Accounts Receivable Cash Flow Forecasting:

A comparison between

SES, ARIMA, LSTM and Hybrid ARIMA-LSTM

Master Thesis 2022

Author: Mathijs Lenderink
Supervisor: dr. Gertjan Verhoeven
Second reader: prof. dr. Jan Boone
Date: 3-6-2022
ANR: 337898
SNR: 2021146
Program: M.Sc. Thesis for Economics
Words: 15,160

Abstract

Forecasting cash flow is important to ensure the health of a company. This paper will research to what extent machine learning techniques can predict the cash flow of the accounts receivable based on historical data at a company of which the accounts receivable consists of few large value entries and many small value entries. In this study I compare the performance of SES, ARIMA, LSTM and Hybrid ARIMA-LSTM models on one company dataset and three synthetic datasets. The models forecast a four-week window and the model performance is analyzed using the root mean squared error and mean absolute error. The results of the synthetic datasets showed that the hybrid model gave the most accurate prediction. The results of the company dataset show that for this company there was not one best model. Further research is needed to conclude whether this is also the case for similar other companies.

Keywords: Time series forecasting, cash flow, Accounts receivable (AR), Simple Exponential Smoothing (SES), Autoregressive Integrated Moving Average (ARIMA), Long Short-Term Memory neural network (LSTM), Hybrid ARIMA-LSTM, Synthetic data, Auto_arima, Keras Tuner, Root mean squared error (RMSE), Mean absolute error (MAE)

Table of Contents

1. Introduction

Cash flow is often stated as one of the most important financial indicators for the health of companies. When cash flow is managed well a company has better insight in their opportunities. It can for example give insight in the analysis whether projects can be taken on or whether they are too expensive at that time. It also shows whether a company is able to hire more employees, invest more, expand the company and is able to withstand demand shocks (Dadteev, Shchukin, & Nemeshaev, 2020). A company with well managed cash flows is overall better in staying solvent, giving a company more choices what to do with their cash, whether to invest it into projects, invest it into the stock market or use the extra cash to settle debts. Especially in uncertain times when the market and demand is volatile well managed cash flow gives a company more security. In essence is cash the main bloodline of a company, a company with well managed cash levels is often considered a healthy company, while a company without cash is most of the times at risk of bankruptcy (Tangsucheeva & Prabhu, 2014). Cash management in a large multinational company is especially important due to the many responsibilities it has. A well forecasted cash flow can help with smoothing banking relationships, easier funding, more transparency, interest rate risk management, currency risk management and tax insights, which are all very important for large multinational companies. It is therefore important for companies to manage their cash flow well, in order to be seen as a healthy company.

Current cash flow forecasting methods can be categorized into multiple categories. First, the classical time series method, such as ETS, ARIMA, Theta, Naïve 2 and Simple Exponential Smoothing. Second, the neural network approach, is the method of using artificial neural networks which can find patterns and can forecast cash flow without needing an expert. It is also common to combine methods from the first category with methods from the second category to create hybrid models, an example of this is the hybrid model consisting of ARIMA combined with a neural network model. The third, factor

analysis, is a method that takes into account the factors that affect cash flow and calculates the correlation of those factors with cash flow. This method will be left out of the analysis as it does not use machine learning. The fourth method, expert systems, are systems which are tailored exactly to a company to imitate the reasoning of a human operator. This system cannot be generalized and only works when tailored to a specific company, due to this it will also be left out of the analysis.

Currently there are few papers that specifically research cash flow prediction using machine learning. The papers that exist often only look at a specific part of cash flow or a specific model. However as cash flow consists of financial time series data there are multiple papers which look at the models I will be using in my comparison, but these papers look at other financial time series data. To the best of my knowledge there are no papers that research cash flow forecasting using machine learning at companies like the company in this case. That is because the company in this case has a cash flow consisting of many small transactions and few large transactions. This structure of cash flow is due to the products the case company produces and the high value of each product. The number of products produced is small, but the value per product is high.

To shed light on the effectivity of different machine learning methods in cash flow prediction in this class of companies I will compare multiple forecasting machine learning methods. The first model I will apply is the Simple Exponential Smoothing model, this model is used as a baseline model due to the simple calculation it applies. The second model which will be compared is the ARIMA model. The ARIMA model is a forecasting method widely used for forecasting data with linear patterns. The popularity of ARIMA is because ARIMA has many of the statistical properties which are needed for a reliable forecasting method. It can also be used on many different data structures because it has various exponential smoothing models which it can apply. This is because ARIMA is a method containing multiple different models, which makes the ARIMA model very flexible (Zhang, 2003). The third model in this comparison is the Long

Short-Term Memory neural network model. The Long Short-Term Memory neural network model is a type of recurrent neural network which is good in handling time series data (Cao, Li, & Li, 2019). Because of this it is a model which is often used to forecast time series data such as financial data, weather data and other time series data. The final model is a hybrid model consisting of ARIMA and a Long Short-Term Memory neural network model. The combination of these two models makes it such that both the linear and the nonlinear patterns within the data are recognized by the models. First the ARIMA model is used to model the linear component within the data and then the Long Short-Term Memory neural network is used to model the ARIMA residuals, which contain the nonlinear patterns within the data (Zhang, 2003) (Panigrahi & Behera, 2017).

Before using the models to predict the cash flow of the accounts receivable of the company dataset I will first train the model on synthetic datasets. With a known, synthetic, dataset I can see whether the models work properly, such that there are no errors in the data preparation or in the models. By using a dataset with a known distribution it is then also possible to evaluate to what extend a model can learn the distribution (Nikolenko, 2021). In this study I will be using three synthetic datasets. The first synthetic dataset will be a linear dataset consisting of a linear constant linear increase over time. The second dataset will consist of a yearly, 52-week, recurring pattern. The third synthetic dataset will consist of the yearly pattern combined with a monthly, 4-week, pattern.

Using the three synthetic datasets and one company dataset I will compare the forecasting accuracies using the root mean squared error and the mean absolute error. With the results of this comparison I will try to answer the following research question in this study: “To what extent can different machine learning forecasting models forecast the cash flow of the accounts receivable in companies with few transactions with large values and many transactions with large values?”

2. Literature Review

Cash flow forecasting is a topic for which research is valuable for companies. This has as an effect that most of the research that is done is case specific and not publicly available. These studies are done to get a competitive edge on the competition, therefore most of the research is not published in research papers. According to (Weytjens, Lohmann , & Kleinsteuber, 2019) before their research there was to the best of their knowledge no publicly available research on advanced cash flow forecasting.

To create a clear overview of the existing literature I will divide the literature into two categories. The first being the background literature regarding cashflow forecasting in general. This category will be helpful to find important existing examples of methods which are successful in forecasting cashflow and their effectiveness. The second category is the existing literature regarding the machine learning methods which can be used to forecast time series data. At the end of this section there is a summery table summarizing the time series forecasting papers.

2.1 Cash flow forecasting literature

I will now give a broader overview of the history of cashflow forecasting. Originally the treasury department of a company was often called a profit center as they were used to generate profits from the financial resources of a company. Starting in the 1990s there was a shift in treasury business functions. The treasury department within companies became responsible for the cash management, financial exposure, hedging currencies, funding the company and managing inhouse cash. This shift made the treasury department a cost center instead of a profit center, such that it minimized exposure and managed cash in the most efficient way (Roszkowska & Prorokowski, 2017). Over the turn of the century the responsibility of key controls and documenting the financial workflows also became a function for the treasury department of large companies. After the

financial crisis another shift can be observed within treasury functions. The treasury functions shifted to a more performance-based treasury department. This performance was mainly focused on managing the profit and loss of the company. With the current functions of treasury a very important part is managing the cash flows of the company (Roszkowska & Prorokowski, 2017).

The cash flow is important to manage well because to attain efficient results of the treasury department it needs full visibility over the cash flow. In theory this is very simple, but in reality payments will often not be made on the due date. Other factors that make cash flow unpredictable are that payment formats differ per country, companies use multiple different banks and each company works differently. For example there are companies that use payment systems which bundle multiple scheduled payments over the week in one transaction on one day. Another factor that is important is that due to regulations there are countries that trap cash in a country (Polak, Robertson, & Lind, The New Role of the Corporate Treasurer: Emerging Trends in Response to the Financial Crisis, 2011). This then makes it difficult to sweep the cash to one central entity such that there is no exposure in the country of the customer. In the case of China it is such that the cash that is paid as prepayment for a system order cannot leave the country until the product is received by the customer. This makes it such that large cash balances remain in China even though the cash in question is owned by the seller (Shira, 2019).

An important shift in treasury management is the automation of treasury processes. This is a priority for many multinational companies as the globalization, rapid increased competition and rapid business growth create automation opportunities. This shift to automation also focusses heavily on the global cash management and forecasting. With this shift to automation the role of artificial intelligence and machine learning is also becoming more prominent (Polak, Nelischer, Guo, & Robertson, 2020) (Dadteev, Shchukin, & Nemeshaev, 2020).

A core activity of the treasury department is the cash flow forecasting. Often the data related to this is incomplete, inaccurate and unreliable, mostly because cash flow relies on many other parties involved. Just like in this paper often the historical cash flow data is used to predict the future. As this is time series data the model that will forecast the cashflow will need to be a time series model. Cash flow from the accounts receivable consist of customer payments. As customers, in this case other companies, often have payment runs in which they pay all outstanding invoices at once a weekly pattern can exist per customer (Tangsucheeva & Prabhu, 2014).

2.2 Time series forecasting models literature

Time series forecasting is a topic which is well-researched. The analysis I do in this thesis is the comparison of multiple time series forecasting methods, specifically on the cash flow of the accounts receivable in a company. To give a comprehensive view of the existing literature of time series forecasting I have summarized the main points of the leading studies over the past years.

Forecasting time series has been done on a broad range of topics. This is because in many research areas there is time series data available. A few examples of areas where time series forecasting has been applied are cash demand data, exchange rate data, electricity price data, agricultural data, physics data, climatology data and tourism data. The forecasting methods used in these time series forecasting range from the statistical methods, such as ARIMA and exponential smoothing, to the computational methods such as recurrent neural networks. Some studies have concluded that a hybrid form of multiple methods resulted in the best forecasts with their data. The combined methods then make use of the advantages of both the statistical and the computational methods (Parmezan, Souza, & Batista, 2019).

A complicated topic is which model to use in different circumstances. Lemke and Gabrys (2010) introduced a meta-learning, a term to describe the process of

acquiring knowledge on which model is best, approach to select a specific model without having to try multiple different models to compare the results. The models they compare for their analysis are Simple Exponential Smoothing, Taylor smoothing, Theta model, polynomial regression, ARIMA, artificial neural network and Elman neural network. They aimed to investigate whether it is possible to link problem specific knowledge to the various models. They used different features to measure the effectiveness per model on specific problems and mapped the decision of which model is the most accurate using decision trees. They do write that these decision tree guidelines are not applicable to all datasets, but that they can often be successfully exploited to make model selection faster (Lemke & Gabrys, 2010).

Another approach of increasing forecasting accuracy is applied by Andrawis, Atiya and El-Shishiny (2011). The main idea of their approach is to combine multiple different forecasts from different models. Using concepts and methods that were earlier found in the existing literature they combined multiple forecasts from different models to gain the superior performance. The models they used, after testing multiple other models, were Gaussian process regression, artificial neural networks and moving average. By combining the forecasts from these models they found that their model accuracy was superior from the model forecasts of each model individually (Andrawis, Atiya, & El-Shisiny, 2011).

Another paper that combined two methods to achieve the best forecasting is the paper by Khashei and Beijari (2011). They propose a new method to combine artificial neural networks with ARIMA, as these hybrid models can be more accurate than each model on its own. They also propose this method to reduce the risk of using a model that is not suited for specific data. By combining a linear model, ARIMA, with a nonlinear model, an artificial neural network, they try to overcome the limitations of both and create more accurate forecasts. Their hybrid model consists of two stages, where the first is the linear component and the second is the nonlinear component. With the already established Zhang hybrid

methodology there are limiting assumptions which Khashei and Beijari (2011) wanted to overcome. The Zhang method has the following limiting assumptions:

1. It assumes that the patterns of the linear and the nonlinear of a time series can be modeled separately by different models
2. It assumes that the linear and the nonlinear component have a relationship that is additive.
3. It assumes that the linear model residuals only contain the nonlinear relationship.

Khashei and Beijari (2011) succeeded in creating a forecasting model without the above assumptions which is more general and accurate. They also conclude that they can generally guarantee that the results of their model will be more accurate than either of the models on their own.

Babu and Reddy (2014) wrote a paper in which they propose another hybrid model of ARIMA and an artificial neural network. They used a more complicated method than Khashei and Beijari (2011) by taking into account the time series before applying the model. This is done by first using a moving average filter to explore the nature of the volatility of the time series. After this they applied ARIMA and the artificial neural network to get the forecast results. This proposed hybrid model of Babu and Reddy (2014) was more accurate for both one-step ahead and multi-step ahead forecasts over multiple different datasets.

Oliveira and Ludermir (2016) proposed another hybrid method which added a data decompose step at the beginning which decomposes the data based on the low and high volatility patterns of the data. It does this by first using an exponential smoothing filter to decompose the data with respect to the low and high volatility patterns. The high volatility is then decomposed to linear and nonlinear components. This proposed model was more complex than other models, but it does offer promising results. With their test datasets they concluded that overall their proposed method was the best, but for some datasets it was not the most accurate. The accuracy of their proposed model and the hybrid

ARIMA with neural network model in general is dependent on the linear and nonlinear patterns in the data (Oliveira & Ludermir, 2016).

