepsilon_{t-q} \quad (2)$$

Similar to an autoregressive model, a moving average model has a parameter $q$, which defines how many forecast errors in the past should be considered when forecasting the current value of a variable.

Thus, having differencing, AM and MA introduced, ARIMA model can be described further. With ARIMA, the current value of a variable $y_t$ differenced $d$ times is equal to the sum of AM($p$) and MA($q$) applied to the $p$ and $q$ past values of the differenced sequence. Formula 3 demonstrates the previous sentence.

$$y'_t = c + \varphi_1 y'_{t-1} + \dots + \varphi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q} + \varepsilon_t \quad (3)$$

Therefore, an ARIMA model is defined by 3 parameters $p$, $d$ and $q$. The meaning of the parameters in the model was described before.

However, the outlined model is only capable of handling non-seasonal data. For seasonal data, a seasonal ARIMA model should be used, which is defined as $\mathrm{ARIMA}(p,d,q)(P,D,Q)_m$ (Formula 4)

$$\mathrm{ARIMA}\ \underbrace{(p,d,q)}_{\text{Non-seasonal part of the model}}\ \underbrace{(P,D,Q)_m}_{\text{Seasonal part of of the model}} \tag{4}$$

The parameters $P, D$ and $Q$ carry the same meaning as $p$, $d$ and $q$ but are applied to a seasonal component of data. Besides, another variable m is introduced indicating a number of data points per year. It is required for processing the seasonal component.

ARIMA models can provide time series forecasting with relatively high precision if the described parameters are assigned to optimal values. For this, software packages could be used such as the popular *auto.arima()* function in R programming language.

STL forecasting

Although the previously described forecasting models can be used to predict time series data of significant complexity, in all of the cases it is assumed that data contains only a single seasonality. However, in practice, data often contains more than one seasonal component. For example, daily time series, if recorded over a prolonged period of time, may include weekly, monthly and yearly seasonal components. Moreover, even weekly time series may present a challenge for traditional forecasting approaches since yearly seasonality is non-integer in this case (Formula 5).

$$365.25/7 \approx 52.179 \tag{5}$$

Finally, a dual-calendar seasonality can be poorly handled by trivial models. A dual-calendar seasonality is common for time series that originate from countries that follow multiple calendars, e.g. Gregorian and Hijri. In such cases, time series data exhibits more than one yearly seasonality. Moreover, unlike other cases of complex seasonality, the yearly patterns in a dual-calendar case overlap but are not necessarily nested, which introduces an additional challenge for a forecasting model of choice. Finally, in cases when a company operates in international market, even greater number of calendars may affect the data.

One of the simplest ways to tackle the described issue is by using the STL decomposition, previously described in Section 2.1.2, but with multiple seasonal components. The decomposed data is then can be used for forecasting by applying any of the previously described methods to the remainder component of the time series with adjustments made taking all the seasonal components into account.

TBATS

TBATS is another model that is capable of dealing with complex seasonalities. The acronym represents a set of techniques used in the model, namely a trigonometric seasonal component, Box-Cox transformation, ARMA errors, a trend and a seasonality. The TBATS model is outlined in this subsection with the information based on De Livera at al. [13] if not stated otherwise.

Firstly, TBATS is a modification built on a simpler BATS model, therefore, BATS will be introduced first. BATS is a model that consists of multiple simpler approaches combined, which makes it useful for forecasting complex time series data. The model includes Box-Cox transformation, ARMA errors as well as trend and seasonal components. The mentioned components form the acronym for the model i.e. BATS. The mentioned Box-Cox transformation is a type of power transformation, which is often useful when dealing with, for instance, skewed data or data with unstable variance. A one-parameter version of the transformation, which is used in the TBATS model, is defined as given in Formula 6.

$$y_i^{(\lambda)} = \begin{cases} \frac{y_i^{\lambda} - 1}{\lambda}, & \lambda \neq 0 \\ \ln y_i, & \lambda = 0 \end{cases} \quad (6)$$

One of the restrictions of having Box-Cox transformation applied to data is the fact that the input values should be strictly greater than zero. However, since time series data often has positive or non-negative values, the restriction usually does not present a problem.

In addition, a trigonometric seasonal component is included, which is represented by the first T in the model acronym. Thus, the model can be defined as in Formula 7.

$$\mathrm{TBATS}(\omega, \varphi, \mathrm{p}, \mathrm{q}, \{\mathrm{m1}, \mathrm{k1}\}, \{\mathrm{m2}, \mathrm{k2}\}, \ldots, \{\mathrm{mT}, \mathrm{kT}\}) \quad (7)$$

Here, ω stands for a Box-Cox parameter, φ stands for a dumping parameter in the trend component, p and q stand for ARMA(p, q) parameters and m1, m2, m3, etc. are used to specify seasonal periods. Finally, since the trigonometric representation of seasonal components is based on Fourier series, k1, k2, k3, etc. parameters are introduced to specify the number of harmonics required for the (*i*)th seasonality.

To conclude the subsection, it should be mentioned that TBATS model can be particularly useful in situations when one or several seasonal components do not stay the same as time progresses. On the other hand, a well-known disadvantage of the model is that it takes significant time to compute an output. [9, chapter 11.1]

Neural networks

The information presented in this subsection is based on Ian Goodfellow et al. [14] if not stated otherwise.

Artificial neural networks (ANN) are mathematical models loosely based on human brains. They are useful for representing complex non-linear functions. Recently, ANNs

gained significant popularity due to a series of successes in multiple fields, such as image classification, natural language processing, voice interpretation, machine translation, audio generation, etc. Although ANNs haven't found such widespread usage in time series forecasting solutions, it is an active area of research and certain recent publications made a significant progress. Some of the published research papers are discussed further in this subsection.

An artificial neural network usually consists of multiple layers of "neurons". A neuron here represents a mathematical function that accepts an output of a previous layer as its input and outputs a result, which, in turn, acts as an input for the following layer.

Each neuron usually consists of two parts, namely a linear part and a non-linear one. The linear part applies a certain weight w to the input, adds a bias b and returns the result. The non-linear part accepts the result of the linear part as its input, applies a function *s(z)* to it and returns the result. The function *s(z)* can theoretically be any mathematical function, but in practice, the choice is usually limited to several popular options e.g. sigmoid, tanh, etc. See Formula 8 for the sigmoid function.

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} \tag{8}$$

The first layer of an ANN is often called an input layer. On the input layer, no function is applied to the data, instead it acts as an entry point for the data. The final layer is usually called an output layer and returns a final value (or a range of values) i.e. the result of the network computation. All the layers in-between the input and output layers are called hidden layers. A neural network without hidden layers conceptually behaves in a similar way to a linear regressor. On the other hand, if an ANN has one or multiple hidden layer, it is capable of representing a non-linear function. In cases when there are hidden layers present in an ANN, it is commonly referred to as a deep neural network (DNN).

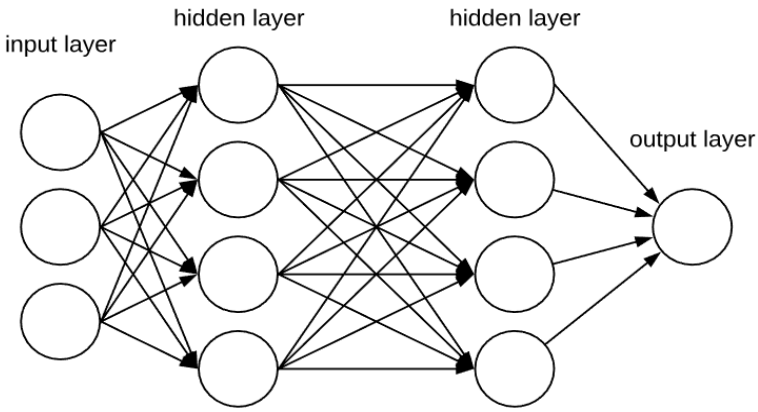



Figure 3. Feed-forward neural network. Based on Goodfellow et al. (2016) [14, chapter 6]

Weights of an ANN are not defined a priori, but instead learned during a process called training. On the high level, the training process can be described as follows. First, weights of a given neural network are randomly initialized. Then, training data i.e. data that consists of both inputs and corresponding outputs, is fed to the network. After a forward pass, a network calculates an output based on its current weight values. The produced output is then compared to the correct one, and the weights are updated in a way to “adjust” the network, so that it would produce an output closer to the correct one if given the same input. While several methods for updating weights exist, stochastic gradient descent (SGD) is by far the most commonly used one. The described procedure is performed multiple times until desirable performance is achieved, or no performance improvement is possible any further. The trained model is then evaluated on a test set.

One major issue often encountered when training a model is overfitting. Overfitting can be defined as a situation when a model adjusts its weights to predict an output for inputs specifically in a training data set at the expense of worse performance when exposed to similar data not found in the training set. Overfitting can happen for multiple reasons but training a model for an excessive number of epochs is a common one. On the other hand, underfitting is a situation when a model requires further training since its performance is not satisfactory even when exposed to the training and/or validation set. Therefore, a training process can be viewed as finding a suitable tradeoff between underfitting and overfitting.

The notions of training, testing and validation sets should also be clarified. A training set consists of pairs of input and output data. It is used to train a model adjusting its weights following the process outlined above. However, since a model can potentially adjust its weights to suit the data contained in a testing set exclusively, a validation set of data is used to evaluate the performance of the model on a data not seen during its training process. The evaluation on a validation set is often performed multiple times during a training process to keep track of “actual” performance of a model and to prevent potential overfitting. Furthermore, although validation data is not seen by a model during its training, it is still taken into consideration when adjusting the model’s hyperparameters e.g. the number of layers, the number of neurons in each layer, a non-linearity function used in the neurons, etc. Therefore, since the adjustments are made while considering results produced by evaluating on the validation set, the model can potentially demonstrate promising results with the validation set, but not necessarily with other data. Thus, to finally evaluate a model’s performance, a test set is used. A model should be evaluated on a test set only when all necessary adjustments have already been made, which demonstrates results on data never seen before.

Figure 2 depicts the simplest type of DNNs often called a feed-forward neural network. Besides this one, other types of artificial neural networks (ANN) exist as well. For example, a convolutional neural network (CNN) is an ANN consisting of multiple layers of filters. Each layer typically contains multiple filters, also known as kernels in this context. Filters of each layer are applied to an output of a previous layer by performing a dot-product operation and typically a non-linear function of choice. Due to the fact that the dimensions of the filters are usually smaller than incoming data, CNN is capable of extracting local patterns within the data. This enables the data to have a notion of location as it flows downstream through layers. Although the performed operation (a sliding window of filters being repeatedly applied to an input) is not a convolution in strictly technical terms, the described ANN is still referred to as a convolutional neural network.

Convolution in the aforementioned cases is generally said to be n-dimensional, where n is the number of dimensions over which the "sliding" operation is performed. For example, when applied to an image, convolution is usually thought to be two-dimensional, whereas when applied to a time series data, one-dimensional convolution is often used. Although CNNs are mostly used for classification problems e.g. image recognition, they

can and have been successfully applied to regression problems in general and time series forecasting in particular. [15]

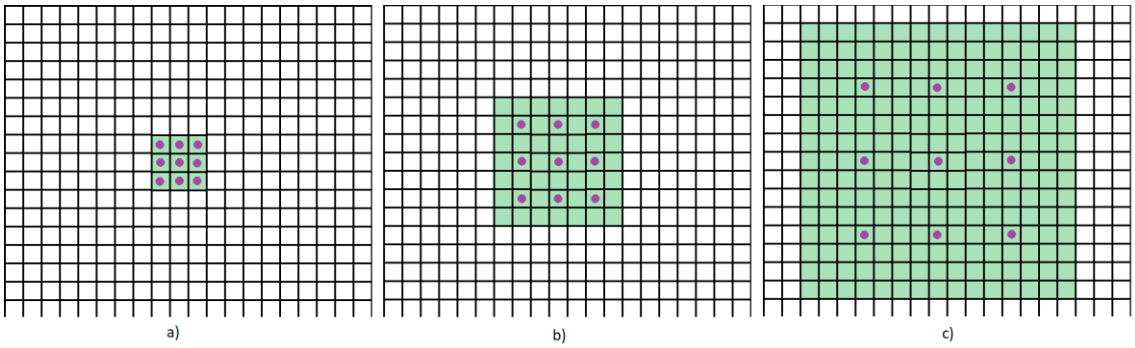



Figure 4. a) 1-dilated convolution, b) 2-dilated convolution, c) 4-dilated convolution. Based on Yu and Koltun (2016) [16]

Before some of the successful publications on using CNNs for time series forecasting can be introduced, a notion of dilated convolution should be explained. In a non-dilated convolution, a kernel is repeatedly applied to the whole input data with a striding step S. The part that has to be emphasized is that although the stride can be greater than one, thus potentially skipping some parts of the input data, a kernel of size K is always applied to K adjacent data points. Contrary to this, with dilated convolution, a kernel is applied to data points that are each L positions away from one another. [16] It is helpful to note that L-dilated convolution is, in fact, a more general definition of “traditional” convolution, any non-dilated convolution can be thought about as 1-dilated convolution. Figure 3 depicts the explained concept.

One of the advantages of dilated convolution is the fact that the receptive field of a kernel applied increases considerably while the number of trainable parameters is kept the same. Besides, unlike the similar concepts of pooling and strided convolution, resolution of data is not reduced as a result of the operation. [16] These aspects make dilated convolution useful in multiple cases.

Recently, several attempts to use one-dimensional convolution for time series forecasting have emerged. The most notable of such examples is Borovykh at al. [15]. The proposed model is largely inspired by the previously published paper about the WaveNet model [17]. Specifically, the paper proposes to use multiple dilated convolutional layers to extract information from time series data. The authors claim that by utilizing layers of

dilated convolution, it is possible to extract data over a significantly wide time frame, thus incorporating more data when doing the forecast. The publication shows promising results when applying the proposed model to financial time series data. [15]

Besides, another type of ANN, namely recurrent neural networks (RNN), has been most often used when dealing with time series data. Due to the recurrent nature of RNNs, they are useful when dealing with sequential data e.g. a sequence of words, measurements over time, etc. Figure 4 demonstrates a typical RNN schematically.

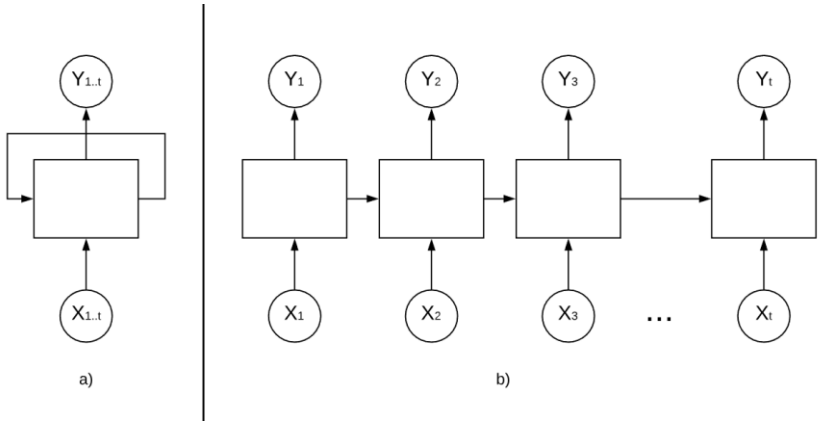



Figure 5. Recurrent neural network. Based on Goodfellow et al. (2016) [14, chapter 10]

In cases when input data is presented as a sequence, only the first element is fed to the network initially. When the second element is fed, the result of the previous computation acts as an input as well. The same process is repeated for each element in the sequence, thus incorporating information about previous sequence elements. However, when simple neurons are used, RNNs tend to perform rather poorly due to several reasons, such as the vanishing gradient. To resolve this problem, several more sophisticated computational blocks are used, among which the long short-term memory (LSTM) and the gated recurrent unit (GRU) are the most popular ones. While describing LSTM and GRU in technical detail is beyond the scope of the thesis, it is sufficient to say that they often perform better at carrying information from previous sequence elements.

Several recent publications demonstrate promising results when applying RNN models for time series forecasting. For example, Siami-Namini et al. [18] claim that the proposed

LSTM-based model significantly outperforms the ARIMA model. Nevertheless, traditional approaches still demonstrate dominance over the described deep learning methods [19].

### 2.2.4 Combining forecasting models

While it might appear as obvious that selecting the strongest model out of multiple options would provide the highest precision of forecasting, multiple studies suggest that a combination of models, which is often referred to as an ensemble, should be used instead. [20, 21, 23, 24] For example, Clemen [20] concludes that a combination of forecasts almost always gives better results compared to individual models. Besides, Armstrong [21] in his review claims that a forecast combination should be used in a majority of cases, especially when a high degree of uncertainty about what model is more correct is present, since it typically results in a solution that is at least as accurate as the worst individual model in the combination. This claim was also supported by a number of other publications [22].

Furthermore, a significant amount of research has been dedicated to finding the most efficient way of combining forecasting models. According to the review by Clemen [20], a simple averaging performs well in a majority of contexts. Rob J. Hyndman and George Athanasopoulos [9, chapter 12.4] in their book support this claim as well. Similarly, Armstrong [21] suggests that when no knowledge about the field in which forecasting is performed is available, using simple averaging is a reasonable option. He also claims that using trimmed mean is beneficial when at least five forecasting models are available [21]. The trimmed mean is obtained by averaging all available models except for the ones showing the highest and the lowest results i.e. the two extreme models. Besides, an effort is noticed in the academia for developing various frameworks for combining time series forecasts. For instance, Hui Zoua and Yuhong Yang [24] propose to use the AFTER algorithm that convexly combines results of several prediction models.

Finally, multiple studies suggest that although individual forecasting solutions might provide better results than model averaging, the latter usually results in a more stable solution, which has a lower chance of producing critical mistakes. Thus, many practitioners might benefit from forecast model combinations, since it is often more crucial in real-life

situations to have stable results rather than unstable ones, even at the expense of potentially better benchmark outcomes. [21, 22]

# 3 Current state analysis

This section describes the production planning and demand forecasting methods and practices currently used by the client company. A number of disadvantages and inefficiencies in the current process are identified, and problems to be resolved are highlighted.

Sales campaigns

Before the current practices for demand forecasting and production planning are outlined, the notion of a sales campaign should be explained. During a campaign, certain customers are able to buy certain products for discounted prices. Since the campaigns are usually limited in terms of the time period during which orders can be placed, it provides an additional incentive for customers to order larger amounts of goods than is normally done. The campaigns play an important role in the demand forecasting since sales volumes of products that are being campaigned usually increase by a significant amount.

Current demand forecasting practices

At present, demand forecasts are produced manually by a group of experts with deep domain knowledge and multiple years of experience in the industry. The forecasts are made regularly, up to several times a week, and incorporate the information about already existing sales orders, sales campaigns, previous year sales data that corresponds with the same time of the year e.g. the same month, week number, day of the week, etc., and about general economic situation on the market. Moreover, national holidays considerably affect the sales volumes as well. For example, sales amounts increase significantly several days before Christmas each year. Such holiday effects are currently taken into consideration as well, although only qualitatively since no formal methodology is used to incorporate the information.

Besides, the tools used for the forecasting by the experts are currently limited to Microsoft Excel, which is used simply for storing the information. Hence, the whole forecasting process is performed by people, who are prone to be influenced by a variety of factors. This, eventually, can result in significant imprecisions of the produced forecasts.

Current production planning practices

Similar to the demand forecasting practices, production planning is currently performed manually by a group of experts. The production planning process is mainly affected by demand forecasts, available workforce as well as by the existing customs. For an example of the existing custom, manufacturing of product N can be considered. The actual product's name is not disclosed due to the company's non-disclosure policies. The product N is currently manufactured twice a week on certain weekdays. While there are multiple factors that contribute to the schedule, a habit of the management personnel is among them. Moreover, no other technologies are used in the production planning process other than the Microsoft Excel software.

Problems similar to those of the forecasting practices can, thus, be identified. The mentioned issue of subjectivity equally applies to the production planning process as well. This and several other issues that are caused by the current methodology are further discussed in the last part of the section.

Problems to solve

In this subsection, the issues that result from the described practices are listed. As already mentioned, the subjectivity of the people currently producing the demand forecasts and production plans is only a single problem that needs to be resolved. Another significant issue is unjustified overproduction. Since underdelivering a product to a customer might have serious consequences for the manufacturing client company, including severing all business ties with the customer, it is crucial to always meet the demand. This, in turn, often results in producing more items than is subsequently sold. Since the shelf-life of the manufactured goods is limited, the overproduction causes the manufacturer to either sell the excessive goods for significantly lower price or, in the worst cases, to discard the products completely, which results in diminished profits. Thus, overproduction

is another major problem that catalyzed the decision to order the forecasting and planning software solution.

Moreover, the inefficiency of the production process is another issue to tackle. In addition to paying for the workforce when the production lines are in operation, preparing a line for production as well as halting a production process involves extra costs. In addition to this, the changeover process, i.e. the necessary maintenance required when switching production of one product to another on the same production line, requires time and, therefore, additional spending. Since, at present moment, planning is performed manually, the production process is suboptimal in terms of the costs. Therefore, the production planning system should optimize the process by reducing the described costs whenever possible.

Finally, the production planning system will save hundreds of work hours for the domain experts, enabling them to perform other activities. Due to the fact that the experts usually occupy managerial positions and possess rich domain knowledge, automating the forecasting and production planning processes will save considerable amounts of money for the client company. Thus, the importance of the production planning system to the company has been shown.

# 4 Technologies and methods

## 4.1. Data

The data used in the solution was obtained from the client company. The data contains sales quantities for all of the products produced by the client company for the time period from 30th October 2016 to 15th June 2018. Besides the sales amounts, the data also contains the information about the campaigns. Finally, each data point contains the information about a customer who made the purchase. This also allows for additional data analysis.

The described data was collected from the client's ERP system via SQuirreL SQL Client software, which is introduced at the end of the Section 4.2. The database underlying the ERP system was accessed and the necessary data was downloaded to local CSV files.

## 4.2. Technology stack

### Prophet

Prophet is a forecasting tool, which provides APIs for Python [25] and R [26] programming languages. The project was open-sourced by Facebook in early 2017 and quickly gained popularity among analysts and practitioners. The forecasting tool is especially useful for business forecast problems. In particular, Prophet shows promising results when data has several distinct properties such as strong seasonal components e.g. weekly, yearly, etc., presence of multiple "holidays", which influence behaviour of time series values around them, and a trend being a non-linear curve that can change due to important business events [27]. More importantly, Prophet is also capable of handling a limited number of outliers and missing data points [27]. All the described properties make the tool suitable for the case of the client company.

Another feature that distinguishes Prophet from other forecasting tools is its ease of use. In fact, with data of reasonable quality, even people without previous training for time series forecasting are able to produce quality results since Prophet's API is intuitive and can be understood by non-experts. On the other hand, there are plenty of parameters that can be configured, which provides possibilities for practitioners with domain knowledge to adjust the default forecasting model to one's needs. [27]

Finally, methods that are used internally by Prophet are described briefly. Without diving into technical details, it can be mentioned that Prophet is an additive regression model that incorporates a data trend, a seasonality and holidays as its components. The trend is defined as a linear or logistic growth curve, where a user is able to specify which of the two types should be used in a given case [28]. The forecasting tool automatically detects trend change points in historical data and assumes that a similar trend and its rate of change is preserved in the future [28]. The yearly seasonal component is defined in terms of Fourier series with Fourier Order being a configurable parameter [29].

Keras

Keras is a deep learning Python library, which provides an easy-to-use API and is capable of using either Tensorflow, Theano or CNTK as its engine [30]. Keras is distinct from other deep learning libraries mainly due to the following factors. First of all, it is user-friendly, allowing for fast-prototyping and enabling practitioners with little experience to experiment with their ideas and create fast implementations. Secondly, it is extensive enough to provide experienced practitioners with the capabilities to implement complex ANNs. Finally, it can be executed on both a CPU and a GPU taking advantage of either without a need for complex configuration or manual optimization [30]. All of the described properties make the library suitable for the task at hand.

Pandas

Pandas is a data analysis and data structures library for Python with special emphasis on performance and ease of use. The high performance is achieved by the fact that some of the most important parts of the library are written in Cython or C programming language [31]. The library provides a rich and powerful API for handling in-memory data structures, called “Data Frames”. Besides, the library has rich functionality oriented specifically for time series processing and analysis [31]. For example, Pandas provides functions for automatic interpolation of time series data, calculating moving window regressions as well as for aggregating data with functions such as “group by”. Finally, the library is used by both commercial and governmental organizations in a variety of industries including finance, neuroscience, advertising, etc. [31]

Pipenv

Pipenv is a packaging tool and a dependency manager for Python projects. It makes one’s workflow significantly simpler by providing automatic management of Python virtualenv. Besides, it automatically creates and updates *Pipfile* and *Pipfile.lock*, ensuring truly deterministic builds and clear dependency management. While *Pipfile* contains all the dependencies of a given project together with their version numbers, *Pipfile.lock* contains the whole dependency tree with hashes and other necessary information to make sure the dependency graph can be deterministically recreated whenever needed.[32]

Statsmodels

Statsmodels is a Python module that provides classes and functions for multiple statistical models. The models include regression, generalized linear models (GLM), robust linear models (RLM), discrete choice models and contrast analysis utilities [33]. Besides, the module contains classes and functions for time series analysis, which include vector autoregressive models (VAR), univariate autoregressive moving average models (ARIMA) as well as exponential smoothing models of various complexity e.g. simple and Holt Winter's version of it [34]. The mentioned Holt Winter's exponential smoothing model is used in the proposed solution described in the Section 5.2.

The *ExponentialSmoothing* function accepts the training data as a parameter and allows specifying the trend and seasonality types e.g. additive or multiplicative. The object returned by the function allows calling its *fit* function, which accepts multiple optional parameters allowing further customization of the exponential smoothing model. One of the parameters is *optimized*, which – when set to True – enables an automatic search for the optimal model parameters aiming to maximize the log-likelihood. [35]

Forecast

Forecast [36] is a widely used package for R programming language that contains functionality for analyzing and forecasting time series data. Among tens of other functions, it contains a fully automatic implementation of ARIMA, which enables searching for ARIMA parameters that are optimal for the given data [36]. Moreover, it includes the implementation for the TBATS forecasting method [36], which was described in Section 2.2.3 of the thesis. Finally, the *stlf* function allows a user to perform STL decomposition and forecasting based on one of the simpler methods with a single function call. The mentioned functions make the package especially relevant in the context of the task, despite the fact that, unlike the other technologies described, the package is implemented in R programming language [26] and not in Python [25]. In addition to the already provided information, further description of the *tbats* and *stlf* functions is presented in the following two paragraphs.

The *tbats* function implements the described earlier TBATS forecasting method. It allows passing all the described TBATS parameters as the function parameters, thus satisfying

a wide range of needs. More importantly, the function allows leaving one or several parameters undefined, which will result in searching for their optimal values [36]. The function utilizes parallel processing by default, thus considerably improving its performance [36].

Meanwhile, the *stlf* function performs the STL time series decomposition, applies one of its forecast methods to the data with removed seasonalities, and finally, applies the seasonality back to the forecasted non-seasonal data. While the method used for forecasting seasonally-adjusted data can be specified, the corresponding parameter can be left undefined allowing the function to make the choice for the forecasting method used [36].

RPy2

Due to the fact that majority of the logic related to time series forecasting for the client's company is implemented in Python, a tool is required that allows utilizing useful aspects of R programming language, such as the widely used forecast package, while keeping Python as the main programming language. RPy2 is a utility that resolves the described issue. It provides an interface between the two programming languages enabling users of Python to utilize the full capabilities of R programming language and its numerous libraries. In the context of the implementation described in the next section, the tool is used to access the *stlf* and *tbats* functions of the forecast package. [37]

Pyramid

Pyramid is an open-source project developed to bring the functionality of the popular *auto.arima* function of the forecast package to Python. The implementation provides an interface similar to that of the forecast package. Internally, Pyramid utilizes other Python libraries, such as statsmodels and scikit-learn. [38]

The function searches for the optimal ARIMA parameters based on AIC or BIC values of potential candidate models. Since searching among all the possible values of the ARIMA parameters would be infeasible in terms of time, the actual search resembles a grid search in that it does not try all the potential parameter sets, but only a limited number of them. Moreover, the upper and lower boundaries for each of the parameter values can be specified, thus further reducing the search space.