There are also hybrid models that consists of multiple different models. Raza, Nadarajah and Ekanayake (2017) created a forecasting model that consisted of a backpropagation neural network, ARIMA, feed forward neural network, wavelet transform and radial basis function. The combination of models allowed them to apply each model such that the best output is achieved. First, they apply wavelet transform to the time series data such that the spikes and fluctuations of the data are removed. They continue by training the feed forward neural network and the radial basis function to gain a higher forecast accuracy. Using Bayesian model averaging they then average the output of both predictors which produces the output result. This combined model produces more accurate results for both seasonal daily and seasonal weekly data (Raza, Nadarajah, & Ekanayake, 2017).

Beside hybrid models with ARIMA it is also possible to have other hybrid models. Panigrahi and Behera (2017) create a model that consists of ETS and an artificial neural network. They propose this model because they think that this combination gives the best chance of capturing the combination of linear and nonlinear pattern in time series data. Just as with other hybrid models in other literature they first try to capture the linear patterns by applying the ETS model, because an ETS is better in handling linear patterns than an artificial neural network. With this prediction they then calculate the residuals by subtracting the prediction from the actual values. This residual time series is then used as input data for the artificial neural network. The prediction values from both models are then summed to obtain the final prediction values. They tried this method on sixteen different time series data sets and compared the results with the results of ARIMA, ETS and a multilayer perceptron model. They also looked at past hybrid models from other studies. The hybrid model did not give the best results for all the time series data sets, but it did give promising results for most of the datasets as the hybrid model outperformed the individual models (Panigrahi & Behera, 2017).

Table 1: Time series forecasting literature summary table

Reference	Method	Conclusion/Contribution
(Zhang, 2003)	ARIMA, hybrid ARIMA-ANN	Introducing the hybrid model consisting of ARIMA and an artificial neural network, where the artificial neural network is used to predict the error of the ARIMA model.
(Lemke & Gabrys, 2010)	SES, ARIMA and ANN	Introduced a meta learning method to make model selection easier.
(Andrawis, Atiya, & El-Shisiny, 2011)	Moving average and ANN	Proposing a new hybrid method which is better in dealing with seasonality within the data.
(Khashei & Bijari, 2011)	ARIMA, ANN and Hybrid ARIMA-ANN	Introducing a hybrid model consisting of ARIMA combined with an artificial neural network which is more accurate than ARIMA or ANN on itself.
(Babu & Reddy, 2014)	ARIMA, ANN and Hybrid ARIMA-ANN	Expanding the hybrid ARIMA-ANN model by first using a moving average filter to find the volatility of the data.
(Oliveira & Ludermir, 2016)	ARIMA and ETS	Proposing a hybrid model consisting of ARIMA and support vector regression. They also used a step to decompose the data based on low and high volatility using ETS.
(Raza, Nadarajah, & Ekanayake, 2017)	ARIMA and ANN	Created a forecasting model that consisted of a backpropagation neural network, ARIMA, feed forward neural network, wavelet transform and radial basis function. This combined model produces more accurate results for both seasonal daily and seasonal weekly data.
(Panigrahi & Behera, 2017)	ETS, ARIMA, ANN and Hybrid ETS-ANN	Using a hybrid model consisting of ETS and an artificial neural network they forecast both linear and non-linear time series.
(Parmezan, Souza, & Batista, 2019)	SES, ARIMA, ANN(LSTM)	Using multiple different models they compare the results over 95 different datasets to conclude which model can best be used on which sort of data.

3. Method

To forecast the cash flow of the account receivables at a large company with few large transactions and many small transactions I will compare the results of the following models:

- Simple Exponential Smoothing (SES)
- Auto Regressive Integrated Moving Average (ARIMA)
- Long Short-Term Memory neural network (LTSM)
- Hybrid model of ARIMA and Long Short-Term Memory neural network

To train, test and analyze these models I will be using four datasets of which three are synthetic datasets and one is the dataset of the case company. The three synthetic datasets all have different characteristics. The first synthetic dataset is a linear dataset with a constant linear increase. The second dataset consists of a yearly, 52-week, recurring sine pattern. Finally, the third synthetic dataset, consists of the yearly recurring pattern with an added monthly, 4-week, pattern.

3.1 Models

In this study I will be using four different forecasting methods. I will then analyze the results of these four methods to conclude which method works best in forecasting cash flow data for companies like the case company based on the root mean squared error and the mean absolute error. In this section I will introduce and explain the models that I will use in my analysis. The first two models, SES and ARIMA, are considered as classical forecasting models. The Long Short-Term Memory neural network model is a neural network forecasting model, which does not fall under the category classical forecasting models. Finally, the hybrid model of ARIMA and Long Short-Term Memory neural network combines both a statistical classic model with a neural network model to gain the best results of both models.

3.1.1 Simple Exponential Smoothing (SES)

The first model, Simple Exponential Smoothing, is a type of model that is best suited to forecast data which has no clear trend or seasonal pattern. This method is, compared to other existing methods, a simple method (Lemke & Gabrys, 2010). The idea behind this method is that the data of the future is going to be close to the data of the past. The model behind this method consists of only one parameter called the smoothing parameter. This smoothing parameter changes the importance of the most recent data. Meaning that either more recent or more distant data becomes more important. The model is called Simple Exponential Smoothing because the weight given to each past observation is reduced exponentially. The smoothing parameter thus changes these weights. The value of this smoothing parameter can range between 0 and 1, where lower values will give more weight to past observations and higher values give more weight to more current data. Lower values of the smoothing parameter then output a smoother forecast line due to the fluctuations that are averaged out. Higher values of the smoothing parameter outputs a more jagged forecast because the fluctuations are not averaged out (Hyndman & Athanasopoulos, 2018) (Parmezan, Souza, & Batista, 2019).

3.1.2 Auto Regressive Integrated Moving Average (ARIMA)

The ARIMA model is a category of statistical models used for the analysis and forecasting of time series data. It was introduced in 1970 by Box and Jenkins and has been a prominent method in forecasting financial time series data (Siami-Namini, Tavakoli, & Namin, 2018) (Adebbiyi, Adewumi, & Ayo, 2014). ARIMA has three main aspects:

Autoregression (p): The use of the dependent relationship between an observation and a p number of lagged observations.

Integrated (d): the use of the difference of observations, where d is the number of times that observations are differenced.

Moving Average (q): The use of the dependency between an observation and the residual error of the moving average of lagged observations, where q is the moving average window.

The complete ARIMA model gives ARIMA(p,d,q). To successfully use the ARIMA model it is not necessary to use the three components of the ARIMA model. This is partly because the integration, the differencing, can be done in the preprocessing of the data. With this in mind the following models can be made from the complete ARIMA model (Adebbiyi, Adewumi, & Ayo, 2014).

ARIMA(p,0,0) = AR(p): When the model is for example ARIMA(1,0,0) it is a first-order autoregressive model. This model can be used when the data can be predicted by a multiple of its previous values plus a constant.

ARIMA(0,0,q) = MA(q): Is a moving average model, where a moving average term is the past error multiplied by a coefficient. (PennState Eberly College of Science, 2022)

ARIMA(p,0,q) = ARMA(p,q): The main benefit of this model is that it is able to adjust to complex time series data with less terms than the complete ARIMA model. The "I" in the ARIMA model is the amount of differencing in the model, when no differencing is used the ARIMA model becomes an ARMA model.

The above models are all non-seasonal ARIMA models, but ARIMA is also capable of processing and modelling seasonal data. The seasonal ARIMA model is the normal ARIMA model with the added seasonal terms which is written as ARIMA(p,d,q)(P,D,Q)m where the first (p,d,q) are the normal non-seasonal model parameters and the second (P,D,Q)m are the seasonal parameters. In the seasonal part of the model the m is the number of observations per year.

Seasonal autoregression integrated moving average, also called SARIMA, is best used with data that has seasonal variations that are not addressed by the first difference of the normal ARIMA model. The selection between the two mostly relies on whether the data series has a trend, ARIMA would be best, or a trend

and seasonally component, then SARIMA would most likely give better results (Parmezan, Souza, & Batista, 2019).

Take for example an SARIMA model with S = 12, so a monthly model as each year has 12 months. This SARIMA model will then beside the normal ARIMA prediction also use the first order seasonal difference to predict *Xt* using *Xt-12*. Thus looking at the same month of last year. If the model would be set to use the second order seasonal autoregressive model, the model would use the two months of both one year back and two years back. So for example when talking about forecasting a value for January 2022 it would use the periods January 2021 and January 2020 (Introduction to ARIMA, 2022).

Seasonal differencing with SARIMA is defined as the difference between the value on time t and the value on the lag of time that is a multiple of the number of observations per year (S). This seasonal differencing has as an effect that it removes the seasonal trend of the data. It is possible that the differences per period within the year are relatively the same over time, resulting in a stationary data series.

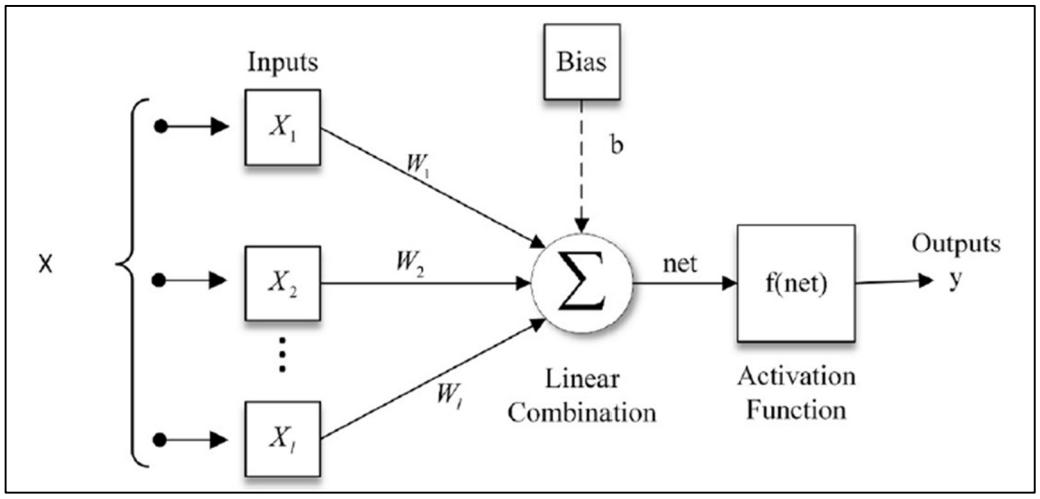
3.1.3 Long Short-Term Memory neural network (LSTM)

Artificial neural networks are a form of machine learning models made to emulate the human brain in processing information. Artificial neural networks are driven by data, because of this a misspecification in the model will have less impact on the performance than when the parameters for, for example, ARIMA would be incorrect. In the research the use of artificial neural networks as prediction and forecasting models has been increasing. Artificial neural networks range from simple artificial neural networks, such as a neural network with one hidden layer, to more advanced neural networks, such as recurrent neural networks.

The most basic form of an artificial neural network is the artificial neural network with one hidden layer which can be used for, besides other uses, classifying

classes that are linearly separable. This perceptron is depicted in figure 1 below. The single neuron consists of multiple parts. First is the input data Xi, consisting of Xi amount of data points. Each data point has a weight associated with it. This weight is the measure of importance of the data point associated with it. All the data points combined with their associated weights are then summed up. The net value, labelled as *net* in the figure, is the combination of these summed weights with a bias *b,* also called a threshold. The final output of the neuron *y* is computed by an activation function. The output of this activation function can be both continuous and binary. Through training the network the weights are adjusted, such that the model becomes more accurate. Training a model is done a number of iterations where for each iteration the model can adjust the weights such that the outcome improves (Hossein Abbasimehr, 2020) (Parmezan, Souza, & Batista, 2019).

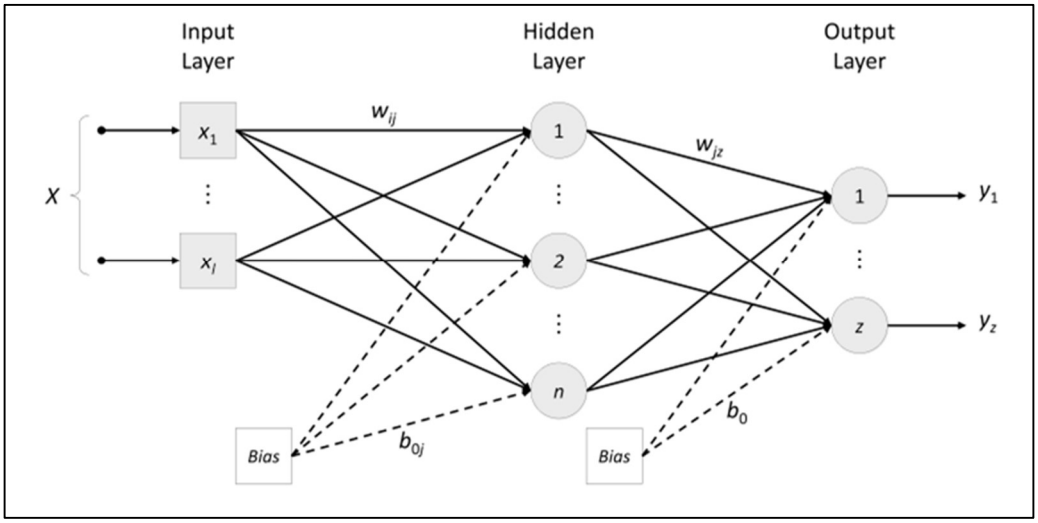
Figure 1: The perceptron



(Hossein Abbasimehr, 2020)

The perceptron, the neural network explained above, is the simplest form of an artificial neural network. A common, more advanced, neural network is the multilayer perceptron, which can be seen in figure 2. This neural network consists of multiple hidden layers and is special as is it trained through a backpropagation learning algorithm (Hossein Abbasimehr, 2020).