In addition to this, the *auto_arima* function allows passing an additional time series as an extra regressor. The feature is used for passing the described campaigns data as a regressor due to its significant effect on the forecast.

SQuirreL SQL Client

SQuirreL SQL Client provides functionality for inspecting the structure of JDBC compliant databases, retrieving data from database tables as well as for issuing SQL commands [39]. The software, which utilizes pluggable drivers, can be used for accessing a variety of different databases due to its extensible nature. A user is able to add her own driver to gain access to a database that is not supported by default.

## 4.3. Evaluation methods

Choosing a method for discriminating one model in favour of another is an important decision that might dramatically influence one's results. This section describes the evaluation methods used in the final year project and provides reasons for why they were selected. Besides, the time horizon of choice and cross-validation techniques are discussed as well. The information provided in this subsection is based on chapter 3.4 of Hyndman and Athanasopoulos. [9 chapter 3.4]

Evaluating a forecasting model usually consists of estimating errors of potential candidate models and subsequently selecting a model that demonstrates the smallest average error when applied to the data of interest. Forecast errors can be divided into multiple groups, such as scale-dependent errors, percentage errors and scaled errors. The scale-dependent errors include such popular options as the mean absolute error (MAE) and the root mean squared error (RMSE).

$$MAE = mean(|error(t)|) \quad (9)$$

$$RMSE = \sqrt{mean(error(t)^2)} \quad (10)$$

In Formulas 9 and 10, $error(t)$ is equal to an absolute difference between the actual and the forecasted values of a variable at time $t$. The percentage error group includes such choices as the mean absolute percentage error (MAPE) and the symmetric mean absolute percentage error (sMAPE) (Formula 11). This group of errors has been used in multiple forecasting competitions including M3-competition [40] and several Kaggle challenges [41, 42]. Nevertheless, several issues associated with the errors make them undesirable when selecting an evaluation method of choice. Firstly, in cases when $y(t)$ is close or equal to zero and $y'(t)$ is reasonably correct, meaning it is close or equal to zero as well, the formula produces either an undefined value, due to the division of zero by zero, or a potentially very big value. In Formula 11, $y'(t)$ is a forecasted value of the variable $y$ at time $t$. Secondly, the result of sMAPE can be negative, which is an unexpected outcome when an "absolute" error is desired.

$$sMAPE = mean(200|y(t) - y'(t)| \,/\, (y(t) + y'(t))) \quad (11)$$

Instead of using the described model evaluation methods, Hydman and Koehler [40] proposed to use the mean absolute scaled error (MASE), which avoids the previously discussed issues. MASE is defined as given in Formula 12, where $e(t)$ is a difference between forecasted and actual values at time $t$, T is the number of data points in the training data set, m defines seasonality of the data i.e. number of data points in a seasonal period, and $Y_i - Y_{i-m}$ represents a difference between 2 data points in the time series being positioned $m$ points apart from each other. Effectively, this error shows how much better or worse a given forecasting model is compared to a naïve seasonal forecast applied to the training data, providing the data of interest, in fact, has a seasonal component.

$$MASE = mean(|\frac{e(t)}{\frac{1}{T-m}\sum_{i=m+1}^{T}|Y_i - Y_{i-m}|}|) \quad (12)$$

Thus, for the purpose of the task at hand, both MASE and RMSE are chosen to be used. The former is used due to its claimed superiority and correctness, while the latter still is a popular choice in a variety of forecasting competitions [43-45] and, therefore, should be considered as well. Nevertheless, the model selection and evaluation are performed based primarily on MASE.

A time horizon of seven days is selected when doing the evaluation. The reason for the choice is primarily based on the need of the client company and the specification for the production planning system. The forecast which is conducted manually by the experts at present is made with a horizon of seven days. Besides, the specification for the planning system requires it to produce a detailed production plan for at least the next seven days. Therefore, taking both the existing habit of the client company's personnel and the specification into consideration, a model should be selected on the basis of evaluating its performance with a seven-day horizon.

Finally, the way cross-validation is performed in the evaluation process should be explained. When evaluating a model, the data is repeatedly split into training and test sets, where the size of the test set is repeatedly incremented by one. The procedure is often referred to as “evaluation on a rolling forecasting origin” [9 chapter 3.4]. The initial split point is set to 2nd April 2018 and the “rolling” continues until the split point reaches 15th May 2018. On each training/test split, the training set is fit to each of candidate models, and the produced forecasts are evaluated based on the seven-day horizon past the split point. After the cross-validation is performed, average MASE values are compared to identify the best model. It is also worth mentioning that the evaluation is conducted separately for

each product. Refer to Figure 5 for a visual representation of the described cross-validation process.

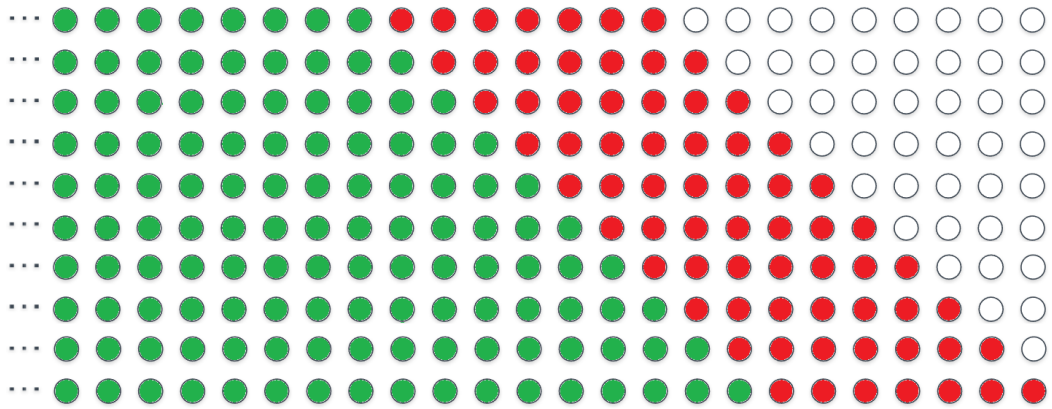


Figure 6. Cross-validation. Based on Hyndman and Athanasopoulos (2018) [9, chapter 3.4].

Here, the green dots represent data points included in the training set, while the red dots represent the test data points i.e. data points which are used when calculating one of the previously described errors.

# 5 Implementation

## 5.1. Preprocessing the data

The current subsection describes transformations applied to the raw sales data before it is provided to the forecasting models. The preprocessing transformations do not differ for various products, but some differences exist in the transformations depending on what model the data is provided to.

After the raw data is collected and transformed to the required format, it is passed to the function that removes outliers, which naturally introduces missing values to the data. Since most of the outlined models expect the data not to have any missing values i.e. to have a strict daily frequency in the given case, it is necessary to perform substitution of the outliers, rather than simple removal. However, as mentioned earlier, the Prophet documentation claims that the model is capable of dealing with missing values efficiently,

thus not requiring one to handle it manually. Furthermore, after performing several benchmarks for the Prophet model providing it with the data with manually substituted outliers and with outliers simply removed i.e. without and with missing values, the results showed that leaving empty spots in the data was a preferable way of dealing with the issue, but only when dealing with the Prophet model.

For substituting the values, a rather simple approach was chosen. It is important to note, however, that the method will change in the future. In the current implementation, the outliers are replaced by medians of the same day of the week. For example, if an outlier that should be replaced occurred on Wednesday, then it is replaced by the median computed across all the Wednesdays within the data. The median is used instead of the mean due to it being more robust to shifts caused by the outliers. In other words, mean is affected by the outliers to a larger extent compared to the median.

```
def remove_outliers(data, m=2.):
    d = np.abs(data - np.median(data))
    mdev = np.median(d)
    s = d / mdev if mdev else 0.

    return data[s < m]
```

Listing 1. The function that removes outliers from the numpy array using the data's median value. Copied from Bannier (2014) [46]

For a similar reason, a median is used when deciding whether a particular data point is an outlier. The Python implementation of the algorithm described in this paragraph can be viewed in Listing 1. First, the data is shifted by the value of its median, thus making it roughly zero-centred. Then, a new median *mdev* of the shifted data is calculated. In the perfect cases, the new median is equal to zero, but in most of the practical examples, it is not. After this, the *S* array having the same number of elements as the original data set is computed. Specifically, an element of the *S* array is equal to the value of the corresponding data element previously shifted by the initial median and divided by the value of the newly calculated median *mdev*. Finally, all the values of the initial dataset for which the corresponding values of the *S* are smaller than a chosen threshold *m*, which is provided as a parameter, are identified as outliers and removed from the dataset (see Listing 1).

```
def remove_outliers(data, m=2., replace=True):
    d = np.abs(data - np.median(data))
    mdev = np.median(d)
    s = d / mdev if mdev else 0.

    if replace:
        medians = np.arange(7)

        for i in range(0, 7):
            medians[i] = np.median(data[i::7])
            data[i::7][s[i::7] >= m] = medians[i]

        return data
    else:
        return data[s < m]
```

Listing 2. The function that replaces outliers with data's median values taking weekly seasonality into consideration. Adapted from Bannier (2014) [46]

However, since the outlier removal is performed with data replacement for most of the models, Listing 2 is provided to demonstrate the version of the algorithm in which the replacement is performed instead of mere removal of the outliers. As already explained, the method serves as a necessary minimum required to proceed with the task at hand and will be substituted with a more relevant algorithm in the future.

Besides the outlier handling, the input data is also normalized. Before normalization can be performed, the data is split into training and test subsets. Since the split is performed multiple times during the cross-validation process, the normalization of the data is performed repeatedly as well. The reason for this is that when normalizing training and test datasets, the mean and/or the standard deviation that are used in normalization formulas must be calculated based on the training set only. Listing 3 demonstrates the function used in the implementation.

```
def normalize(train_set, test_set):
    std = train_set.std(axis=0)
    train_set /= std

    test_set /= std

    return train_set, test_set
```

Listing 3. Data normalization function. Both training and test sets are divided by the standard deviation calculated from the values of the training set only.

As can be seen in Listing 3, only the standard deviation is used in the function i.e. only the amplitude is adjusted. It was decided that shifting the data by the value of the mean

is unnecessary since all of the data points occupy approximately the same range of values throughout the observation period.

After this, the normalized data with the outliers removed is fed into the forecasting models. Section 5.2 describes the parameters of the forecasting models that were introduced in Section 2.2.3.

## 5.2. Forecasting model descriptions

This section describes in detail the models used in the implementation. Besides, the way the model combination is implemented is described as well. The models to be discussed are the Prophet model, the 1D-convolutional neural network model, the ARIMA model, the exponential smoothing model, the TBATS model and the STLF model.

### Prophet model

The Prophet model uses the Prophet forecasting tool described in Section 4.2. Most of the parameters of the model are set to their default values. Among the exceptions are the yearly seasonality boolean parameter, which is explicitly set to true, and sales campaigns time series, which is specified as an additional regressor. Specifying the campaigns as an additional regressor is important because of their significant impact on the sales volumes, which makes it a crucial piece of information that should be considered when forecasting the demand. The explicitly specified yearly seasonal component is important to force the Prophet model to consider the yearly seasonality, which can be clearly observed in the data. It is necessary to explicitly specify the parameter due to the fact that the data contains only a single full year cycle, because of which Prophet might decide to ignore its yearly seasonal component by default. After some experimentation, it was concluded that taking even this single yearly seasonality into consideration improves the accuracy of the model.

### Convolutional neural network model

The one-dimensional convolutional neural network model was initially inspired by WaveNet model [17]. After further research was done, Borovykh et al. [15] was the main reference publication when experimenting with hyperparameters and architecture of the CNN. In its final implementation, the model consists of four layers: a one-dimensional convolutional layer with dilation rate of 2, other two one-dimensional convolutional layers with dilation rates of 4 and 8 and a "dense" layer, which also acts as an output layer. The activation function used in all of the convolutional layers is the rectified linear unit (ReLU), with the exception of the last dense layer, which uses the linear activation. Each of the convolutional layers contains 55 filters, where each filter is of length 5. The size of the window that is used for input data when a forecast is required to be produced, or when training the data, is 50. This means that to forecast a value at time *T*, prior values between the times of *T – 50* and *T – 1* inclusive are used as input to the network. When it comes to training parameters, the mean squared error (MSE) is used as the loss function, the Adam algorithm is used as an optimizer, and the training is performed for 40 epochs. Lastly, a batch size of 128 is used. Most of the listed hyperparameter values were obtained experimentally after running multiple training cycles and comparing the model's performances on the validation dataset. It also should be mentioned that some of the hyperparameters, such as the number of training epochs, for example, should ideally change depending on a product whose data the network is operating on. This and other improvements are discussed in Section 6.2.

ARIMA model

The ARIMA model uses the *auto_arim*a function provided by the Pyramid library that was described in Section 4.2. In the given case, start values of the ARIMA parameters *p* and *q* are set to 1, while maximum allowed values for the parameters are both set to 5. The ARIMA seasonality parameter *m* is set to 7, and the parameter *d* is strictly set to be 1, which means it will not be searched. Furthermore, since a seasonal variant of the ARIMA is used, the restrictions are also specified for the seasonal parameters *P* and *D*. Specifically, the start value of *P* is set to 0, whereas *D* is hardcoded to the value of 1. Finally, the *xreg* parameter of the *auto_arima* function is used to provide the campaign data as an external regressor. See Figure 6 for the full list of the parameters passed to the function.

```
def get_arima_model(timeseries, xreg=None):
    return auto_arima(
        timeseries,
        exogenous=xreg,
        start_p=1,
        start_q=1,
        max_p=5,
        max_q=5,
        m=7,
        start_P=0,
        seasonal=True,
        d=1,
        D=1,
        trace=True,
        error_action='ignore',
        suppress_warnings=True,
        stepwise=True
    )
```

Listing 3. *auto_arima* function call with all the passed parameter values listed

The values of the parameters *d* and *D*, as well as the *start*- and *max*- parameters presented in Listing 3, were selected based on experimentation results. The parameter *m*, however, was set to 7 since the weekly seasonal component is the major one among the seasonalities present in the data. Since each week contains exactly 7 data samples, each associated with a day, the value of the parameter is, thus, justified.

Exponential smoothing model

The Statsmodels module is used for the exponential smoothing forecasting model. Both the *trend* and the *seasonality* parameters are set to be additive since the data does not display any of the characteristics common for multiplicative components e.g. the seasonality component's amplitude increasing together with the trend. Moreover, the *dumped* trend parameter is set to true. If, on the other hand, the trend is not considered to be dumped, the model tends to produce forecasts with exaggerated demand levels for some of the products. Finally, when the *fit* function of the returned object is called, the parameter *optimized* is set to true to search for the optimal parameter values of the *fit* function

TBATS model

The *tbats* function of the forecast package is used for the TBATS forecasting model. In the current implementation, no parameters are provided to the function allowing it to perform a complete exploration of the whole search space. Since the search is executed in a reasonable amount of time, this can be tolerated.

STLF model

Similar to the TBATS model, the forecast package is used for the STL forecasting model. Again, no optional parameters are provided, which enables the function to select the optimal forecasting method applied to the data after it is decomposed. Also, as in the case of the TBATS model, rpy2 library is used for accessing the functionality of the R package using Python programming language.

Model combination

Finally, the model combination should be discussed briefly. Multiple forecast combination approaches were tried for the data. Specifically, simple averaging and weighted averaging based on various model criteria, such as AIC, BIC, MASE, RMSE, etc., were considered. While some products' data benefited from averaging using weights, forecasts for many products exhibited worse results when averaging by the before mentioned criteria. Therefore, it was decided to use the simple averaging instead. The results of the averaging as compared to the performance of the individual models, as well as the suggestions for potential improvements in the model combination processes are outlined in Section 6.1 and Section 6.2 respectively.

## 5.3. System architecture

This subsection describes the high-level architecture of the production planning system and explains the relationships between its components. Besides, typical use case scenarios are outlined in the subsection. There are four major components in the system that will be described further, namely the Java back-end, the database, the ERP system hosted by the client company, and the Flask microservice, which provides the REST API

for the demand forecasting functionality. Figure 7 demonstrates the described architecture. It is important to note that the system is not implemented at the time of writing, which makes the proposed design susceptible to changes and improvements.

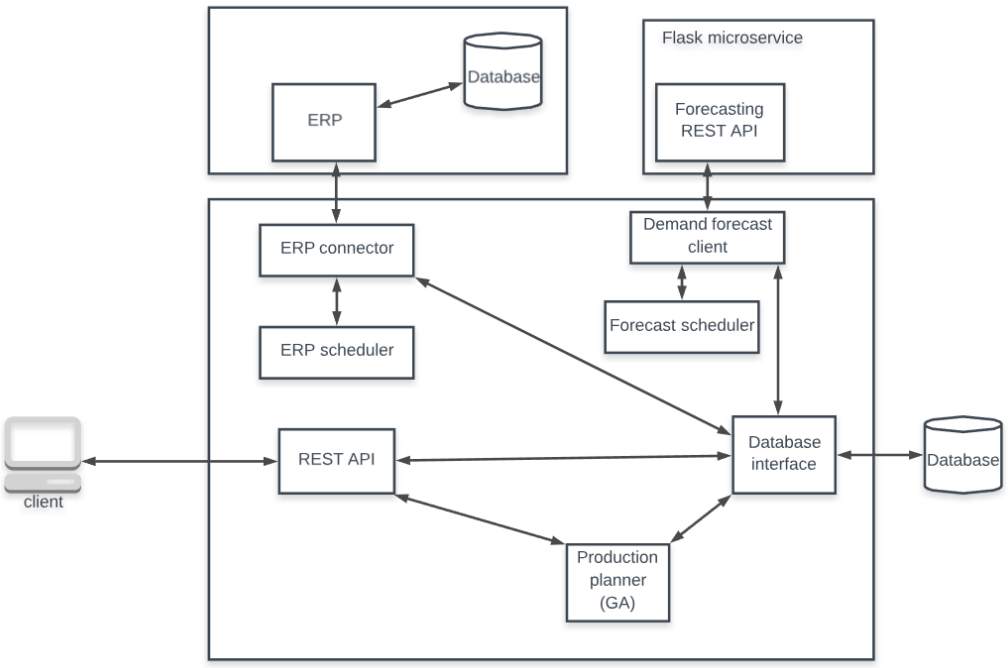



Figure 7. The production planning and demand forecasting system architecture

As can be seen from Figure 7, the back-end consists of several major components. First of all, it provides the REST API, which is used by clients to utilize the production planner's functionality. For example, the API can be used to request a demand forecast or a production plan for a specified time period. Secondly, the back-end contains the ERP system connector, which is responsible for communication with the client's ERP system retrieving such information as sales orders and sales amounts as well as sales campaigns. The connector is driven by the ERP scheduler, which fetches the data from the ERP system at regular time intervals utilizing the described connector. After the data is fetched, the local database is synchronized with the new data. The described actions are necessary since it is considerably faster to fetch the data, once it is required, from the local database rather than from the one that is associated with the ERP system.

Besides, the service contains the demand forecast scheduler, which, by using the demand forecast client, sends requests to the forecasting API to generate demand forecasts for specified products for specified time periods as needed. The microservice receives the sent data, which usually consists of past sales and campaign time series, and inputs it to the forecasting model. The forecasts are returned by the REST API in JSON format, transformed to Plain Old Java Objects (POJOs) and persisted to the local database for future use. In addition to this, the demand forecast client periodically sends newly emerged data to the microservice to retrain the existing models. Similar to the example of the ERP scheduler, having the operations executed in the background, allows the forecasts to be retrieved directly from the local database, once they are required.

More importantly, the production plan optimizer is another component of the system. It is responsible for generating a detailed production plan for a specified time horizon when provided with the demand for the time period as well as with information on the products of interest and initial storage levels. Furthermore, the plan optimizer is capable of recalculating an already existing plan given a set of so-called production targets i.e. user-specified production volumes for certain days for certain products that are not allowed to be changed by the planner. This forces the planner to recalculate the detailed production plan taking the user's requirements into consideration. The described use case is an important one since the client company insisted on having the functionality to manually customize the production plans as required.

Finally, the service contains the component responsible for interacting with the database. All of the components listed previously interact with the database via the interface to both retrieve the data required for their operations as well as to store data, which is typically a result of their operations.

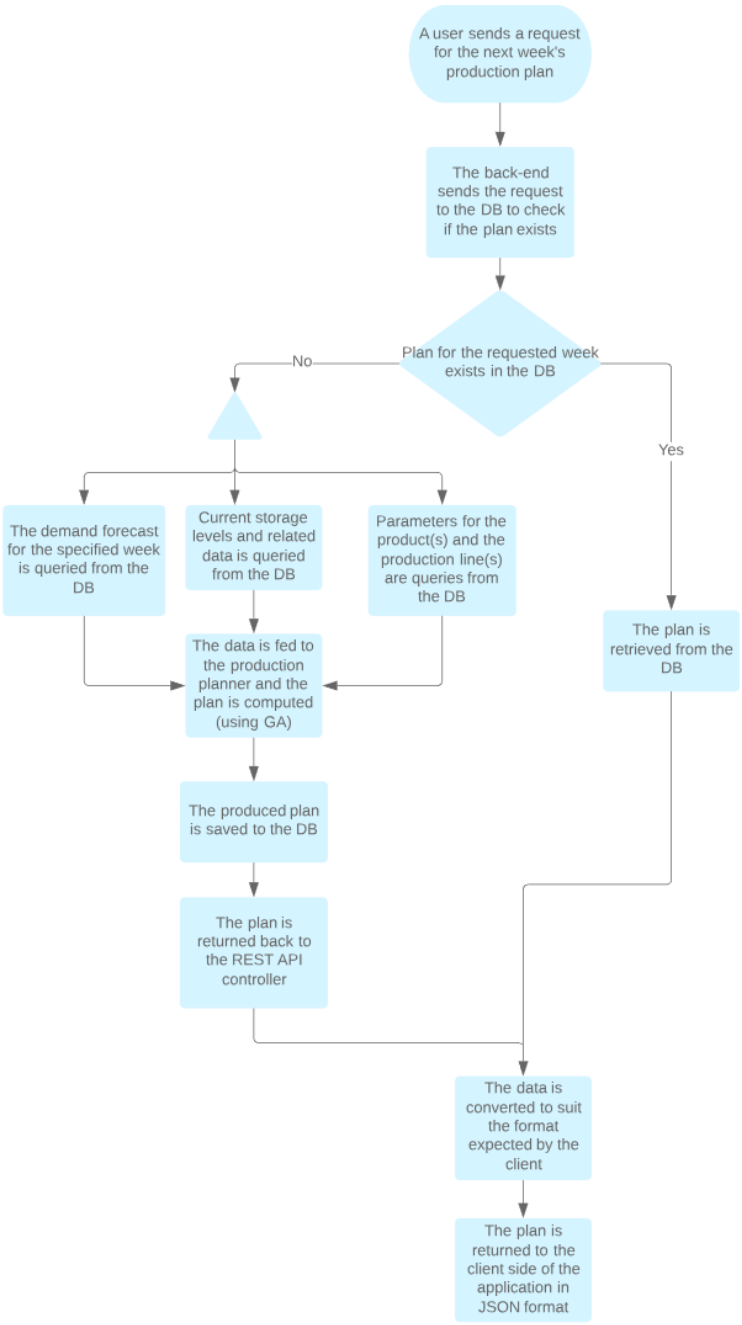



Figure 8. A typical use case scenario of the production planning system. A user requests a detailed production plan for a specified time period in the future.

Having the system described, typical workflows should be presented as well. Usually, an interaction with the system starts with a user making a request from the client side of the application. For the given example, it is assumed the user requests a production plan for a particular week to be displayed. First, the client sends the request to the REST API. Then, the controller queries the database and checks whether the plan has already been generated previously and stored. If this is the case, the plan is retrieved from the database, the necessary data conversions are performed, and the plan is returned to the client. Otherwise, the database is queried for the other data necessary to produce the plans. The data includes parameters of products of interest, information about production lines that are used to produce the products and the forecasted demand for the following several weeks. This data is then provided to the plan optimizer component, which calculates the production plan for the requested period. When the optimizer is ready, the production plan is saved to the database for future use and returned to the client. Figure 8 demonstrates the described logic.

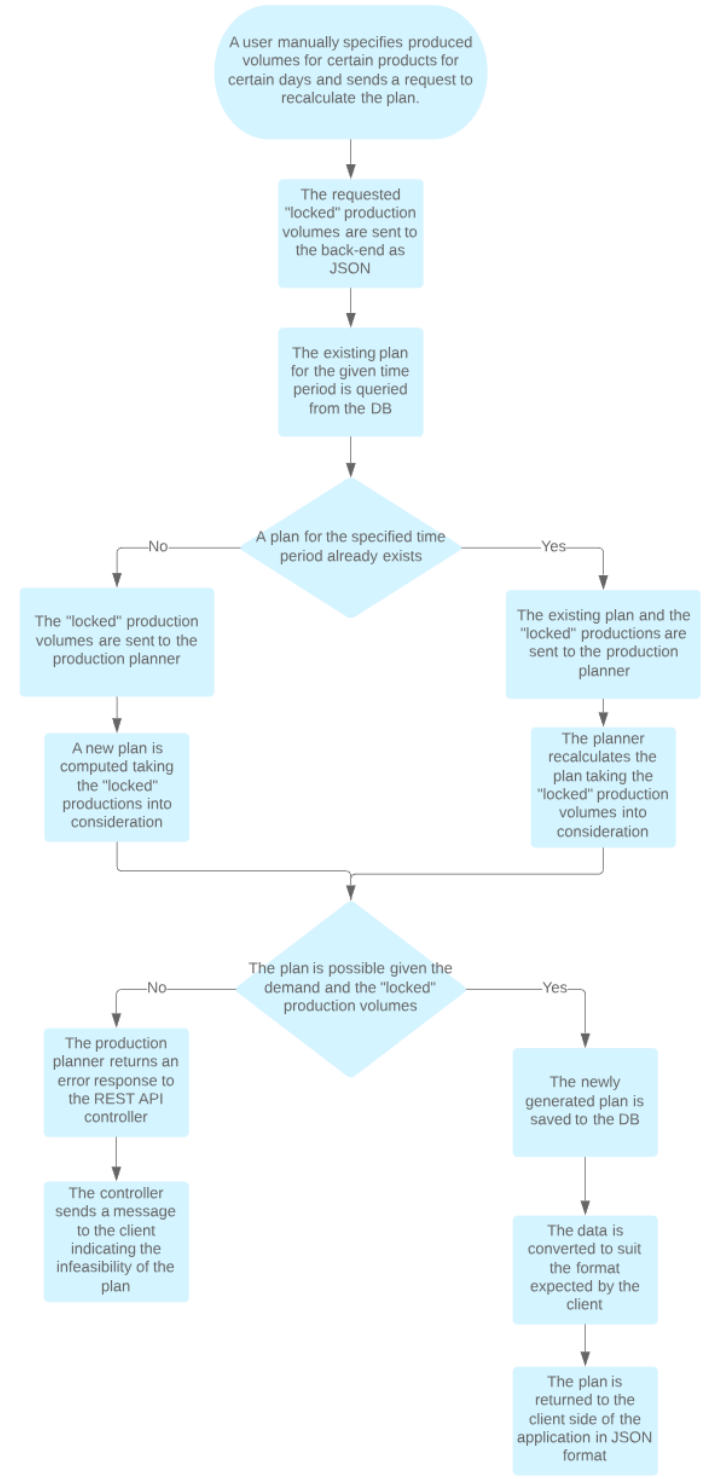



Figure 9. A typical use case scenario of the production planning system. A user manually specifies the desired production levels for certain products for certain days and requests recalculation of the production plan.