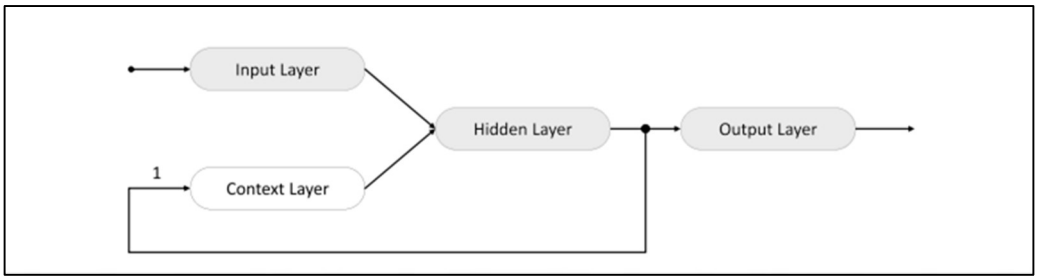
Figure 2: The structure of a multilayer perceptron with a single hidden layer



(Parmezan, Souza, & Batista, 2019)

The neural networks explained above are neural networks that have one or more hidden layers, but both are categorized as feed-forward neural networks. Because both the perceptron and the MLP neuron to neuron connections and signals are in one direction, from input (x) to output (y). Another artificial neural network category is the recurrent neural network. In the recurrent neural network category the connections and signals between the neurons can form cycles and the signals within these models can move different directions. The most basic recurrent neural network is the Simple Recurrent Network, in short SRN. In this Simple Recurrent Neural Network the active state of the hidden layer at a certain time is conditioned on the previous state by a context layer (Parmezan, Souza, & Batista, 2019). For an overview of the structure of the recurrent neural network see below figure 3.

Fig 3: The structure of a Simple Recurrent Neural Network



(Parmezan, Souza, & Batista, 2019)

The class of recurrent neural networks are efficient at for example speech classification and prediction. Specifically data where the sequences of the input and output data are dependent. This is the reason why recurrent neural networks are often used to predict time series data (Namin & Namin, 2018).

Even though recurrent neural networks have been very effective and accurate they do have two drawbacks. The first drawback is the problem of exploding gradients. The exploding gradient problem occurs when the algorithm assigning the importance of the weights gives a very high importance to the weights without a specific reason. As a result the model becomes unstable and is no longer able to learn from the training data. The second drawback is the vanishing gradient problem. The vanishing gradient problem is the opposite of the exploding gradient problem. It refers to the problem that the algorithm can assign very low values to the importance of the weights. This has as result that the model cannot learn the correlation between events, making the model unable to learn from the training data (Pascanu, Mikolov, & Bengio, 2013) (Siami-Namini, Tavakoli, & Namin, 2018).

A more complex recurrent neural network is the Long Short-Term Memory neural network. This neural network is part of the class of recurrent neural networks, but it has a gated architecture. This gated architecture is a proven way to deal with the vanishing and exploding gradients problem. The architecture of the Long Short-Term Memory neural network allows the network to learn longer range dependencies without having the risk of vanishing and exploding gradients (Parmezan, Souza, & Batista, 2019). The Long Short-Term Memory neural

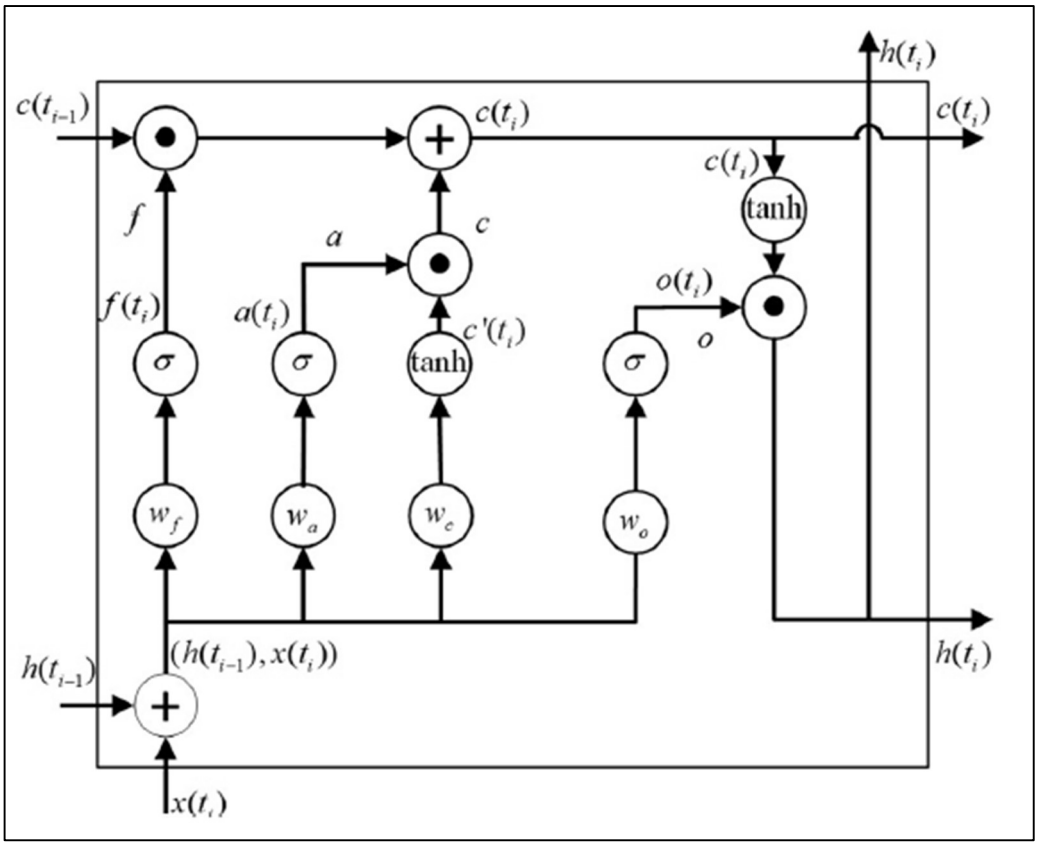
network is designed like this to capture long-term dependencies, making it a good neural network model for time series data (Ioannis E. Livieris, 2020). Long short-term memory neural networks have been successful in for example stock market prediction, analysis of audio data, text generation and translation (Greff, Srivastava, Koutn´ık, Steunebrink, & Schmidhuber, 2017). In existing literature this method is also used to forecast cash flow (Weytjens, Lohmann , & Kleinsteuber, 2019).

The structure of a Long Short-Term Memory neural network differs from the previous explained recurrent neural network because it has a cell and gates that manages the information flow within the network. The Long Short-Term Memory neural network cell contains an input gate, a forget gate, internal state and an output gate. Compared to the most basic neural network cell, which only has an input layer and an output layer the LSTM cell can be used to store long term time dependency information using the internal state (Greff, Srivastava, Koutn´ık, Steunebrink, & Schmidhuber, 2017) (Hossein Abbasimehr, 2020).

The following figure shows the layout of the Long Short-Term Memory neural network. The notations in the figure have the following meaning (Hossein Abbasimehr, 2020):

- x(t) is the input value
- h(t) is the output value at time t
- c(t) is the cell state at time t
- b_x are the biases of x, where x can be the input gate, the forget gate, the internal state and the output gate.
- W_x are the weights matrixes of x where x can be the input gate, the forget gate, the internal state and the output gate.
- a(t) are the output results for the input gate, the forget gate, the internal state and the output gate.

Figure 4: The structure of a Long Short-Term Memory cell



(Hossein Abbasimehr, 2020)

The mathematical calculation of a Long Short-Term Memory model is as follows (Hossein Abbasimehr, 2020):

$$a(t_i) = \sigma(w_a x(t_i) + w_{ha} h(t_{i-1}) + b_a) \quad (1)$$

$$f(t_i) = \sigma(w_f x(t_i) + w_{hf} h(t_{i-1}) + b_f) \quad (2)$$

$$c(t_i) = f_t \times c(t_{i-1}) + a_t \times \tanh(w_c x(t_i) + w_{hc}(h(t_{i-1}) + b_c) \quad (3)$$

$$o(t_i) = \sigma(w_o x(t_i) + w_{ho} h(t_{i-1}) + b_o) \quad (4)$$

$$h(t_i) = o(t_i) \times \tanh(c(t_i)) \quad (5)$$

In this calculation the σ and tanh are the activation functions of the model. The × in the last calculation means a point-wise multiplication. The Long Short-Term Memory neural network learns in the following steps (Greff, Srivastava, Koutn´ık, Steunebrink, & Schmidhuber, 2017):

1. Computing the output of the model using equations 1 to 5.
2. Computing the error between the output data and input data of each layer.
3. The resulting error is reversely propagated to the input gate, the cell and the forget gate.
4. Based on the resulting error term the weights of the gates are updated using an optimization algorithm.

These four steps are then repeated for a given number of times, which will result in the optimal values of weights and biases.

3.1.4 Hybrid ARIMA Long Short-Term Memory neural network model

The last model I will use is a combined model using ARIMA to find the linear patterns and Long Short-Term Memory neural network to find the nonlinear patterns from the residuals of the ARIMA model. The motivation behind the use of this combined model comes from the difficulty to determine whether a model is mainly composed of linear patterns or of nonlinear patterns. This difficulty also means that it is not always clear whether to use a model that is best in recognizing the linear patterns, such as ARIMA, or whether it is best to use a model which is better in recognizing the nonlinear patterns, such as a Long Short-Term Memory neural network. Therefore it is normal for researchers to try and compare multiple different models to see which model works best with their data. The problem with this selection is that this model is not necessarily the best model for future use. The forecasting data can change over time such that more linear patterns appear in the data where first only nonlinear patterns existed or the other way around.

By combining ARIMA with a neural network model this future uncertainty is limited. This combined model also means that model selection can be easier since both models are applied. Also because most real-world time series data consists of data which has both linear and nonlinear patterns it is useful to have a model

which can recognize both. When using only non-combined models these real-world time series data cannot use both the nonlinear patterns and the linear patterns to make the most accurate forecast. This is because ARIMA cannot deal well with nonlinear patterns and a Long Short-Term Memory neural network on itself cannot handle the linear patterns as good as ARIMA can. By combining ARIMA with a Long Short-Term Memory neural network more complex patterns of both linear and nonlinear structure can be modelled more accurately (Zhang, 2003).

In the existing literature most studies that compare multiple different models on multiple different time series datasets also conclude that not one method is best for all the different datasets (Khashei & Bijari, 2011) (Oliveira & Ludermir, 2016) (Panigrahi & Behera, 2017). Due to this a combined model can be a more universal method that can be applied in different cases and on different time series datasets.

The hybrid model consists of two parts. The first part is the linear component of the data, the second part is the nonlinear part of the data. This separation of the data can be written in the following way.

$$y_t = L_t + N_t$$

In which L_t represents the linear component at time *t*, N_t represents the nonlinear component at time *t* and y_t is the time series value at time *t*. For the hybrid model I start with estimating the linear component. To do this I first use ARIMA on the time series data set to get the forecast estimate of the linear component. To then use the Long Short-Term Memory neural network to enhance this forecast I train the neural network on the residuals of the ARIMA model. The residuals are calculated as follows.

$$R_t = y_t - \hat{L}_t$$

Where R_t is the residual value at time *t*, y_t is the actual time series value at time *t* and $\hat{L}_t$ is the forecast value at time *t* from the linear model, in this case ARIMA.

In normal model analysis the residuals are an important measure of the fit of the model, as it measures how far off the predicted value is from the actual value at time *t* (Abonazel & Abd-Elftah, 2019). By using the residuals of the linear model as training data for the Long Short-Term Memory neural network model the neural network model can use all its pattern detection on the nonlinear part of the model, as the linear patterns are already modelled by the ARIMA model.

To calculate the final prediction of the hybrid model it then combines the forecasts of the ARIMA model and the Long Short-Term Memory neural network model by summing both values. This will result in the following calculation.

$$\hat{y}_t = \hat{L}_t + \hat{N}_t$$

Where $\hat{y}_t$ is the forecasted value at time *t*, $\hat{L}_t$ is the forecasted value of the ARIMA model and $\hat{N}_t$ is the forecasted value of the residual at time *t* which is forecasted by the Long Short-Term Memory neural network model.

Because the workings of both ARIMA and a Long Short-Term neural network are explained in detail I will now shortly summarize the hybrid model method. The hybrid model consists of two calculation steps. The first is the ARIMA model which will forecast the value using the linear patterns which ARIMA can recognize. The second is the Long Short-Term Memory neural network model which will use the residual values from the ARIMA model to recognize the nonlinear patterns within the data. The forecast of the ARIMA model and the forecast of the Long Short-Term Memory neural network are then combined to get the final forecast value.

3.2 Hypothesis

The performance per model discussed in this paper depends on the patterns within the data. This is because each model can handle either linear or nonlinear patterns better than other models. The drawback of cash flow data is that the

amount per payment often varies, which makes forecasting the exact amount with high accuracy unlikely. But I expect that the data of the cash flow of the accounts receivable does have both linear and nonlinear patterns. Therefore my hypothesis is that the Simple Exponential Smoothing model will have the lowest forecasting accuracy. The ARIMA model will recognize the linear patterns within the data such data its forecasting accuracy is higher than that of the Simple Exponential Smoothing model. I expect that the Long Short-Term Memory neural network will have a higher forecasting accuracy than ARIMA, mostly because I expect to see more nonlinear patterns within the cash flow of the accounts receivable data. I expect to see more nonlinear patterns because of the varying payment amounts and the non-standard structure of the accounts receivable. It is also often the case in existing literature that for time series data the neural network and specifically the Long Short-Term Memory neural network has a higher forecasting accuracy than ARIMA. Finally, the hybrid model of ARIMA and the Long Short-Term Memory neural network is according to the literature good in recognizing both the linear and the nonlinear patterns (Zhang, 2003) (Oliveira & Ludermir, 2016). Therefore I expect the hybrid model to have the highest forecasting accuracy from the models compared in this paper.