Another use case is related to the already mentioned feature of the plan optimizer to recalculate the existing plan based on custom specifications of production volumes for specific products. Such a scenario usually starts with a user observing a previously generated production plan and wishing to change i.e. “lock” certain production amounts. When the user is ready with specifying the production targets, the request with the relevant data is sent to the back-end. Then, the database is queried to retrieve the plan that should be reprocessed. After this, the plan optimizer is executed accepting as an input the existing plan and the new production targets. After the recalculation, the newly generated plan is persisted to the database together with the production targets associated with it, and then returned to the client represented in JSON format. Figure 9 depicts the flowchart describing the outlined scenario.

# 6 Results and discussion

This section contains the results of the benchmark performed for each of the previously discussed models. The numerical results presented are followed by related discussions. Besides, the second subsection contains suggestions and ideas for potential future improvements.

## 6.1. Benchmark results

As mentioned previously, the benchmark consists of a number of forecasts, each with a horizon of 7 days. The forecasts were executed for 9 products that had large enough sales amounts in the previous 2 years. For each of the products, the forecasts were produced using the “rolling” training/test split of the historical data. The cut-off point was given 44 different values between 02.04.2018 and 15.05.2018 inclusive, thus producing 44 forecasts for each of the benchmarked products for each of the models. Moreover, for each of the 2772 forecasts produced, the previously described errors, namely RMSE, MASE and SMAPE were calculated.

However, before presenting the average values for the errors, it is useful to discuss some of the individual forecasts that, hopefully, represent the majority of the predictions produced. Since the actual product names cannot be revealed due to the non-disclosure

agreement with the client company, the products discussed are labelled as Product A, Product B, etc. Moreover, the client company objected to revealing the actual sales figures, because of which the normalized scale is used in the current section i.e. the actual sales values are replaced by them being divided by their means.

Appendix 1 demonstrates the forecasts for all the six models listed in Section 5.2, executed for the dates between 03.04.2018 and 09.04.2018 inclusive. In addition to the forecasts of the models, the forecast that was produced as an average of the models for each of the data points is depicted as well. As can be seen, all of the calculated errors demonstrate relatively high precision. Besides, it can be observed that, although the STLF model does demonstrate the best performance, which results in MASE of 0.36202, the averaged solution is not significantly worse with MASE of only 0.3977, which is the second best result for Product A for the given week. Finally, it should be noticed that all of the models tend to over-forecast the sales i.e. the produced predictions have larger values than the test data, at least when considering the given week. The reasons for which all of the models made the over-forecasting mistakes should be analyzed as it can bring light to an important aspect that can potentially improve the solutions.

Similarly, Appendix 2 shows the forecasts for the same models and with the same cut-off date. Product C, whose data was used in the forecasts, proved to be somewhat more challenging to forecast for most of the models, as it can be seen in the graphs. As in the case of Product A, the forecast produced by averaging the models was the second best with MASE being equal to 0.486 – nearly 0.1 increase compared to the previously discussed product. The exponential smoothing model happened to be the best in the given case with MASE of 0.4547 and RMSE of 0.4072. When observing the graphs produced, it can be noticed that the first two days of the forecasted period contributed the most to the error values. This should also be analyzed further.

Finally, Appendix 3 demonstrates another set of forecasts produced by the models. Compared to the previous two forecast sets, this is the least precise one. Another difference in the forecast is that its cut-off date was chosen to be 06.04.2018. This was done to adequately demonstrate a case when produced forecasts have lower precision but are still useful for future demand estimations. Unlike the previous two examples, the averaged model did not show promising results in this case with MASE being as high as 0.5939, which makes the averaged solution the third worst. The best result was produced

by the exponential smoothing model with MASE being equal to 0.4978 and RMSE to 0.3899. While this is comparable to the best result of the Product C forecasts, the other models produced less accurate predictions, which can be seen in Appendix 2. Besides, similar to the case of Product A, all of the forecasts tend to produce values larger than the required ones for the given time period.

Nevertheless, the results discussed in the previous three paragraphs cannot be reliable for decision-making since the models can demonstrate a significant difference in their performance when applied to the data of a different time period. For this reason, the benchmark that was already introduced at the beginning of the section was executed. The results of the benchmark are provided in Appendix 4. The best performing model based on MASE is bolded for each of the products, and the smallest errors of each type are highlighted as well. In addition, it should be mentioned that SMAPE is missing for some of the models. Such cases indicate that one or more SMAPE calculations produced an undefined value. The reasons for SMAPE resulting in an undefined value were explained in Section 4.3.

As can be seen from Appendix 4, in the cases of Product B, Product C, Product D and Product G, the averaging solution outperformed all of the individual models involved, when using MASE as a criterion. Moreover, the results for Product A and Product F show that the averaged solutions are ranked as second best with only the Prophet model performing better in the case of Product A and the TBATS model being more accurate in the case of Product F. Finally, in all the rest of the tables, namely those of Product R, Product H and Product I, the averaged solution resulted in MASE that is at least smaller than that of the worst performing model. Thus, the results described in the paragraph support the previously outlined theory that averaging forecasting models perform at least as good as the least precise model in the ensemble. Moreover, the theory that, in some cases, simple averaging may outperform each of the models involved was supported by the results as well.

Another aspect that should be considered is the fact that two of the results, namely that of Product E and Product I, didn’t show sufficiently accurate results with MASE being greater than one for the forecasts of all the models. In addition to this, the benchmark results of Product F show significant errors as well with MASE being greater than 0.9 for all of the forecasting solutions. These cases should be analyzed more carefully since

they can obstruct the successful operation of the planning system. On the other hand, the results of product D demonstrate exceptionally low error values with MASE and RMSE being equal to 0.3151 and 0.2688 respectively for the averaged solution. However, it is interesting to note that the SMAPE values are extremely high in this case. Although such contradictory results require further investigation, one possible explanation is the disadvantageous characteristics of SMAPE that were described in Section 4.3.

Furthermore, three of the products were best forecasted by the ARIMA model, while TBATS was the best forecasting solution for other 2 products. The fact that 1D-convolutional neural network did not occur to be the best model for any of the products might be due to two reasons. Firstly, it might show that the more traditional forecasting methods tend to outperform deep learning solutions, in particular, 1D-convolution in the given case. However, a more probable reason for the results is that the neural network model was not sufficiently researched and experimented with, which therefore resulted in the implementation with suboptimal hyperparameters. Suggestions for improvements related to this and other issues are discussed further in Section 6.2.

## 6.2. Future improvements

This subsection describes some of the improvements that can be implemented in order to increase the accuracy of the forecasting solutions. The following five paragraphs propose five different groups of improvements.

First of all, the amount of training data should be increased. As mentioned previously, at the time the described benchmark was executed, the available historical data spanned 1 year and 7 months (October 2016 - June 2018). While this allows for relatively successful forecasting for most of the products, increasing the amount of historical data would certainly be beneficial. Since the access to the data previous to October 2016 is required to significantly increase the amount of historical data, the client company was questioned about possibilities of obtaining the earlier sales and campaign data. As specified earlier, the data used in the benchmark was retrieved from the ERP system that is currently in use by the client company. Since the ERP system was in use only since 2016, receiving the earlier historical data requires accessing the legacy system. However, according to the client company's managing personnel, the sales patterns have

changed significantly compared to the ones used before the year 2016. Therefore, it is argued that the data will not improve the forecasting precision of the models. Nevertheless, it is a viable option to try since some products' forecasts might benefit from having the additional training data.

Secondly, as already was suggested previously, handling of outliers should be improved. During the literature review done as a part of the final year project, little time was devoted to outlier handling. Therefore, more knowledge should be acquired in this direction, and decisions should be made related to the following questions.

- What algorithm is the most appropriate in the given case to be used for identifying what data points are considered as outliers?
- What technique should be used to replace the data points identified as outliers?
- Should the outliers be replaced at all, or instead remain in the data intact?

Besides, the reasons that caused the outliers should be analyzed since they can potentially provide useful insights and help in the future development of the models.

Thirdly, the parameters of the models should differ depending on a product. Although the products do demonstrate similar behaviours in terms of demand, there are significant differences in demand patterns of some of the products. Therefore, the forecasting models' precision can be increased by adjusting their hyperparameters to suit each of the products individually. For example, the reason that the 1D-convolutional neural network model showed somewhat mediocre results can be due to the fact that some of its important parameters, such as the number of training epochs, the filter length, the number of filters per layer, etc., should depend on what product's data is used in the training/validation cycles. Similarly, other models might benefit from the per-product customization.

In addition to this, improvements can be made to the process of combining the models. As discussed in Section 2.2.4, the trimmed mean, when two models that over-forecast and under-forecast the most are excluded from the averaged solution, can be beneficial since more than 5 models are involved in the current solution. Moreover, the topic of forecast model combination should be studied further, and other more sophisticated techniques researched.

Finally, further research can be done to find other model types that could suit the given case. Having additional forecasting models of different types could bring improvements in performance when used on their own as well as when they are added to the averaging solution. Furthermore, forecasting models of the same type but with significantly different parameters could be added to the averaging solution as well.

# 7 Conclusion

The objective of the thesis was to investigate possible solutions that could be used for demand forecasting in the context of the production planning system. Besides, it was required to research whether a combination of multiple forecasting solutions was a viable alternative to using a single prediction model. The forecasting techniques that could be applied in the given case were studied and outlined in Section 2. Based on the theory, the forecasting models were implemented and benchmarked to test how efficient each of the proposed solutions was. Besides, the ensemble of the models was implemented and benchmarked. The results demonstrate that the averaged solution exhibits at least as high performance as the worst model in the ensemble. Moreover, in the four cases out of the nine examined, the averaged solution outperformed every model used in the ensemble. Finally, the TBATS and ARIMA models showed high-grade results relative to other forecasting methods. Therefore, the objectives of the thesis were successfully accomplished.

Following the research described in the thesis, the proposed solution will be integrated into the alpha-version of the production planning software and tested in the real context with a subset of the manufactured products. In parallel to the test deployment, the suggested improvements will be researched and possibly added to the solution. Since the customization of the system to the client's needs was one of the most crucial aspects of the initial specification, it is important to deploy the alpha-version of the solution as soon as possible to enable the client company to provide feedback already on the early stages of the software development.

All in all, the demand forecasting solution is expected to be an essential part of the production planning system. In turn, the system will save hundreds of human hours, reduce amounts of discarded products and make the production process more efficient. This will

provide a competitive advantage to the client company enabling it to increase its profitability and market share, which is argued to be beneficial to both the employees of the company as well as to its customers.

## References

1. Fargher, Hugh E., and Richard A. Smith. Method and system for production planning. U.S. Patent No. 5,586,021. 17 Dec. 1996. URL: https://patents.google.com/patent/US5586021A/en [Accessed 12th September 2018]

2. Martin Kenneth Starr. Production and Operations Management 2008

3. J E Beasley. Master production schedule. URL: http://people.brunel.ac.uk/~mastjjb/jeb/or/masprod.html [Accessed 12th September 2018]

4. Mariana Pedros Casal Ribeiro de Carvalho. The Optimization of Production Planning and Scheduling: A Real Case Study in Ice-cream Industry. 2015. URL: https://fenix.tecnico.ulisboa.pt/downloadFile/281870113702787/Artigo_MarianaCarvalho(65851).pdf [Accessed 30th September 2018]

5. Carlos A. Méndez, Jaime Cerdá, Ignacio E. Grossmann, Iiro Harjunkoski, Marco-Fahl. State-of-the-art review of optimization methods for short-term scheduling of batch processes. 2005. URL: http://egon.cheme.cmu.edu/Papers/BatchReview-MendezGrossmann.pdf [Accessed: 30th September 2018]

6. Muge Erdirik-Dogan, Ignacio E. Grossmann. Simultaneous Planning and Scheduling of Single-Stage Multiproduct Continuous Plants with Parallel Lines. 2007. URL: http://egon.cheme.cmu.edu/Papers/4627_erdirik-grossmann.pdf [Accessed 12th October 2018]

7. Kalyanmoy Deb, Koushik Pal. Efficiently Solving: A Large-Scale Integer Linear Program Using a Customized Genetic Algorithm. Lecture Notes in Computer Science 3102:1054-1065. June 2004. URL: https://www.researchgate.net/publication/220742933_Efficiently_Solving_A_Large-Scale_Integer_Linear_Program_Using_a_Customized_Genetic_Algorithm [Accessed 5th October 2018]

8. Melanie Mitchell. An Introduction to Genetic Algorithms. 1998. URL: https://dl.acm.org/citation.cfm?id=522098

9. Rob J Hyndman and George Athanasopoulos. Forecasting: Principles and Practice [online] last updated April 2018. URL: https://otexts.org/fpp2/ [Accessed 29th August 2018]

10. G. Mahalakshmi, S. Sridevi, S. Rajaram. A survey on forecasting of time series data. 2016. URL: https://ieeexplore.ieee.org/document/7725358 [Accessed 13th September 2018]

11. Mirjana Ivanović, Vladimir Kurbalija. Time series analysis and possible applications. 2016. URL: https://ieeexplore.ieee.org/document/7522190 [Accessed 5th October 2018]

12. R. B. Cleveland, W. S. Cleveland, J. E. McRae, I. J. Terpenning. STL: A seasonal-trend decomposition procedure based on loess. 1990. Journal of Official Statistics, 6(1), 3–73

13. Alysha M. De Livera, Rob J. Hyndman, Ralph D. Snyder. Forecasting time series with complex seasonal patterns using exponential smoothing. 2011. Journal of the American Statistical Association, 106(496), 1513-1527. URL: https://robjhyndman.com/papers/ComplexSeasonality.pdf [Accessed 3rd September 2018]

14. Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning. MIT Press. 2016. URL: http://www.deeplearningbook.org [Accessed 26th September 2018]

15. Anastasia Borovykh, Sander Bohte, Cornelis W. Oosterlee. Conditional time series forecasting with convolutional neural networks. 2018. URL: https://arxiv.org/abs/1703.04691 [Accessed 2nd October 2018]

16. Fisher Yu, Vladlen Koltun. Multi-Scale Context Aggregation by Dilated Convolutions. 2016. URL: https://arxiv.org/abs/1511.07122 [Accessed 11th October 2018]

17. Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. 2016. URL: https://arxiv.org/pdf/1609.03499.pdf [Accessed 2nd September 2018]

18. Sima Siami-Namini, Akbar Siami Namin. Forecasting Economics and Financial Time Series: ARIMA vs. LSTM. 2018. URL: https://arxiv.org/abs/1803.06386 [Accessed 10th October 2018]

19. Spyros Makridakis, Evangelos Spiliotis, Vassilios Assimakopoulos. Statistical and Machine Learning forecasting methods: Concerns and ways forward. 2018. PLOS One. URL: https://journals.plos.org/plosone/article/file?type=printable&id=10.1371/journal.pone.0194889 [Accessed 12th October 2018]

20. Robert T. Clemen. Combining forecasts: A review and annotated bibliography 1989. URL: https://www.sciencedirect.com/science/article/pii/0169207089900125 [Accessed 6th October 2018]

21. J. Scott Armstrong. Combining Forecasts. 2001. URL: http://forecastingprinciples.com/paperpdf/Combining.pdf [Accessed 11th October 2018]

22. Kevin K.F. Wong, Haiyan Song, Stephen F. Witt, Doris C. Wu. Tourism forecasting: To combine or not to combine? 2007. URL: https://www.sciencedirect.com/science/article/pii/S0261517706001415 [Accessed 26th September 2018]

23. David F. Hendry, Michael P. Clements. Pooling of forecasts. 2004. URL: http://www.nuff.ox.ac.uk/economics/papers/2002/w9/dfhmpcfrcncectj.pdf [Accessed 13th October 2018]

24. Hui Zou, Yuhong Yang. Combining time series models for forecasting. 2004. URL: http://users.stat.umn.edu/~zouxx019/Papers/after.pdf [Accessed 26th August 2018]

25. Python Software Foundation. Python Language Reference, version 3.6. URL: http://www.python.org [Accessed 14th October 2018]

26. R Core Team (2013). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/ [Accessed 14th October 2018]

27. Sean J. Taylor, Ben Letham. Prophet: forecasting at scale. Facebook research blog. Blog. 2017. URL: https://research.fb.com/prophet-forecasting-at-scale/ [Accessed 14th October 2018]

28. Sean J. Taylor, Benjamin Letham. Forecasting at Scale. 2017. PeerJ Preprints. URL: https://peerj.com/preprints/3190.pdf [Accessed 13th October 2018]

29. Ben Letham, Anshuman Chhabra. Prophet documentation. 2018. URL: https://facebook.github.io/prophet/docs [Accessed 14th October 2018]

30. Keras documentation contributors. Keras documentation. 2018. URL: https://keras.io/ [Accessed 14th October 2018]

31. PyData. Pandas website. 2018. URL: https://pandas.pydata.org/ [Accessed 14th October 2018]

32. Pipenv documentation contributors. Pipenv documentation. 2018. URL: https://pipenv.readthedocs.io/en/latest/ [Accessed 14th October 2018]

33. Seabold, Skipper and Perktold, Josef. Statsmodels: Econometric and statistical modeling with python. 2010. 9th Python in Science Conference. URL: http://conference.scipy.org/proceedings/scipy2010/pdfs/seabold.pdf [Accessed 14th October 2018]

34. Time Series analysis tsa. Statsmodels documentation. Time series analysis section. URL: https://www.statsmodels.org/stable/tsa.html [Accessed 14th October 2018]

35. Statsmodels documentation. URL: https://www.statsmodels.org/dev/generated/statsmodels.tsa.holtwinters.ExponentialSmoothing.fit [Accessed 14th October]

36. Hyndman R, Athanasopoulos G, Bergmeir C, Caceres G, Chhay L, O'Hara-Wild M, Petropoulos F, Razbash S, Wang E, Yasmeen F. forecast: Forecasting functions for time series and linear models. R package version 8.4. 2018. URL: http://pkg.robjhyndman.com/forecast. [Accessed 14th October 2018]

37. RPy2 documentation contributors. RPy2 documentation. 2017 URL: https://rpy2.readthedocs.io/en/version_2.8.x/overview.html [Accessed 14th October 2018]

38. Taylor G. Smith. Pyramid documentation. 2018. URL: https://www.alkaline-ml.com/pyramid/about.html [Accessed 21st October 2018]

39. SQuirreL SQL Client. URL: http://squirrel-sql.sourceforge.net [Accessed 14th October 2018]

40. Rob J. Hyndman, Anne B. Koehler. Another look at measures of forecast accuracy. 2006. International Journal of Forecasting, 22, 679–688. URL: https://robjhyndman.com/publications/automatic-forecasting/ [Accessed 14th October 2018]

41. Kaggle. Store Item Demand Forecasting Challenge. Evaluation. URL: https://www.kaggle.com/c/demand-forecasting-kernels-only#evaluation [Accessed 14th October 2018]

42. Kaggle. Web Traffic Time Series Forecasting. Evaluation. URL: https://www.kaggle.com/c/web-traffic-time-series-forecasting#evaluation [Accessed 14th October 2018]

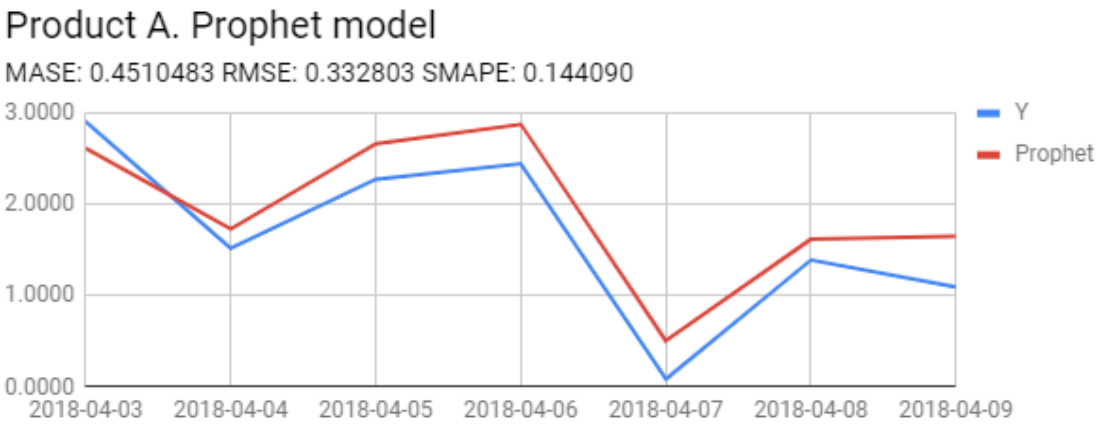
43. Kaggle. Restaurant Revenue Prediction. Evaluation. URL: https://www.kaggle.com/c/restaurant-revenue-prediction#evaluation [Accessed 14th October 2018]

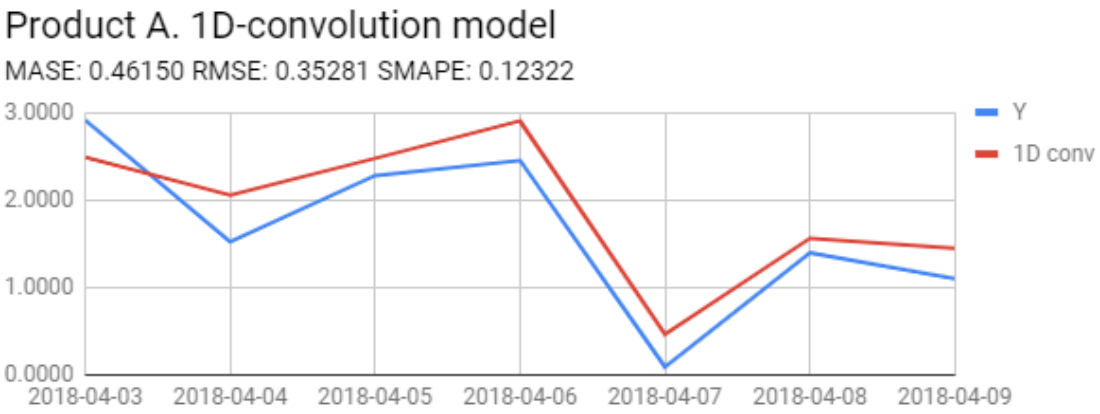
44. Kaggle. Predict Future Sales. Evaluation. URL: https://www.kaggle.com/c/competitive-data-science-predict-future-sales#evaluation [Accessed 14th October 2018]

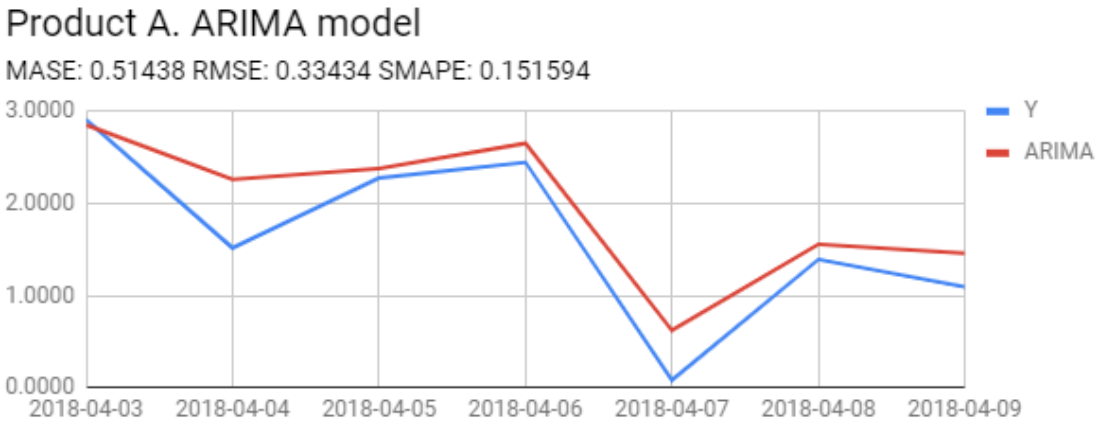
45. Kaggle. Avito Demand Prediction Challenge. Evaluation. URL: https://www.kaggle.com/c/avito-demand-prediction#evaluation [Accessed 14th October 2018]

46. Benjamin Bannier. Stack Overflow. 2014. URL: https://stackoverflow.com/a/16562028/10468398 [Accessed 14th October 2018]

## Forecasts for Product A

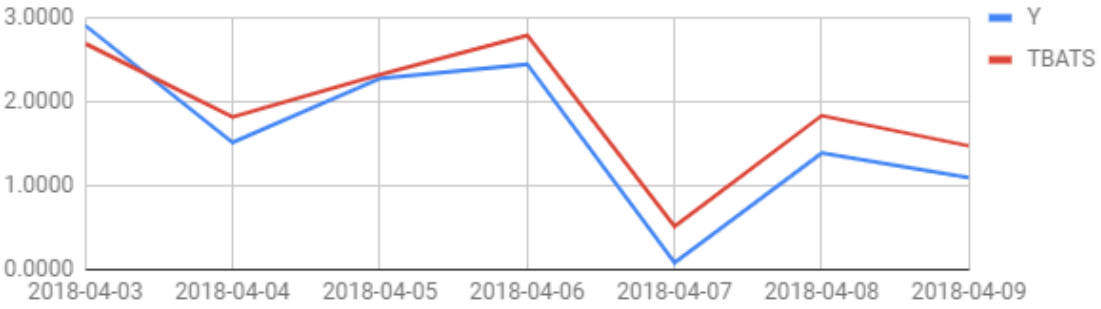


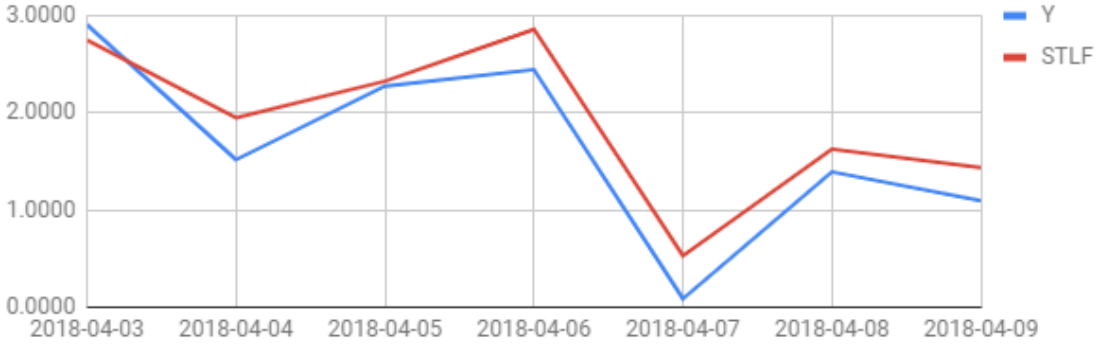
## Product A. TBATS model

MASE: 0.41424 RMSE: 0.26868 SMAPE: 0.12895



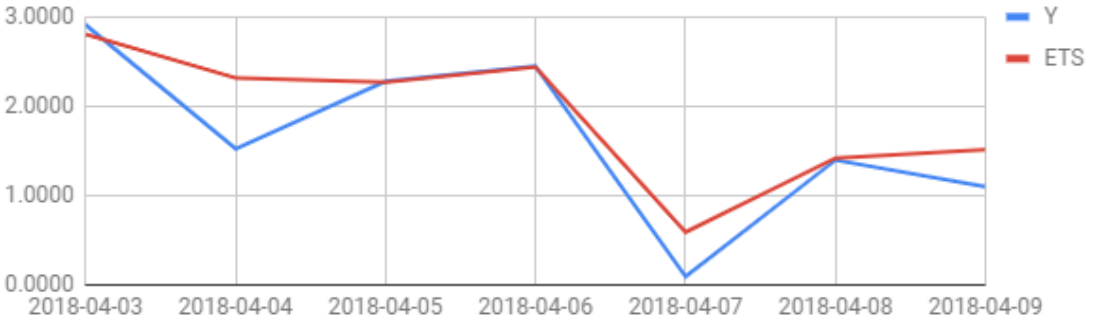

## Product A. STLF model

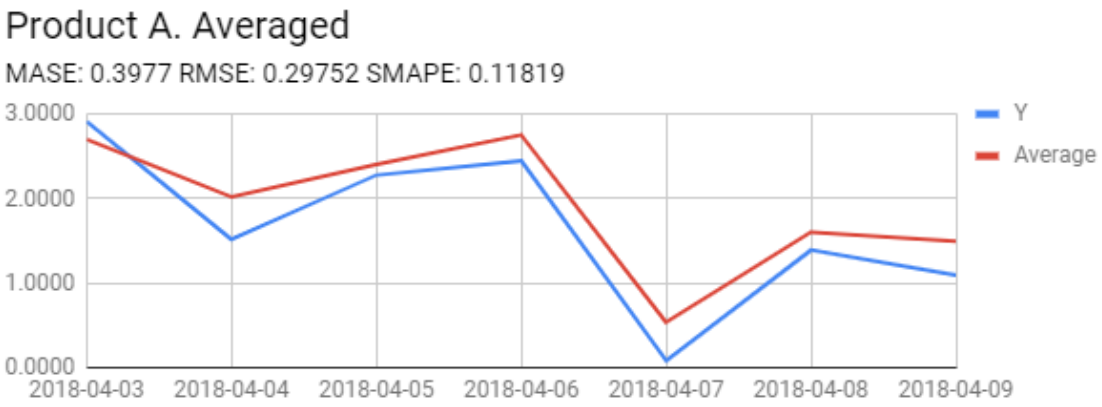
MASE: 0.36202 RMSE: 0.26353 SMAPE: 0.10048