4. Data description

4.1 Synthetic data

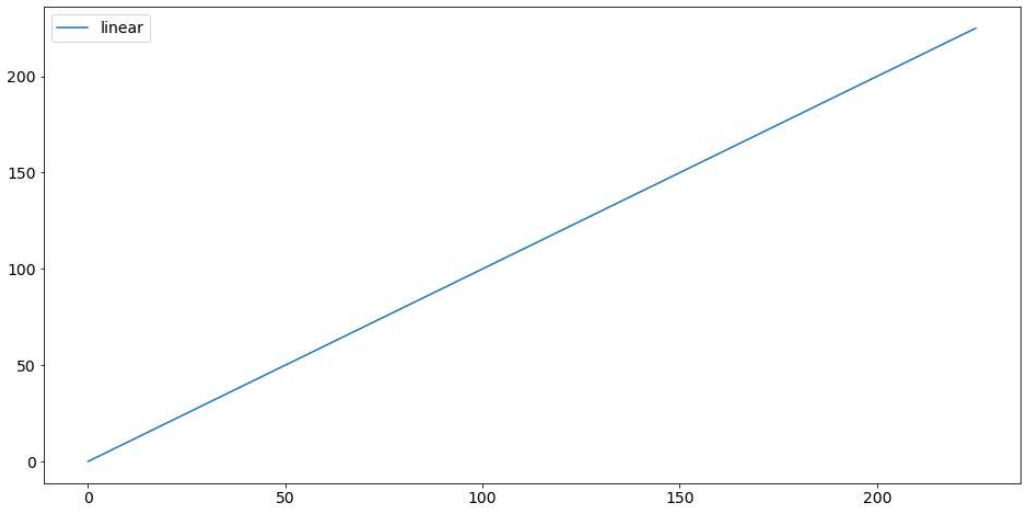
Before using the models to forecast the cash flow of the accounts receivable I will test the models using synthetic datasets. Using synthetic datasets to test the models gives multiple advantages:

- It is easier to recognize errors in the models or in the data preparation.
- It creates the opportunity to see how well the models can learn known patterns within the synthetic dataset.
- It gives more opportunity to compare model accuracy between datasets.
- By creating a synthetic dataset it is possible to create data which can represent any situation.
- It avoids confidentiality and privacy issues.

4.1.1 Linear dataset

The linear dataset consists of 260 data points, which start at 0 and have a constant increase of 1 per data point.

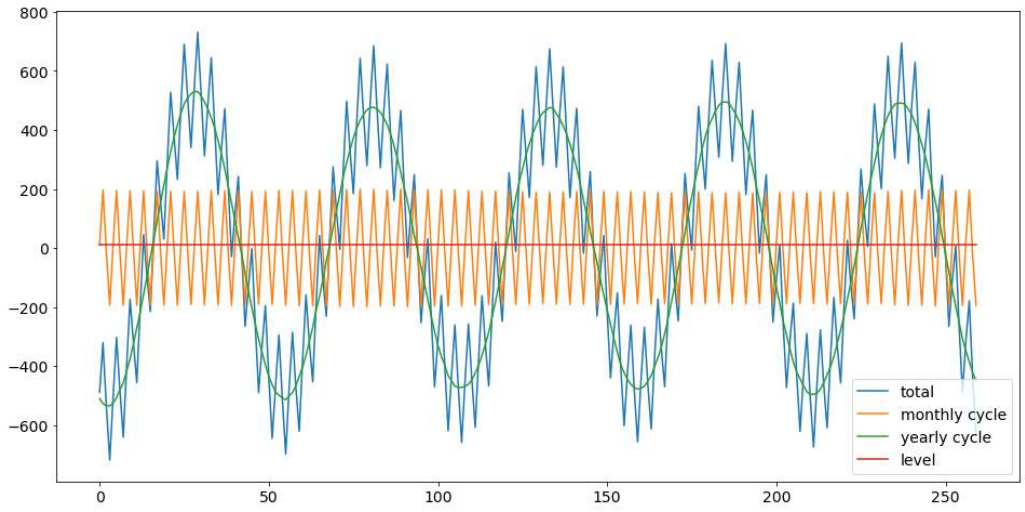
Graph 1: Linear synthetic dataset



4.1.2 Yearly and monthly dataset

The second two synthetic datasets are created using sine functions. First a yearly, 52-week, recurring sine pattern consisting of 5 years is created. Then a monthly, 4-week, recurring sine pattern is created. These two patterns are then added up to create a dataset which has yearly and a monthly cycle. In the following graph you can see the monthly pattern (orange), the yearly pattern (green) and the combined monthly-yearly pattern (blue).

Graph 2: Yearly and monthly synthetic dataset



4.2 Company Data

To forecast the cash flow of the accounts receivable of case company I will use the accounts receivable dataset[1]. This dataset consists of all the accounts receivable entries registered in their business operations database. The products the case company produce have a very high value, but the number of products produced per year is small. This results in accounts receivable which consist of a small number of very large value entries and many small value entries.

[1] Description of the company data and summary statistics are limited due to confidentiality.

The data of the case company consists of all accounts receivable entries over 226 weeks. Meaning that there are multiple entries per day and multiple dates associated to that entry. The entries have a posting data, an invoice data, a net due date and a clearing data. The posting date is not important for this study as it is the date on which the accounts receivable entry was added in the database. The invoice date is the date on which the invoice is sent to the customer. The net due date is the final date on which the payment should be received. This date is calculated per accounts receivable entry by adding the payment term to the invoice date. For example an invoice was sent on the first of January and the payment terms are such that the amount must be paid within 90 days, then the net due date is 90 days after the first of January. Finally, the clearing date of the entry is the date on which the payment was cleared, which is the date of payment and thus the cash flow date.

4.3 Data Preparation

4.3.1 Company Data preparation

To forecast the cash flow of the accounts receivable I have summed the data per week based on the clearing date. This results in the actual cash inflow of the accounts receivable per week. I also convert the data type of the amount of interest column to integer such that there are no data type errors. The code below is the base dataset loading and creation code that is used in all the company data models. The dataset is loaded with the clearing date as index such that the resample function will sum the amount per week based on the clearing date.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt


#Loading in the dataset with the clearing date as index
df = pd.read_csv("AR_Items_All_CoCodes_2018_2022.csv", delimiter = ',',en-
coding = "UTF-16",header=0, infer_datetime_format=True,
parse_dates=['Clearing_date'], index_col=['Clearing_date'])
#Set the datatype of the amount column to type integer
df['Amount_gc_ecc'] = df['Amount_gc_ecc'].astype(int)

#Sum the amount by week on the index of the clearing date
df = df.resample('W').sum()

#Selecting the timespan of the dataset
df = df['YYYY-MM-DD':'YYYY-MM-DD']
```

4.3.2 Data scaling

I normalize the data using the MinMaxScaler function from the sklearn package. This function transforms the data to a value between 0 and 1 by scaling the data such that the smallest value of the dataset amount column is 0 and the largest value of the dataset amount column is 1. I first use fit_transform on the training set such that the scaler is fit on the training dataset end the training set is scaled between 0 and 1. I continue by scaling both the validation and test sets using the transform function. By first fitting the scaler on the training data there is no possibility of future values of the validation or test sets to influence the model training process.

Below is the code used to split the data into a train and a test set for the SES and ARIMA model. I then scale the data such that the scaler is fit on the train data and the test data is scaled on the same scale.

```
#Selecting the number of test weeks
N_test_weeks = 32

#Splitting the dataset in train and validate sets
y_to_train= y[:len(y)-N_test_weeks]
y_to_test = y[len(y)-N_test_weeks:]

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
y_to_train_scaled = scal-er.fit_transform(pd.DataFrame(y_to_train.values))
y_to_test_scaled = scaler.transform(pd.DataFrame(y_to_test))
```

For the Long Short-Term Memory neural network the code to split and scale the datasets is different. This is because to optimize the neural network model a validation set is also needed, meaning that the dataset is split into a train, validation and test set. This validation set is then used to optimize the model. See below the code used to scale the dataset for the Long Short-Term Memory neural network model. The Long Short-Term Memory neural network dataset consists of two columns of which the second column is a 48-week shift of the data such that the values of last year are within the data. At the end of the following code I again stack the scaled train, validation and test datasets to create the correct input shape of the data in the next section.

```
#creating the lagged 1 year variable
df["Amount_lagged"] = df.shift(48)

#Number of validation and test weeks
N_val_weeks = 32
N_test_weeks = 32

#selecting only the outcome amount data
train_data = dataset[:len(dataset)-N_test_weeks-N_val_weeks]
val_data = dataset[len(dataset)-N_test_weeks-N_val_weeks:len(dataset)-
N_test_weeks]
test_data = dataset[len(dataset)-N_test_weeks:]

#scaling of the data
scaler = MinMaxScaler()
scaled_train_data = scaler.fit_transform(train_data)
scaled_val_data = scaler.transform(val_data)
scaled_test_data = scaler.transform(test_data)

#combine the scaled datasets such that the data roll can be applied
dataset = np.vstack((scaled_train_data, scaled_val_data,
scaled_test_data))
```

4.3.3 Long Short-Term Memory neural network data preparation

The data preparation for the Long Short-Term Memory model is more complicated. This is because the input data for the model will consist of the previous four weeks combined with the current week and the next three weeks with a lag of one year. I create this lagged variable by creating a lagged 48-week shift of the amount column. I then select the past four weeks of both the normal amount and the lagged one-year amount. The input data then consists of the

previous four weeks, the next three weeks with a lag of one year and the value of the current week with a one-year lag.

To create this new shape I created a function called “lstm_data”. This function takes as input the data frame with scaled values of both the normal and the lagged columns. It then creates as output a new array consisting of the four previous weeks and the next four weeks with a lag of one year.

The input of the function is df and timestamps. Where df is the data frame of the amount per week and the amount per week with a lag of 48 weeks. The variable timestamps is the number of previous weeks needed in the output dataset. Meaning that for example if the value for timestamps is 4 then the four previous weeks and the four previous weeks with a lag of one year will be the output for one forecast week.

```
def lstm_data(df, timestamps):
    array = np.empty((0,df.shape[1]))
    range_ = df.shape[0]-(timestamps-1)
    for t in range(range_):
        dfp = df[t:t+timestamps, :]
        array = np.vstack((array, dfp))

    df_array = array.reshape(-1,timestamps, array.shape[1])

    return df_array
```

For a neural network to learn to forecast it needs a timeseries to learn from. This timeseries I create using the “lstm_data” function. The resulting training data per week will look like this:

$[W_{t+1}, W_{t+2}, W_{t+3}, W_{t+4}] = [W_t, W_{t-1}, W_{t-2}, W_{t-3}], [W_{t-48}, W_{t-49}, W_{t-50}, W_{t-51}]$

Where W_t is week t. This is the shape the data needs to be used as input data for the model. But the output of the “lstm_data” function gives the following for each week: $[W_t, W_{t-1}, W_{t-2}, W_{t-3}], [W_{t-48}, W_{t-49}, W_{t-50}, W_{t-51}]$

In the following code I first create the dataset using the “lstm_output” function with the number of previous weeks set to four. I then select the output data and define it as Y_df. To get the data related to each data point in this dataset I roll

the X data related to these data points back one week to get the following shape of data per datapoint:

$[W_{t+1}, W_{t+2}, W_{t+3}, W_{t+4}] = [W_t, W_{t-1}, W_{t-2}, W_{t-3}], [W_{t-48}, W_{t-49}, W_{t-50}, W_{t-51}]$

With the new X and Y datasets I then again split the data into the train, validation and test datasets. From the train dataset the first 52 entries have wrong or no data due to the 48-week shift and the 4 week roll of the data, therefore these entries are dropped.

```
#creating the LSTM data input shape
dataset = lstm_data(dataset, 4)

#splitting Y_df and X_df where X_df is rolled 4 weeks back such that the
past 4 weeks forecast the upcoming 4 weeks
Y_df = dataset[:,:,:1]
X_df = np.roll(dataset, 4, axis = 0)


#Creating train set
Y_df_train = Y_df[:len(Y_df)-N_test_weeks-N_val_weeks,:,:]
X_df_train = X_df[:len(Y_df)-N_test_weeks-N_val_weeks,:,:]

#Creating validation set
Y_df_val = Y_df[len(Y_df)-N_test_weeks-N_val_weeks:len(Y_df)-
N_test_weeks,:,:]
X_df_val = X_df[len(X_df)-N_test_weeks-N_val_weeks:len(X_df)-
N_test_weeks,:,:]

#Creating test set
Y_df_test = Y_df[len(Y_df)-N_test_weeks:,:,:]
X_df_test = X_df[len(X_df)-N_test_weeks:,:,:]

#deleting first year from both due to the lag the shift created and the
roll of 4 weeks (48+4)
#Only needed in train set because the X_df was shifted foreward
Y_df_train = Y_df_train[52:]
X_df_train = X_df_train[52:]
```

5. Model creation

In this section I will introduce the programming of each model and I will explain how the optimal parameters have been chosen per model. To compare the model accuracy each model will forecast the next 4 weeks. The models used in this analysis have been created using python and the Long Short-Term Memory neural network uses Keras, which is an open-source neural network library.

5.1 Simple Exponential Smoothing model

The Simple Exponential Smoothing model makes use of the function "SimpleExpSmoothing" from the package "statsmodels". In the code below the function "forecast_loop" is defined. This function takes the train and test data as input, which it then uses to predict a number of weeks ahead such that for each of those weeks the next 4 weeks are forecasted. This then results in an output shape of the data of a list of 4-week forecasts. In the code below this is done for the Simple Exponential Smoothing model.