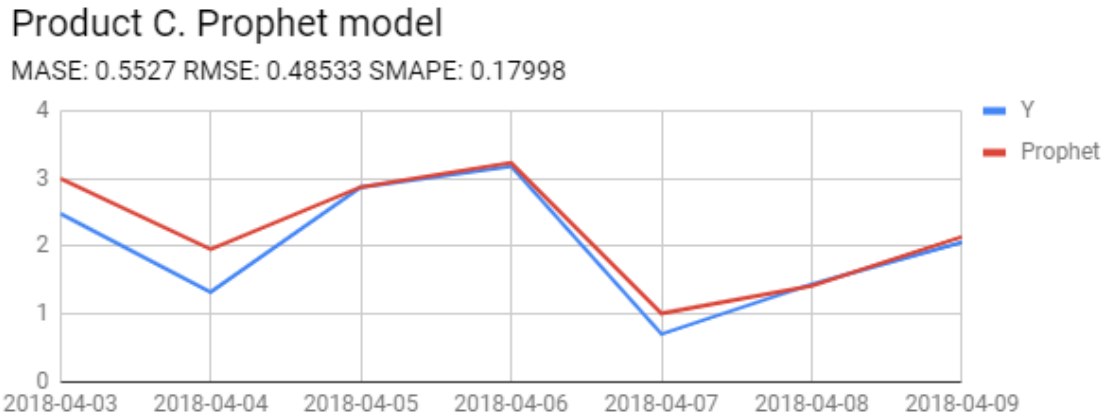

## Product A. Exponential smoothing model

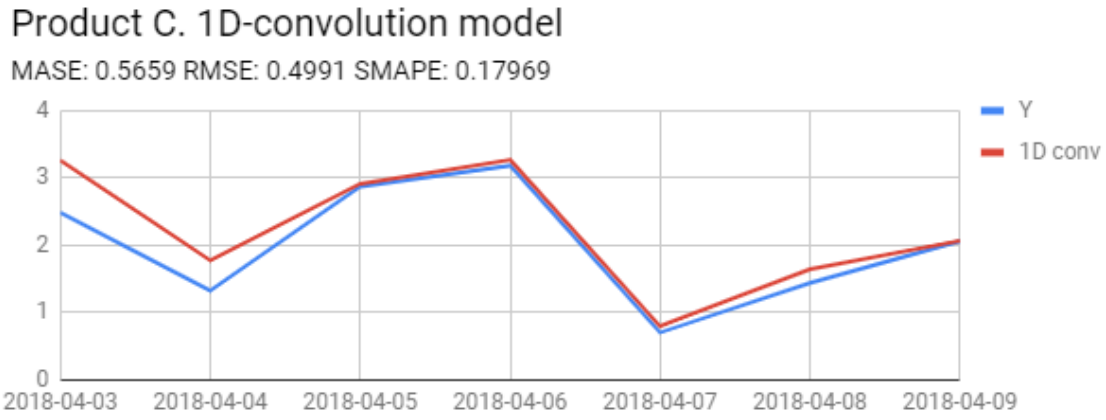
MASE: 0.46855 RMSE: 0.33056 SMAPE: 0.124942

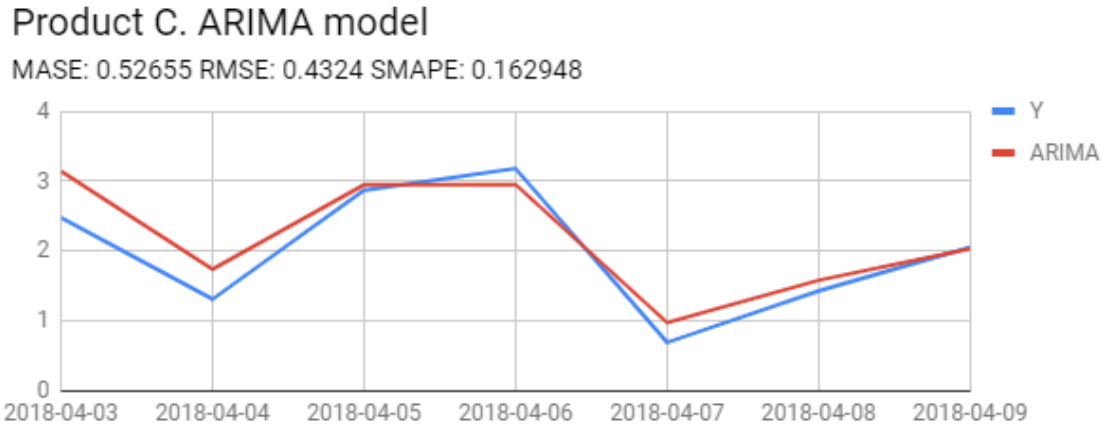



Product A. Averaged
MASE: 0.3977 RMSE: 0.29752 SMAPE: 0.11819
3.0000
2.0000
1.0000
0.0000
2018-04-03
2018-04-04
2018-04-05
2018-04-06
2018-04-07
2018-04-08
2018-04-09
Y
Average

## Forecasts for Product C

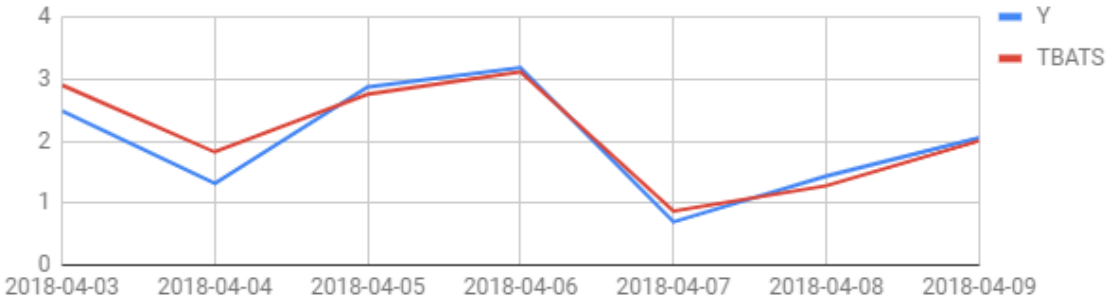


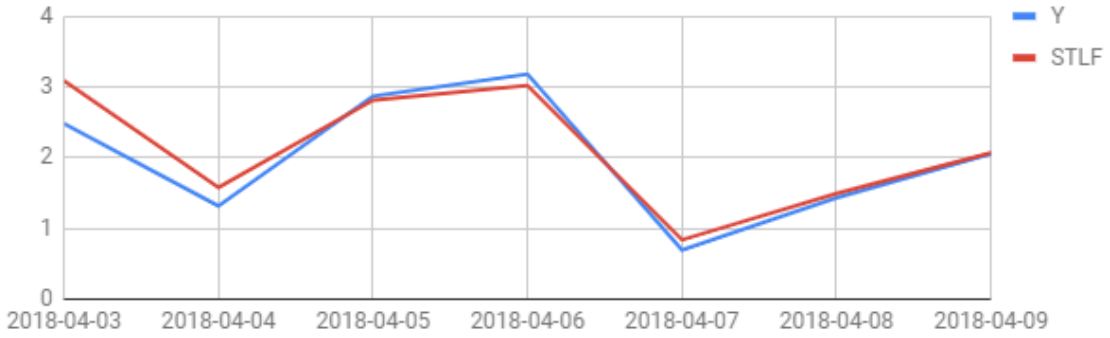
## Product C. TBATS model

MASE: 0.7065 RMSE: 0.59689 SMAPE: 0.20312



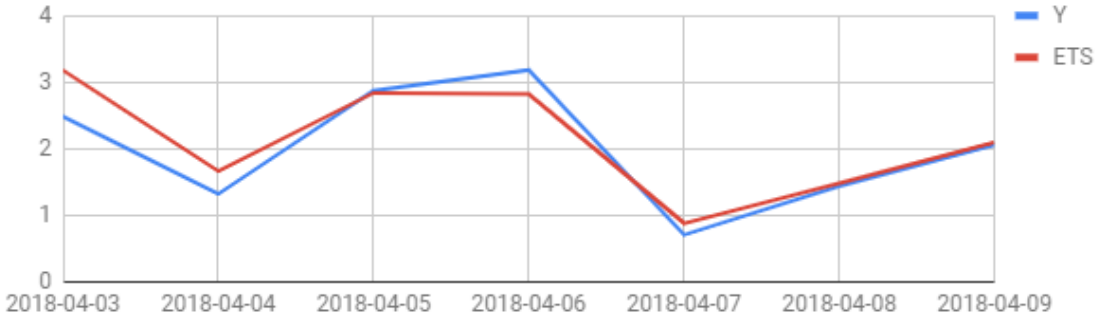

## Product C. STLF model

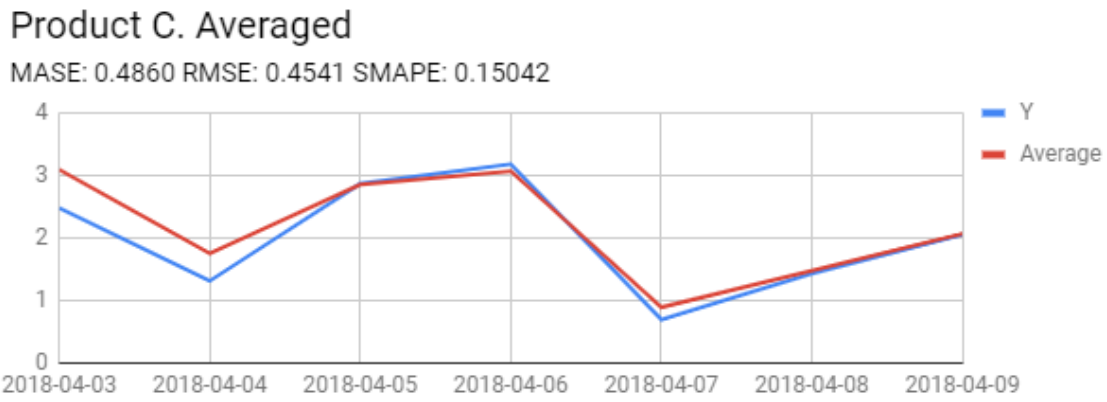
MASE: 0.6052 RMSE: 0.50611 SMAPE: 0.1916



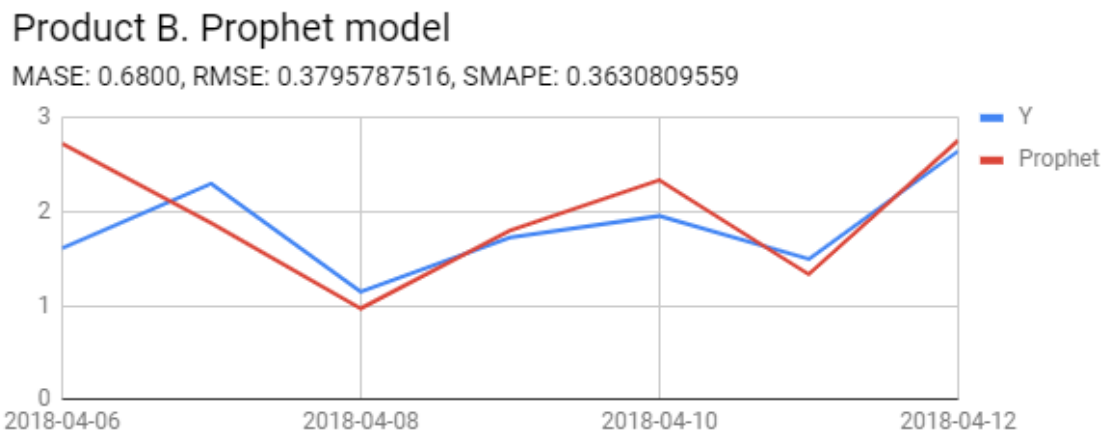

## Product C. Exponential smoothing model

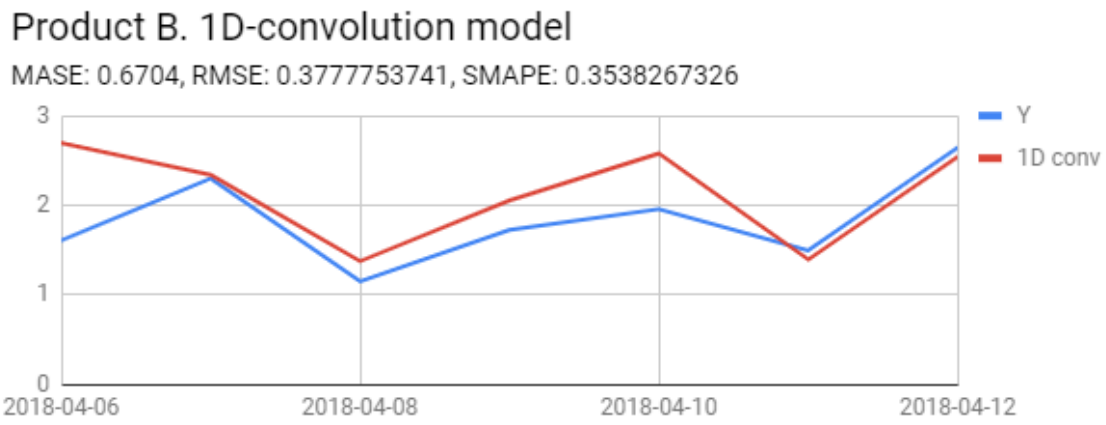
MASE: 0.4547 RMSE: 0.4072 SMAPE: 0.14371

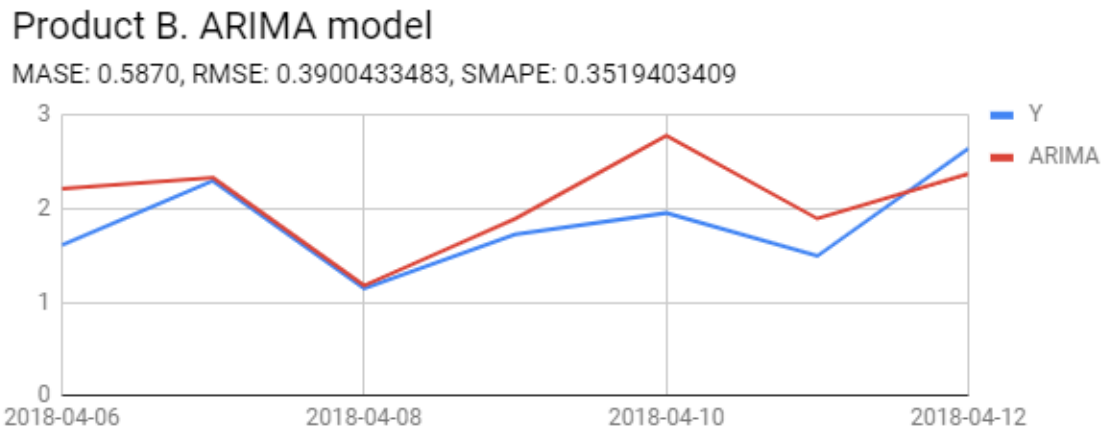



Product C. Averaged
MASE: 0.4860 RMSE: 0.4541 SMAPE: 0.15042
4
3
2
1
0
2018-04-03
2018-04-04
2018-04-05
2018-04-06
2018-04-07
2018-04-08
2018-04-09
Y
Average

**Forecasts for Product B**

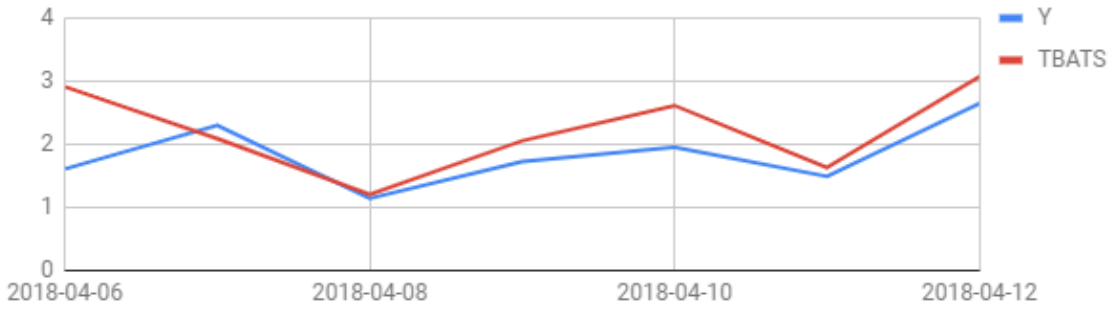


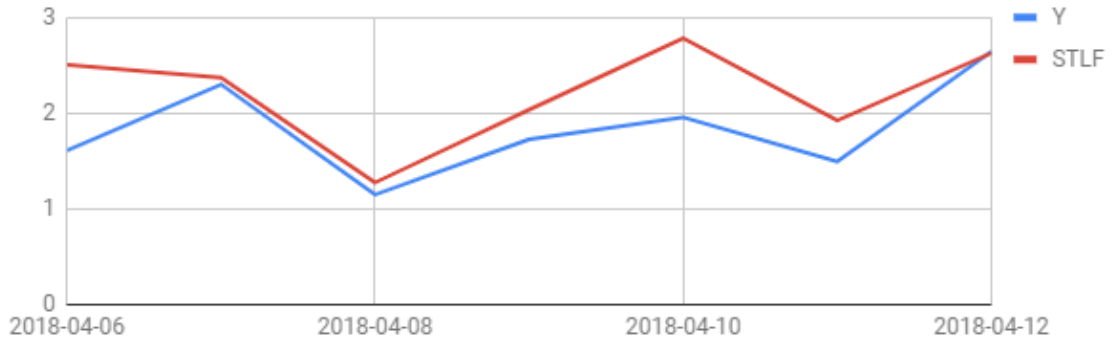
Product B. TBATS model

MASE: 0.5819, RMSE: 0.335552063, SMAPE: 0.3456601594



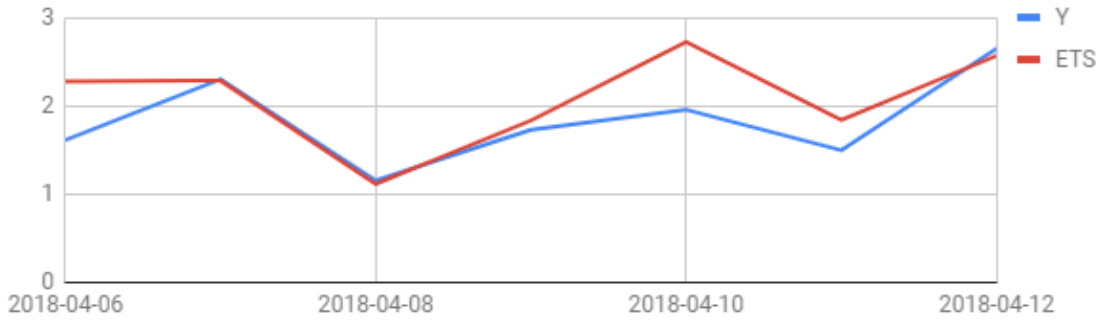

Product B. STLF model

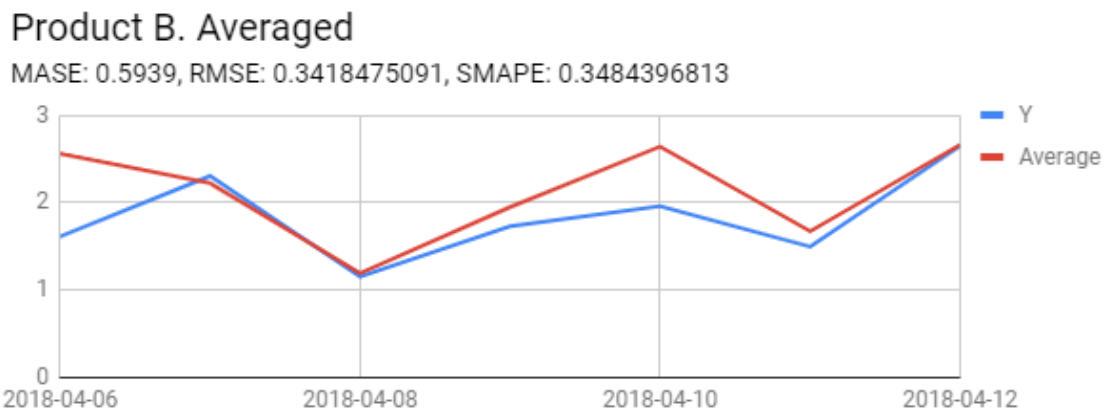
MASE: 0.5563, RMSE: 0.3281372728, SMAPE: 0.3366980602




Produt B. Exponential smoothing model

MASE: 0.4978, RMSE: 0.3899454504, SMAPE: 0.3270253634




Product B. Averaged
MASE: 0.5939, RMSE: 0.3418475091, SMAPE: 0.3484396813
3
2
1
0
2018-04-06
2018-04-08
2018-04-10
2018-04-12
Y
Average

## Benchmark results

| **Product A** | MASE | RMSE | SMAPE |
|---|---|---|---|
| **Prophet** | **0.6542218273** | **0.4260477568** | 0.2733417523 |
| 1D convolution | 0.7004403477 | 0.4653857568 | 0.2788098159 |
| ARIMA | 0.67631965 | 0.4704169591 | 0.2826369682 |
| TBATS | 0.7104953114 | 0.4655035159 | 0.2848271 |
| STLF | 0.6963246795 | 0.4788156886 | 0.2833850114 |
| Exponential smoothing | 0.6702669227 | 0.4742371182 | 0.2754486886 |
| Average | 0.6635883909 | 0.4500770682 | **0.2722653364** |

| **Product B** | MASE | RMSE | SMAPE |
|---|---|---|---|
| Prophet | 0.6036514705 | 0.4821738818 | 0.1910299045 |
| 1D convolution | 0.5836166432 | 0.4792911 | 0.1833612114 |
| ARIMA | 0.6200273545 | 0.4840787841 | 0.1906533136 |
| TBATS | 0.6004788295 | 0.486314425 | 0.1844558545 |
| STLF | 0.6371208727 | 0.4945507114 | 0.1969914705 |
| Exponential smoothing | 0.6470071068 | 0.5043705955 | 0.1988225545 |
| **Average** | **0.5597954614** | **0.4503159773** | **0.1751829136** |

| **Product C** | MASE | RMSE | SMAPE |
|---|---|---|---|
| Prophet | 0.6622596955 | 0.4616685591 | 0.191085875 |
| 1D convolution | 0.7003889114 | 0.4852601091 | 0.1922473591 |
| ARIMA | 0.667633875 | 0.4635831386 | 0.1862433045 |
| TBATS | 0.6632180614 | 0.4585986841 | 0.1867430523 |
| STLF | 0.6375331614 | 0.451560025 | **0.1749866409** |
| Exponential smoothing | 0.6663720182 | 0.4573174659 | 0.1818089068 |
| **Average** | **0.6326286682** | **0.4492324909** | 0.1756769591 |

| **Product D** | MASE | RMSE | SMAPE |
|---|---|---|---|
| Prophet | 0.3634174068 | 0.29163435 | 0.8425854676 |
| 1D convolution | 0.3549949386 | 0.2835225318 | 0.6421064946 |
| ARIMA | 0.3597991205 | 0.3016072705 | 0.8562578541 |
| TBATS | 0.3988602068 | 0.3260807909 | 0.8594660703 |
| STLF | 0.3423043864 | 0.2964972318 | **0.503035925** |
| Exponential smoothing | 0.3407955341 | 0.2929003273 | 0.6142696364 |
| **Average** | **0.31513855** | **0.2687538114** | 0.5052656977 |

| **Product E** | MASE | RMSE | SMAPE |
|---|---|---|---|
| Prophet | 1.445235284 | **0.6483075955** | |
| 1D convolution | 1.352521273 | 0.6632207159 | |
| ARIMA | **1.33525972** | 0.6840316909 | |
| TBATS | 1.471706818 | 0.6532086841 | |
| STLF | 1.348076048 | 0.6932001091 | |
| Exponential smoothing | 1.342383814 | 0.6920497091 | |
| Average | 1.361788395 | 0.6524288205 | |

| **Product F** | MASE | RMSE | SMAPE |
|---|---|---|---|
| Prophet | 0.9712771568 | 0.6896002795 | 0.662743916 |
| 1D convolution | 1.039631284 | 0.7139918455 | 0.757952875 |
| ARIMA | 0.9401765045 | 0.6833682045 | 0.6556537364 |
| **TBATS** | **0.9148826545** | **0.6723071364** | 0.5760731465 |
| STLF | 0.979229225 | 0.7109044705 | 0.5979842386 |
| Exponential smoothing | 0.9332848932 | 0.6831480659 | **0.505010675** |
| Average | 0.9330956159 | 0.6791622341 | 0.5839680091 |

| **Product G** | MASE | RMSE | SMAPE |
|---|---|---|---|
| Prophet | 0.8967182136 | 0.6882920227 | **0.3813184386** |
| 1D convolution | 0.9085901318 | 0.6872097795 | 0.4342572318 |
| **ARIMA** | **0.8941516545** | **0.6830228591** | 0.3958995205 |
| TBATS | 0.8948834295 | 0.6804583886 | 0.3913324114 |
| STLF | 0.9542992068 | 0.7349185523 | 0.3965698409 |
| Exponential smoothing | 0.9011430136 | 0.6985685568 | 0.3888545386 |
| Average | 0.8884328591 | 0.6871133841 | 0.3832674227 |

| **Product H** | MASE | RMSE | SMAPE |
|---|---|---|---|
| Prophet | 0.630555541 | 0.737407016 | |
| 1D convolution | 0.708693659 | 0.802099520 | |
| **ARIMA** | **0.613717475** | **0.713896395** | |
| TBATS | 0.649200807 | 0.720968155 | |
| STLF | 0.661618689 | 0.744325761 | |
| Exponential smoothing | 0.621917934 | 0.725924339 | |
| Average | 0.635580693 | 0.716317602 | |

| **Product I** | MASE | RMSE | SMAPE |
|---|---|---|---|
| Prophet | 1.234075523 | 0.977577943 | |
| 1D convolution | 1.631033000 | 1.280780018 | |
| ARIMA | 1.215858393 | 0.959142107 | |
| **TBATS** | **1.193653245** | **0.948836843** | |
| STLF | 1.255058248 | 1.002478041 | |
| Exponential smoothing | 1.199753220 | 0.953790775 | |
| Average | 1.260928480 | 0.978447959 | |