```
from statsmodels.tsa.api import SimpleExpSmoothing

def forecast_loop(train, test, n_weeks, forecast_weeks):
    prediction_4_week = []
    for i in range(n_weeks - forecast_weeks + 1):


        fit = SimpleExpSmoothing(train).fit()
        prediction = fit.forecast(4)
        train = np.append(train, test[i])
        #print(prediction)
        prediction_4_week.append(prediction)

    return prediction_4_week

prediction = forecast_loop(y_to_train_scaled, y_to_test_scaled, 32, 4)
```

To calculate the accuracy both the actual values and the forecasted values need to be in the same format such that the root mean squared error and the mean absolute error can be calculated. In the code below I create a list of both the actual values and the forecasted values. To compare the actual values with the predicted values I first need to reshape the actual values to the shape such that it is a list of

four-week amounts. I do this using the "data_to_4_weeks" function. This function has as input a list of values and as output a list of lists where each smaller list consists of four weeks of data. Such that it reshapes the data as shown below.

$$[1,2,3,4,5,6,7,8] \rightarrow \begin{bmatrix}[1,2,3,4]\\[2,3,4,5]\\[3,4,5,6]\\[4,5,6,7]\\[5,6,7,8]\end{bmatrix}$$

As the forecast_loop function already has this shape as output I only create a list of the output. I then continue by calculating the root mean squared error and the mean absolute error using the "mean_squared_error" and "mean_absolute_error" functions from the sklearn metrics package.

```
#data to 4 weeks function
def data_to_4_weeks(test, n_weeks, forecast_weeks):
    actual_4_week = []
    for i in range(n_weeks - forecast_weeks+1):
        week = pd.DataFrame([test[i],test[i+1],test[i+2],test[i+3]])
        actual_4_week.append(week)


    return actual_4_week

#creating actual 4 week forecast shape and list
actual = data_to_4_weeks(y_to_test_scaled, 32,4)
actual = np.array(actual)
actual_list = []
for i in actual:
    for i in i:
        for i in i:
            actual_list.append(i)

#creating prediction list
prediction = np.array(prediction)
prediction_list = []
for i in prediction:
    for i in i:
        prediction_list.append(i)

#calculation of the root mean squared error and the mean absolute error
from sklearn.metrics import mean_squared_error
RMSE_SES = mean_squared_error(actual_list, prediction_list,squared = False)
print(RMSE_SES)

from sklearn.metrics import mean_absolute_error
MAE_SES = mean_absolute_error(actual_list, prediction_list)
print(MAE_SES)
```

5.2 ARIMA model

The ARIMA model is a more complex model which takes multiple parameters that have an effect on the accuracy of the model. Since the model I will be using will be the seasonal ARIMA model I will not only have to optimize the three normal parameters, but also the three seasonal parameters (Box, Jenkins, & C., 2016). The seasonal ARIMA model also takes as input a measure of datapoints in each year, this will be set to 52 as there are 52 weeks in a year.

5.2.1 Hyperparameter optimization

To optimize the hyperparameters of the seasonal ARIMA model I have used the "auto_arima" package from the library "pmdarima". In the code below is the method I have used to find the optimal parameters for this model.

```
#Stationary test
from pmdarima.arima import ADFTest
adf_test = ADFTest(alpha = 0.05)
adf_test.should_diff(y_to_train)

#Hyperparameter selection using auto_arima
from pmdarima import auto_arima
arima_model = auto_arima(y_to_train_scaled, m=52, seasonal = True, trace = True)
```

I first used the function "ADFTest" from the "pmdarima" library to test whether the data is stationary. The outcome of this test is then used to decide whether the stationary parameter in the "auto_arima" function should be set to True or False. In the second part the "auto_arima" function is used to find the optimal parameters for the most accurate model.

The "auto_arima" function tests the seasonal ARIMA model with different parameters to measure which parameters work best (Garima Jain, 2013). For our data the result was that the most accurate seasonal ARIMA model would be the ARIMA(p,d,q)(P,D,Q)[m] model with the following parameters:

Linear dataset:	(0,1,0)(0,0,0)[52]
Yearly dataset:	(1,0,1)(1,0,0)[52]
Monthly-yearly dataset:	(3,0,2)(2,0,0)[52]
Company dataset:	(5,1,2)(2,0,0)[52]

5.2.2 ARIMA model parameters interpretation

The results of the auto_arima function give insights into the data. This is because the form of the model depends on what patterns there are in the data. Starting with the most simple of the ARIMA models, the ARIMA model for the linear dataset, which consists of ARIMA(0,1,0)(0,0,0)[52]. This means that all parameters are 0 except the differencing parameter which is 1. This model predicts using the first difference of the last observation, meaning that the prediction of the next period is the current period with the difference of the current period. This gives the following equation:

$$\hat{t}_{+1} = t + (t - t_{-1})$$

Which for a dataset with a constant linear increase gives the best result as the increased amount is the same as the difference between current and last period.

For the yearly dataset the optimal ARIMA model is the ARIMA(1,0,1)(1,0,0)[52]. Meaning that the autoregressive term, the seasonal autoregressive term are and the number of forecast errors in the prediction equation are 1. This in essence makes it an ARMA model, which takes into account the past values and the past error terms. By also having the seasonal autoregressive term set to 1 this complete model becomes an SARMA model, which uses the past values, the past error terms and the values of last season.

Both the models for the monthly-yearly dataset and the company dataset are more complicated than the above two, but they work similar. The ARIMA model for the monthly-yearly dataset is a more complicated SARMA model where the

autoregressive term is 3, the seasonal autoregressive term is 2 and the number of forecast errors in the prediction equation is 2. The ARIMA model for the company dataset is even more complicated. It is a SARIMA model where the autoregressive term if 5, the differencing term is 1, the number of forecast errors in the prediction equation is 2 and the seasonal autoregressive term is also 2.

5.2.3 Final ARIMA model

With the optimized hyperparameters that resulted from the "auto_arima" function I build the following ARIMA model. In the code below the ARIMA model for the linear dataset is used. Again I make use of the forecast_loop function to create a list of 4-week forecasts. This list is then reshaped such that the root mean squared error and the mean absolute error can be calculated.

```
from pmdarima.arima import ARIMA as pmdARIMA

def forecast_loop(train, test, n_weeks, forecast_weeks):
    prediction_4_week = []
    for i in range(n_weeks - forecast_weeks + 1):
        arima_model = pmdARIMA((0,1,0),seasonal_order=(0, 0, 0,
52),trace=True)
        arima_model.fit(train)
        prediction = pd.DataFrame(arima_model.predict(n_periods = 4))
        train = np.append(train, test[i])
        #print(prediction)
        prediction_4_week.append(prediction)

    return prediction_4_week

prediction = forecast_loop(y_to_train_scaled, y_to_test_scaled, 32, 4)

prediction = np.array(prediction)

prediction_list = []
for i in prediction:
    for i in i:
        for i in i:
            prediction_list.append(i)

from sklearn.metrics import mean_squared_error
RMSE_ARIMA = mean_squared_error(actual_list, prediction_list,squared =
False)
print(RMSE_ARIMA)

from sklearn.metrics import mean_absolute_error
MAE_ARIMA = mean_absolute_error(actual_list, prediction_list)
```

5.3 Long Short-Term Memory neural network model

Building the Long Short-Term Memory neural network I used the Keras neural network library. A neural network can consist of many different layers that all together become a neural network model. Just as with the ARIMA model, this neural network model too needs optimization. This is also important because a Long Short-Term Memory neural network needs more than just a LSTM layer, but it also needs for example a Dense layer, a layer of neurons that is connected to each neuron of the preceding layer, which has the desired output shape.

5.3.1 Model layers optimization

To optimize the number of layers and type of layers of the neural network model I have used the “keras_tuner” package. The “keras_tuner” package finds the optimal number of layers, nodes and dropout by building and testing multiple different specified models. This process is further explained below in the code explanation.

To optimize the model I defined the function “build_model” with the possible layers inside. The “build_model” function is itself the neural network model, but with a variable number of layers. Each “for” loop in the function gives a variable number of layers, both for the LSTM layers as for the Dense layers. Only the last Dense layer is a fixed layer as it is needed to get a one value output, the predicted value. Within each for loop multiple layers can be set, each with a variable number of neurons. The value for the number of layers is changed by the “hp” input which has a minimum value, a maximum value and a step value. It then tries in this example values from 32 to 128 with a step value of 32, which will result in the following possibilities: [32, 64, 96, 128]. Combined this function tries multiple different setups of the model, with a varying number of layers and a varying number of neurons in each layer.

```
from kerastuner.tuners import RandomSearch
from kerastuner.engine.hyperparameters import HyperParameters
from keras.callbacks import EarlyStopping, Callback

def build_model(hp):
    model = Sequential()
    for i in range(hp.Int('n_layers_LSTM', 0, 2)):
          model.add(LSTM(hp.Int(f'LSTM_{i}_units',
                                min_value=32,
                                max_value=128,
                                step=32),activation = "relu" , re-
turn_sequences = True))

       model.add(LSTM(hp.Int('LSTM_out_units', min_value=32,
max_value=128, step=32),activation='relu', return_sequences = False))

    for i in range(hp.Int('n_layers_Dense', 0, 2)):
          model.add(Dense(hp.Int(f'Dense_{i}_units',
                                 min_value=32,
                                 max_value=128,
                                 step=32),activation = "relu" ))

       model.add(Dropout(hp.Float('Drop-
out_rate',min_value=0,max_value=0.8,step=0.1)))

       model.add(Dense(4, activation='relu'))

       model.compile(loss='mean_squared_error', optimiz-er='adam',metrics
= ['mse'])
    return model
```

The above build_model function has two for loops and 3 fixed layers. It starts with a for loop which can add between 0 and 2 Long Short-Term Memory neural network layers to the model. It then continuous by adding a Long Short-Term Memory neural network layer. I split the variable amount of Long Short-Term Memory neural network layers and the fixed one layer because for the model to work the last Long Short-Term Memory layer needs to have the parameter return_sequences set as False. This is because with the parameter set to True the Long Short-Term Memory layer will has an output which cannot be processed by the Dense layer which comes after the Long Short-Term Memory layers. When this parameter is set to True the Long Short-Term Memory layer will output all hidden states of all timesteps, while when it is set to False the layer will output only the last hidden state.

The code then continuous with a for loop which will add between 0 and 2 Dense layers to the model. All the Long Short-Term Memory and Dense layers explained above can all have a variable amount of between 32 and 128 nodes

with a step of 32. After this for loop a dropout layer is added which will have a parameter which can be between 0 and 0.8 with steps of 0.1. This dropout layer is used to combat overfitting of the model by randomly ignoring neurons. The final layer of the model is a Dense layer with 4 nodes which is used to create the desired model output of a 4-week forecast.

```
es = EarlyStopping(monitor='val_loss', patience=10, re-
store_best_weights=True)

tuner = RandomSearch(build_model, objective='val_loss', max_trials=100,ex-
ecutions_per_trial=3, overwrite = True)

tuner.search(X_df_train, Y_df_train, batch_size = 4, epochs=200, callbacks
= [es], validation_data=(X_df_val, Y_df_val))

best_model = tuner.get_best_models()[0]
best_parameters = tuner.get_best_hyperparameters(1)[0]
print(best_parameters.values)
```

In the code above I first define the early stopping function. This early stopping function looks at model performance and will stop the model from overtraining by looking at the performance of the model on the validation dataset. With the parameters in the above code this early stopping function will stop the model from training further if after 10 epochs the model has not increased in accuracy based on the accuracy of the model on the validation dataset. It will then restore the model to the last best form of the model.

I continue by defining the tuner which will tune the model defined in the build_model function. This tuner will then search for the optimal number of layers and neurons. The tuner will randomly create and train multiple forms of the model to see which form of the model has the best model accuracy (O'Malley, et al., 2019). In the tuner defined above the parameter max_trials is set to 100 which means that the tuner will create 100 random models based on the possibilities given in the build_model function. The parameter executions_per_trial is set to 3 meaning that for each form of the model the model is trained three times. This is necessary because a model can have different levels of accuracy each time it is trained, even though the number of layers and neurons stays the same.

The tuner is then called to tune the build_model model with the train data. The parameter batch_size is set to 4 meaning that the model weights are updated after 4 observations are passed the model. The parameter epochs is set to 200 meaning that for each model that is trained it will go over the whole training dataset 200 times. Then the tuner search will use the earlier defined early stopping function with the validation datasets as validation data.

5.3.2 Final Long Short-Term Memory neural network model

With the output of the Keras tuner function explained above I created the optimal model for this data based on the lowest root mean squared error of the model forecast on the validation set. The model below is the optimal model for the linear dataset based on the results of the Keras tuner.

In the following code I start by defining the training parameters which consist of the following parameters:

- Verbose: Is the output of the model during training.
- Epoch: The Epoch amount is the number times the model trains the complete model. Each epoch is one cycle of training the model.
- Batch size: Is the number of samples after which the weights of the model are updated.

The code continues with the creation of the model. Firstly, the model I use is the sequential model, which is able to process time series data beside other forms of data. I then add layers to the model, starting with one Long Short-Term Memory layer with 96 neurons. The second layer is a Dense layer with 128 neurons. These layers have as activation function the function “relu” which stands for Rectified Linear Unit. This activation function is a linear function which will output the input if the input is positive, in other cases it will output zero. The last Dense layer has only 4 neurons, this layer is necessary to get the 4-week forecast as output. I then compile the model and add the loss function of the root mean squared error and the optimize function which is ‘adam’. The Adam activation

function is an algorithm made for first order gradient optimization used to optimize the network weights (Kingma & Ba, 2015). Just as with the Keras tuner here too I use an early stopping function to reduce the chance of overfitting. I finally train the model using the train data and the earlier defined model parameters. During the training the early stopping function will use the validation data to measure the model accuracy on the validation data and stop the training process if the model becomes less accurate.

```
verbose, epochs, batch_size = 1, 1000, 4

#Defining the model
model = Sequential()
model.add(LSTM(units = 96 ,return_sequences = False ,   activa-
tion='relu'))
model.add(Dense(96,  activation='relu'))
model.add(Dropout(0))
model.add(Dense(4))
model.compile(loss='mse', optimizer='adam')

# fit network
es = EarlyStopping(monitor='val_loss', patience=50, re-
store_best_weights=True)

model.fit(X_df_train, Y_df_train, epochs=epochs, batch_size=batch_size,
verbose=verbose, validation_data = (X_df_val, Y_df_val), callbacks = [es])
```

With the defined and trained model I then predict the test weeks using the model. To compare the models I reshape the actual and predicted values such that the root mean squared error and mean absolute error can be calculated.

```
#Prediction values from model
prediction = model.predict(X_df_test)

#Actual values
actual = Y_df_test[:,1]

actual_list = Y_df_test.tolist()

LSTM_actual_list = []
for i in actual_list:
    for p in i:
        LSTM_actual_list.append(p[0])

LSTM_prediction_list = []
for i in prediction:
    for p in i:
        LSTM_prediction_list.append(p)

LSTM_4_week_RMSE = mean_squared_error(LSTM_actual_list, LSTM_predic-
tion_list, squared = False)
print(LSTM_4_week_RMSE)

from sklearn.metrics import mean_absolute_error
LSTM_4_week_MAE = mean_absolute_error(LSTM_actual_list, LSTM_predic-
tion_list)
print(LSTM_4_week_MAE)
```

5.4 Hybrid ARIMA Long Short-Term Memory neural network model

The hybrid model is a combination of both the ARIMA model and the Long Short-Term Memory neural network model. The hybrid model works by first modelling the linear part of the time series data using the ARIMA model. As the original time series data has not changed we can use the same method and parameters for this model as for the ARIMA model. The optimization of the ARIMA model for this time series data has already been explained in the ARIMA model chapter. The code in this section is the code used for the hybrid model for the yearly and monthly combined dataset starting with the ARIMA model. In the code below I first use the ARIMA model to get the ARIMA residuals which the Long Short-Term Memory model uses as train data.

```
#Creating the ARIMA model for the hybrid model
from pmdarima.arima import ARIMA as pmdARIMA

arima_model = pmdARIMA((3,0,2),seasonal_order=(2, 0, 0, 52),trace=True)
arima_model.fit(y_to_train_scaled)
prediction = pd.DataFrame(arima_model.predict(n_periods = 32))

residuals_arima = arima_model.resid()
residuals_arima = np.append(residuals_arima, residuals_test)
df = pd.DataFrame(residuals arima)
```

This gives us the prediction of the ARIMA model which will now be enhanced by modelling the ARIMA residuals with a Long Short-Term Memory neural network. As the Long Short-Term Memory neural network is not modelled on the original data the model does need to be optimized on the new data, which consists of the ARIMA residuals. To optimize the new Long Short-Term Memory neural network model with the ARIMA residual data I again use the "keras_tuner" package and the "build_model" function which are explained in the Long Short-Term Memory neural network optimization section above. From the Keras tuner results I create the following model consisting of a Long Short-Term Memory layer of 96 neurons and a Dense layer of 4 neurons. I also found that for this model a batch size of 16 gave more accurate forecasting results.

```
#Defining training parameters
verbose, epochs, batch_size = 1, 1000, 16

#Defining the model
model = Sequential()
model.add(LSTM(units = 96 ,return_sequences = False ,   activa-
tion='relu'))
model.add(Dense(4))
model.compile(loss='mse', optimizer='adam')

# fit network
es = EarlyStopping(monitor='val_loss', patience=20, re-
store_best_weights=True)

model.fit(X_df_train, Y_df_train, epochs=epochs, batch_size=batch_size,
verbose=verbose, validation_data = (X_df_val, Y_df_val), callbacks = [es])
```

The prediction of the ARIMA model combined with the residual prediction of the Long Short-Term Memory neural network model will give the final hybrid model prediction.

6. Analysis

6.1 Results

To compare the different models I use the root mean squared error as a measure of accuracy (Siami-Namini, Tavakoli, & Namin, 2018). The root mean squared error is calculated based on the 32 test weeks of which the predicted data per model is compared to the actual values. A lower root mean squared error means a more accurate model as the error is lower. The results of the root mean squared error calculation are in the table 1 below. Another measure of accuracy I use is the mean absolute error. Similarly to the root mean squared error the model accuracy is better when the mean absolute error has a lower value.

The root mean squared error is calculated using the following formula (Chai & Draxler, 2014):

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}}$$

In which N is the total number of observations, $Predicted_i$ is the predicted value for observation i and $Actual_i$ is the actual value for observation i.

The mean absolute error is calculated as follows:

$$MAE = \frac{\sum_{i=1}^{N}|Predicted_i - Actual_i|}{N}$$

In which N is the total number of observations, $Predicted_i$ is the predicted value for observation i and $Actual_i$ is the actual value for observation i.

I applied the four models on the four different datasets and calculated the root mean squared error and the mean absolute error. These results can be found in the following table.

Table 2: Model results

Dataset	Model	RMSE	MAE
Linear	SES	0.00824	0.00661
Linear	ARIMA	0.00001	0.00001
Linear	LSTM	0.00029	0.00025
Linear	Hybrid	0.00001	0.00000
Yearly	SES	0.10937	0.09112
Yearly	ARIMA	0.00503	0.00401
Yearly	LSTM	0.00921	0.00796
Yearly	Hybrid	0.00501	0.00397
Monthly	SES	0.14963	0.12658
Monthly	ARIMA	0.00827	0.00666
Monthly	LSTM	0.01495	0.01212
Monthly	Hybrid	0.00778	0.00635
Company	SES	0.12338	0.09777
Company	ARIMA	0.09501	0.08289
Company	LSTM	0.10580	0.08575
Company	Hybrid	0.09434	0.08165

[2]

In the table the root mean squared error and the mean absolute error are listed per model per dataset. The lower the root mean squared error and mean absolute error the better the performance of the model is on the test set. The Long Short-Term Memory neural network models used to calculate the results are selected based on their root mean squared error on the validation set. The selected models were then used to calculate the root mean squared error and mean absolute error of their prediction on the test period.

[2] In the table SES stands for Simple Exponential Smoothing, LSTM stands for Long Short-Term Memory neural network, Hybrid is the hybrid ARIMA and Long Short-Term Memory neural network model, RMSE stands for root mean squared error and MAE stands for mean absolute error. The root mean squared error and means absolute error are calculated on the same period per dataset. The results in this table are calculated using the test set of each dataset and the used models are selected based on their performance on the validation set of each dataset.

On the synthetic datasets the performance of the Simple Exponential Smoothing model is for all three datasets the least accurate. For the synthetic datasets the ARIMA model also outperforms the Long Short-Term Memory neural network model. Finally, for all three datasets the hybrid model has the most accurate results based on the root means squared error and the mean absolute error.

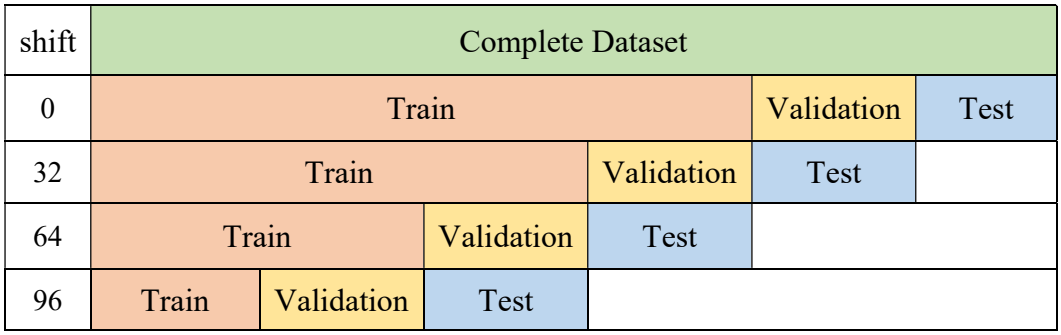
These results can also be seen for the company dataset prediction. For the company dataset of the cash flow of the accounts receivable the Simple Exponential Smoothing model is the least accurate, ARIMA outperforms the Long Short-Term Memory neural network and finally the hybrid model is the most accurate model based on the root mean squared error and the mean absolute error.

In this analysis the Simple Exponential Smoothing model was meant to be a baseline model, because it consists of a moving average model which is best used for data with no clear trend or pattern. Therefore, it was expected that the Simple Exponential Smoothing model would give the least accurate forecast. This is the case because for example with the linear data the Simple Exponential Smoothing model does not recognize the linear increasing pattern within the data and thus for each forecasted week ahead the forecast will lack behind the data. This is not the case for the ARIMA, Long Short-Term Memory neural network and Hybrid models, because those models are able to recognize the trend or pattern within the data.

6.2 Cross-validation

The performance of the models on the company data showed that for that specific dataset and time period the hybrid model worked best, but to see whether those results are robust I use cross-validation. For the cross validation I have shifted the training and test period back three times. The results from this cross-validation test will show whether the results that are show above are consistent. The below figure shows how cross-validation for time series data works.

Figure 5: Time series cross-validation



Cross-validation with time series data consists of shifting the training and test period forward to test whether the model results will stay the same also for other training and test periods. In the table below the results of these cross-validation tests are shown. I shifted the data three times where each shift is 32 weeks.

Table 3: Cross-validation results of the company dataset

Data Shift	Model	RMSE	MAE
0	SES	0.12338	0.09777
0	ARIMA	0.09501	0.08289
0	LSTM	0.10580	0.08575
0	Hybrid	0.09434	0.08165
32	SES	0.07646	0.06188
32	ARIMA	0.07808	0.06190
32	LSTM	0.55767	0.35326
32	Hybrid	0.07823	0.06121
64	SES	0.36915	0.24655
64	ARIMA	0.36589	0.26938
64	LSTM	0.36053	0.23883
64	Hybrid	0.36668	0.26813
96	SES	0.27013	0.20726
96	ARIMA	0.19604	0.15965
96	LSTM	0.19557	0.14519
96	Hybrid	0.19625	0.15918

[3]

[3] The data shift column is the number of weeks at the end of the dataset which are dropped. Since the test period consists of 32 weeks, the step in which the data shifts is also 32 weeks. In the table SES stands for Simple Exponential Smoothing, LSTM stands for Long Short-Term Memory

The cross validation of performance of the models shows that not one model is consistently the most accurate. For the first period, which is the same as the original test period, the hybrid model performs best based on both the root mean squared error and the mean absolute error. For the second period, which is based on the data without the last 32 weeks, the Simple Exponential Smoothing model performs best based on the root mean squared error, but the hybrid model performs best based on the mean absolute error. For the third period, which is based on the data without the last 64 weeks, the Long Short-Term Memory neural network model gives the most accurate results based on the root mean squared error and the mean absolute error. Finally, for the fourth period, which is based on the data without the last 96 weeks, the Long Short-Term Memory neural network model gives the most accurate result based on the root mean squared error and the mean absolute error.

The inconsistency of the model performance can be explained by the volatile cash flow of the accounts receivable dataset. It shows how well each model can manage outliers in the data and since the cash flow of the accounts receivables of this company is quite volatile it does contain different outliers per test period. The results also show that often model performance is quite similar, especially the ARIMA and the hybrid model. Since the hybrid model base prediction is the ARIMA prediction the small difference shows that the added error prediction by the Long Short-Term Memory neural network does not significantly improve the accuracy of the prediction.

6.3 Model performance graphs

The following graphs show the model performance based on how far the prediction is from the actual values. Meaning that if a value is 0.2 then the

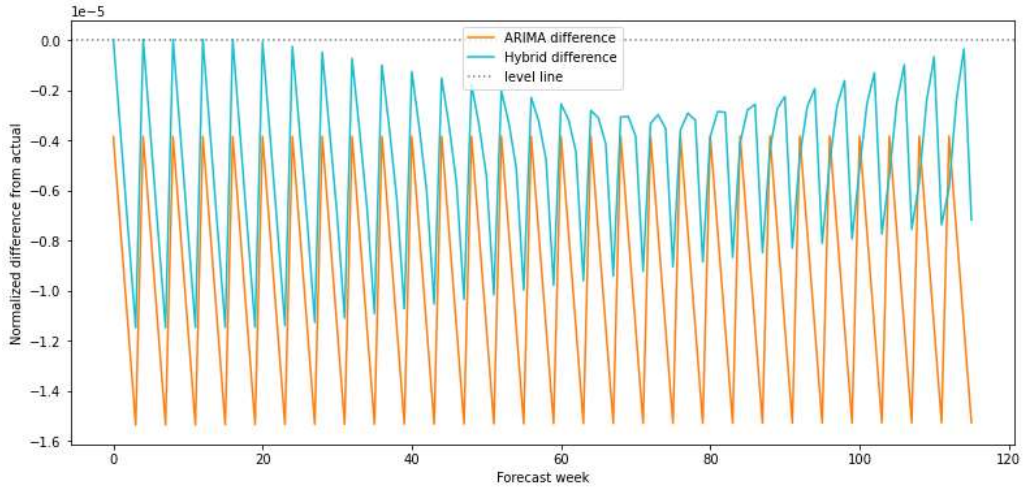
neural network, Hybrid is the hybrid ARIMA and Long Short-Term Memory neural network model, RMSE stands for root mean squared error and MAE stands for mean absolute error. The models used to calculate these results have the same parameters as the models used to calculate the original results.

prediction was 0.2 higher than the actual value. The closer the model difference line is to the level line the better the model performance.

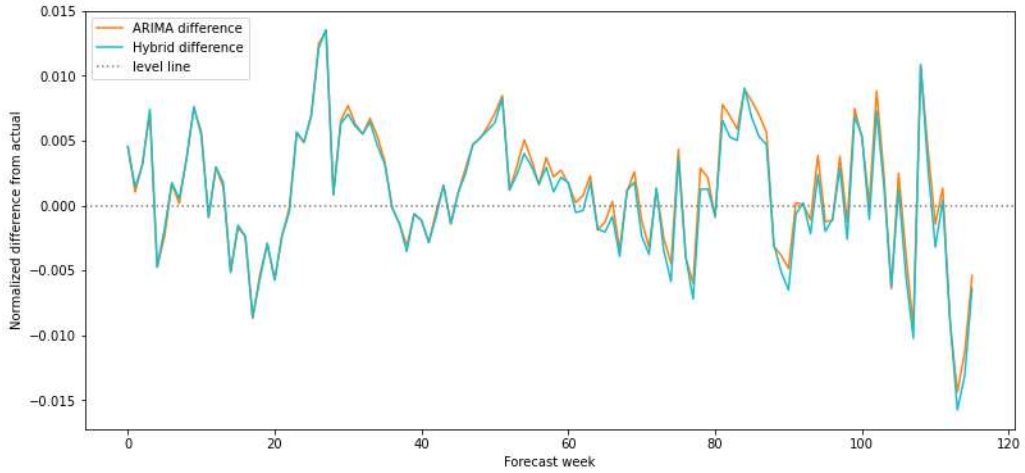
$$Difference = model\ prediction - actual$$

The following graphs show the forecasted weeks as one long list, such that all the four-week forecasts per week are reshaped into one long list. The graphs below only show the more accurate models to keep the model performance between the more accurate models visible, for all difference graphs see Appendix 1.1.

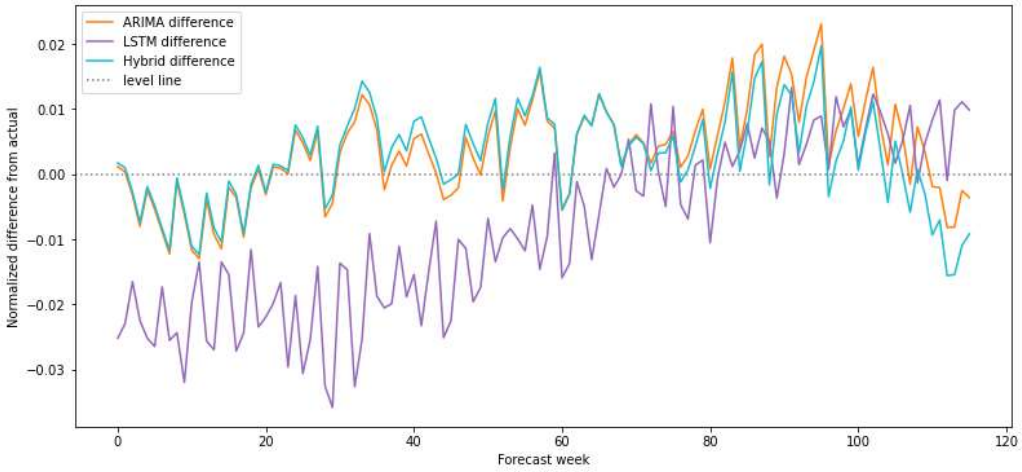
Graph 3: Linear dataset difference graph ARIMA, Hybrid



Graph 4: Yearly dataset difference graph ARIMA, Hybrid



Graph 5: Monthly dataset difference graph ARIMA, LSTM, Hybrid



6.4 Four week forecast model performance

Each model gives a forecast per week of the next four weeks. To analyze whether the models decrease in accuracy the further ahead they forecast into the future I calculated the root mean squared error and the root mean absolute error per model per forecast week for all four datasets. In the below table the results of this analysis are shown for the synthetic datasets sorted per model. This shows how much accuracy each model loses the further ahead the forecast is. When the root mean squared error or the mean absolute error increase with the number of weeks ahead the forecast is it means that the model loses accuracy the further ahead into the future the forecast is. For most models a loss in accuracy is to be expected when a model forecasts further into the future, this is because data will become less reliable and more shocks can enter the data. However, for the synthetic datasets this is not the case as the patterns and trend in those datasets stay the same over time. The extent to which each model can learn these patterns in this case influence the accuracy of the models. For example, the Simple Exponential Smoothing model uses a moving average model and will thus not use patterns or trends in the data. For the Simple Exponential Smoothing model it is thus to be expected that it becomes less accurate the further into the future it forecasts, but the results show whether this is also the case for the models which are able to learn patterns and trends in the data.

Table 4: Model performance per forecast week on the synthetic datasets

	Dataset:	Linear		Yearly		Monthly-Yearly	
Week	Model	RMSE	MAE	RMSE	MAE	RMSE	MAE
1	SES	0.004405	0.004405	0.0409	0.0371	0.1363	0.1163
2	SES	0.008811	0.008811	0.0811	0.0736	0.1650	0.1391
3	SES	0.013216	0.013216	0.1203	0.1094	0.1401	0.1147
4	SES	0.017621	0.017621	0.1585	0.1444	0.1554	0.1362
1	ARIMA	0.000004	0.000004	0.0045	0.0035	0.0051	0.0043
2	ARIMA	0.000008	0.000008	0.0052	0.0041	0.0080	0.0068
3	ARIMA	0.000011	0.000011	0.0054	0.0044	0.0093	0.0077
4	ARIMA	0.000015	0.000015	0.0051	0.0041	0.0098	0.0079
1	LSTM	0.000341	0.000319	0.1228	0.1025	0.0168	0.0133
2	LSTM	0.000145	0.000135	0.1270	0.1049	0.0175	0.0143
3	LSTM	0.000432	0.000405	0.1286	0.1071	0.0110	0.0093
4	LSTM	0.000143	0.000127	0.1224	0.1011	0.0135	0.0116
1	Hybrid	0.000003	0.000002	0.0045	0.0035	0.0054	0.0039
2	Hybrid	0.000003	0.000003	0.0052	0.0041	0.0074	0.0061
3	Hybrid	0.000005	0.000005	0.0053	0.0043	0.0084	0.0073
4	Hybrid	0.000010	0.000010	0.0050	0.0039	0.0093	0.0081

[4]

Both the root mean squared error and mean absolute error in the table show that the Simple Exponential Smoothing model loses accuracy the further ahead the model predicts. For both ARIMA and the hybrid model it is also the case that the further into the future the model predicts overall the less accurate the predictions become. For the Long Short-Term Memory neural network model there is no visible decrease in the model accuracy, but the model accuracy is also not stable. For all three datasets there is no clear increase or decrease in the root means squared error or the mean absolute error of the Long Short-Term Memory neural network model.

[4] The week column is the number of weeks ahead the forecast is. In the table SES stands for Simple Exponential Smoothing, LSTM stands for Long Short-Term Memory neural network, Hybrid is the hybrid ARIMA and Long Short-Term Memory neural network model, RMSE stands for root mean squared error and MAE stands for mean absolute error.

Table 5: Model performance per forecast week on the company dataset

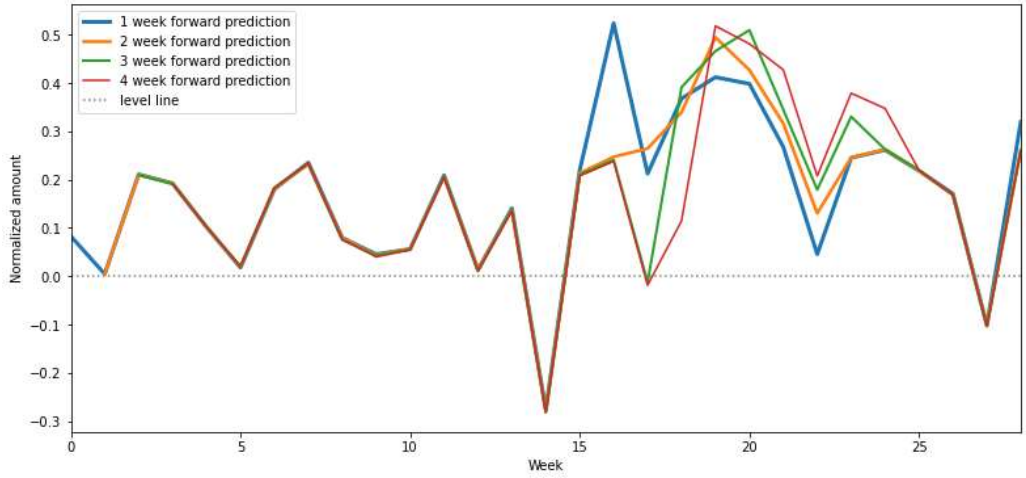
Week	Model	RMSE	MAE
1	SES	0.2320	0.1933
2	SES	0.2277	0.1958
3	SES	0.2392	0.2037
4	SES	0.2434	0.2045
1	ARIMA	0.0967	0.0824
2	ARIMA	0.0951	0.0864
3	ARIMA	0.0966	0.0843
4	ARIMA	0.0915	0.0785
1	LSTM	0.1249	0.0915
2	LSTM	0.1263	0.0933
3	LSTM	0.1285	0.0976
4	LSTM	0.1388	0.1061
1	Hybrid	0.0961	0.0816
2	Hybrid	0.0948	0.0854
3	Hybrid	0.0947	0.0822
4	Hybrid	0.0916	0.0773

[5]

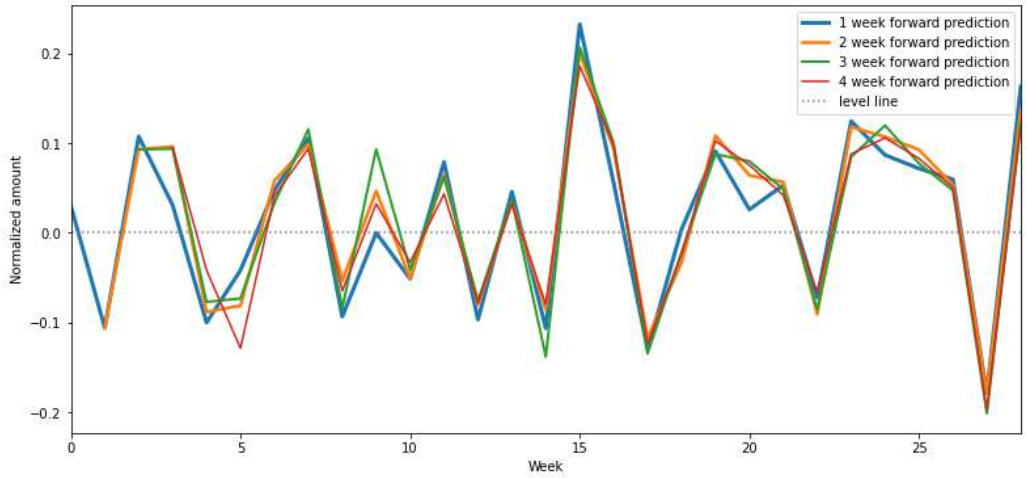
The results for the company dataset in the above table show that overall the Simple Exponential Smoothing model and the Long Short-Term Memory neural network model lose accuracy the further ahead they forecast into the future. Both the ARIMA and the Hybrid model overall do not decrease much in accuracy the further they predict into the future. To make this visible below are difference graphs per model. The closer the lines are to the 0 line the more accurate the prediction is.

[5] The week column is the number of weeks ahead the forecast is. In the table SES stands for Simple Exponential Smoothing, LSTM stands for Long Short-Term Memory neural network, Hybrid is the hybrid ARIMA and Long Short-Term Memory neural network model, RMSE stands for root mean squared error and MAE stands for mean absolute error.

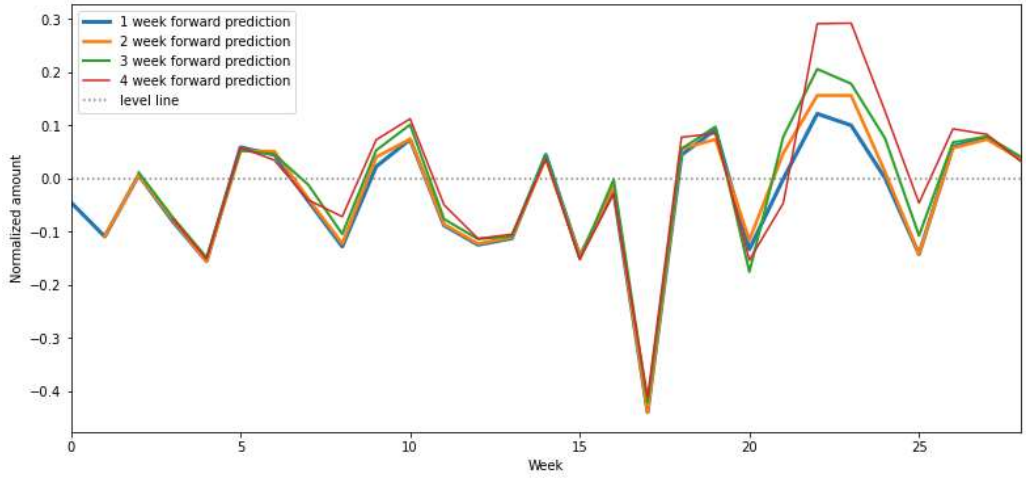
Graph 6: SES difference graph per forecast week on the company dataset



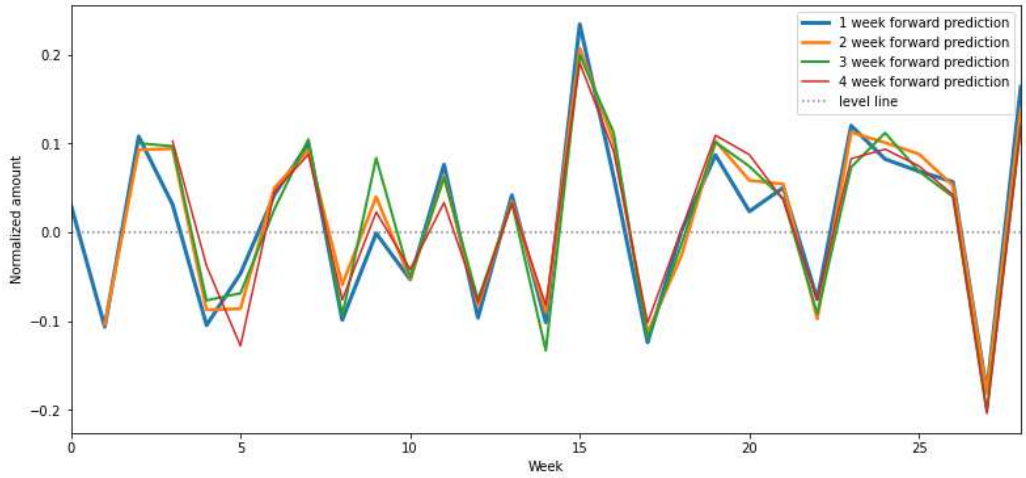
Graph 7: ARIMA difference graph per forecast week on the company dataset



Graph 8: LSTM difference graph per forecast week on the company dataset



Graph 9: Hybrid difference graph per forecast week on the company dataset



In the graphs it is visible that for the company dataset the Simple Exponential Smoothing and Long Short-Term Memory neural network lose accuracy the further into the future they predict. This is especially visible in graph 8 in which there is a peak around week 22 in which the one week ahead prediction is much more accurate than the four week ahead prediction. For the Simple Exponential Smoothing model it is a different case as it uses a moving average meaning that the accuracy of the Simple Exponential Smoothing model is mostly influenced by shocks in the data. For both the ARIMA and the Hybrid model there is less visible decreased accuracy the further into the future the model predicts.

7. Research limitations

7.1 ARIMA limitations

A limitation of the ARIMA model is the change in the data of a rolling forecast model. As time changes the data can also change, for example the structure can change or customer payment behavior can change. This can alter the data such that the most optimal model parameters change. Because of this it is also necessary to check whether the auto_arima function reports altered optimal model parameters as the data changes to make sure that the ARIMA model predictions stay as accurate as possible.

7.2 Long Short-Term Memory neural network limitations

The performance of the Long Short-Term Memory neural network model that is observed from the analysis is because the neural network model is able to train and improve itself over many iterations. The limitation this process poses is that over multiple times training the model the model accuracy is not constant. The accuracy varies as each time the model is trained it learns differently. Because of this varying accuracy between trained models I have trained multiple models and selected the model with the highest accuracy on the validation set measured by the root mean squared error. This extra selection step makes the Long Short-Term Memory neural network model more complicated than for example the ARIMA model which does not have a varying accuracy.

For the Long Short-Term Memory neural network it is also necessary to spit the dataset into train, validation and test sets. The model is trained on the train set and then optimized using the early stopper function, which uses the validation set. Then the best model is selected using the validation set. As the validation set consists of a number of the last weeks these last weeks are not used in training the model. This limits the Long Short-Term Memory neural network model as it is not trained using the most recent data. Due to this it is possible that the most

recent patterns of the data are not recognized by the model, which will result in a less accurate prediction.

Another limitation for using neural network models in business practice is that the prediction outcome of a neural network model comes out of a “black box” where no exact calculation can be found or traced (Atsalakis, 2016). For a model to be used in business practice there must be a certain amount of evidence and transparency. This is mostly because the people who make decisions using the forecast need the reasoning behind the forecast value to back up their decisions. This is also mentioned in an article written by Randy Bean and Thomas H. Davenport in the Harvard Business Review analytics and data science website. In this article it is said that in their survey 93% of the people saw people and process issues instead of technical limitations as the obstacle for companies not adopting data driven solutions (Bean & Davenport, 2019).

7.3 Hybrid ARIMA Long Short-Term Memory neural network limitations

As the hybrid model consists of an ARIMA model combined with a Long Short-Term Memory neural network model it is limited by both the limitations of the ARIMA model and the Long Short-Term Memory neural network model. On top of the listed limitations above the hybrid model is also more complicated. The results also show that the hybrid model does not significantly increase the accuracy compared to the normal ARIMA model. For the monthly dataset the ARIMA root mean squared error and mean absolute error are 0.00827 and 0.00666 respectively, the hybrid root mean squared error and mean absolute error are 0.00778 and 0.00635 respectively. This is a decrease of 0.00049 for the root mean squared error and 0.00031 for the mean absolute error. Also for the company dataset the increased accuracy was minimal. As for the company dataset the ARIMA root mean squared error and mean absolute error are 0.09501 and 0.08289 respectively, the hybrid root mean squared error and mean absolute

error are 0.09434 and 0.08165 respectively. This is a decrease of 0.00067 for the root mean squared error and 0.00124 for the mean absolute error. During the cross validation the results also showed that the hybrid model can perform worse than the ARIMA model, meaning that the added Long Short-Term Memory neural network decreased the forecasting accuracy of the model. This small increase or even decrease of the forecasting accuracy should be considered when deciding which forecasting model a company will be using.

7.4 Recommendations for further research

This study uses three synthetic datasets and one company dataset. The results from the cross validation show that there is not one best model to forecast cash flow of the accounts receivable based on historical data at a company which has accounts receivable that consists of few high value transactions and many small value transactions. To see whether this is also the case at other companies with the same structure of accounts receivable and other companies with a different structure of the accounts receivable further research is needed using datasets from multiple different companies.

The Long Short-Term Memory neural network model used in this study uses the last four weeks and the four forecast weeks with a one-year lag as input. Further research can investigate other Long Short-Term Memory neural network input data, such as adding weeks with a quarterly lag into the input data. Another possibility is using more than the four previous weeks as input data. Further research can thus investigate different shapes and values of the input data of the Long Short-Term Memory neural network model to research whether this will result in better forecasts.

This study focused on the performance of models which use historical data. Since cash flow data consists of transactions it is often the case that companies have a database with all planned transactions or the planned payment date of accounts receivable. By combining historical pattern recognition with these planned

transactions it can be possible to create a model which is more accurate than models which are based on only historical data. Further research is needed to see which models are best in combining historical data with planned transaction data.

Another way which more accurate models can be created is by splitting up the data before the use of forecasting models. For example when a company sells different products and services it is possible to split the data of certain products or services of from the main dataset. In this example a company sells products which need servicing on a regular basis which is also done by the company which sells the products. This company then has two main revenue streams, the revenue stream of products sales and the revenue stream of the servicing they provide. It can be the case that the revenue of the servicing is linearly increasing, while the product sales revenue has a yearly or monthly cyclic pattern. In this case it would be beneficial for model accuracy to split the data by revenue stream before using forecasting models. By first splitting the data the forecasting models can more easily recognize the patterns in the data. Further research can investigate the possibility of splitting data before using forecasting models at different companies to see whether this will result in more accurate models.

8. Conclusion

In this study I used four different models to predict the cash flow of the accounts receivable with a four-week window using historical weekly data. The models I compared were the Simple Exponential Smoothing model, the ARIMA model, the Long Short-Term Memory neural network model and a hybrid model consisting of both ARIMA and a Long Short-Term Memory neural network. All four of the models were optimized to give the best results. The Simple Exponential Smoothing model has one parameter as input which was optimized by the Simple Exponential Smoothing function itself. To optimize the ARIMA model I used the "auto_arima" function, which tries multiple forms of the ARIMA model to find the model parameters that give the best results. Finally, for the Long Short-Term Memory neural network model I used the Keras Tuner package which optimized the number of layers and number of neurons per layer by comparing the results of different forms of the model based on my parameters.

To test and optimize the models I first used three synthetic datasets. These synthetic datasets consisted of one with a linear increasing pattern, one with a yearly repeating pattern and finally a dataset consisting of a yearly combined with a monthly repeating pattern. For the synthetic datasets I conclude that the Simple Exponential Smoothing model was the least accurate. This was expected as the Simple Exponential Smoothing model performs best on data which has no trend or pattern, which is not the case in the synthetic datasets. The second least accurate model was the Long Short-Term Memory neural network model. The ARIMA and the hybrid model were the most accurate, but of those two the added error forecasting by the Long Short-Term Memory neural network in the hybrid model gave the most accurate forecast.

The accuracy of the models was measured using the root mean squared error and the mean absolute error. The results of the synthetic datasets showed that overall the hybrid model gave the most accurate forecast. However, from the cross validation on the company dataset I conclude that this is not the case for the

company data. The results of the cross validation on the company dataset showed that over the different test periods not one model gave the most accurate results.

To test the accuracy of the models per forecast week, I also compared the performance of the models based on the number of weeks they forecast into the future. As each model forecasts a four-week window I compared the root mean squared error and mean absolute error based per model per forecast week. The results of this test on the synthetic datasets showed that the Simple Exponential Smoothing model, the ARIMA model and the hybrid model decreased in accuracy the further into the future the models forecast. For the synthetic datasets the Long Short-Term Memory neural network had no clear increase or decrease in forecasting accuracy the further into the future the model forecast. The results of this test on the company dataset showed that the Simple Exponential Smoothing model and the Long Short-Term Memory neural network model both decreased in forecasting accuracy the further into the future the models forecast on the company dataset. For both the ARIMA and the hybrid model the forecasting accuracy on the company dataset stayed constant over the four forecast weeks.

The results from this study show that overall the ARIMA, the Long Short-Term Memory neural network and the hybrid models are more accurate than the baseline Simple Exponential Smoothing model. This shows that there are patterns within the data which are recognized by these models to give a more accurate prediction than the moving average prediction from the Simple Exponential Smoothing model. However, further research is needed to conclude whether the findings on this company dataset are also valid for other companies with the same structure of the cash flow of the accounts receivable. The findings in this paper cannot be generalized to other companies as each company has a different structure of the cash flow of the accounts receivable.

9. References

Abonazel, M. R., & Abd-Elftah, A. I. (2019). *Forecasting Egyptian GDP Using ARIMA Models.* Cairo University, Giza, Egypt: Institute of Statistical Studies and Research.

Adebbiyi, A. A., Adewumi, A. O., & Ayo, C. K. (2014). *Stock Price Prediction Using the ARIMA Model.* International Conference on Computer Modelling and Simulation.

Andrawis, R. R., Atiya, A. F., & El-Shisiny, H. (2011). Forecast combinations of computational intelligence and linear models for the NN5 time series forecasting competition. *International Journal of Forecasting 27*, 672-688.

Atsalakis, G. S. (2016). Using computational intelligence to forecast carbon prices. *Applied Soft Computing 43*, 107-116.

Babu, C. N., & Reddy, B. E. (2014). A moving-average filter based hybrid ARIMA–ANN model for forecasting time series data. *Applied Soft Computing 23*, 27-38.

Bean, R., & Davenport, T. H. (2019, February 5). *Harvard Business Review*. Retrieved from Companies Are Failing in Their Efforts to Become Data Driven: https://hbr.org/2019/02/companies-are-failing-in-their-efforts-to-become-data-driven

Box, G. E., Jenkins, G. M., & C., G. (2016). *Time Series Analysis: Forecasting and Control.* Hoboken, New Jersey: John Wiley & Sons.

Cao, J., Li, Z., & Li, J. (2019). Financial time series forecasting model based on CEEMDAN and LSTM. *Physica A 519*, 127-139.

Chai, T., & Draxler, R. R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)? - Arguments against avoiding RMSE in the literature. *Geosci. Model Dev., 7*, 1247–1250.

Cheng, M.-Y., Hoang, N.-D., & Wu, Y.-W. (2012). *Prediction of project cash flow using time dependent evoutionary LS-SVM inference model*. Retrieved from https://www.iaarc.org/publications/fulltext/Prediction_of_project_cash_flow_using_time-depended_evolutionary_LS-SVM_inference_model.pdf

Dadteev, K., Shchukin, B., & Nemeshaev, S. (2020). Using artificial intelligence technologies to predict cash flow. *Procedia Computer Science*, 169 264–268.

Garima Jain, D. B. (2013). *A Study of Time Series Models ARIMA and ETS* . India : Galgotias College of Engineering and Technology/Computer Science.

Greff, K., Srivastava, R. K., Koutn´ık, J., Steunebrink, B. R., & Schmidhuber, J. (2017). LSTM: A Search Space Odyssey. *TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS*.

Hossein Abbasimehr, M. S. (2020). *An optimized model using LSTM network for demand forecasting.* Tehran, Iran: Computer & Industrial Engineering.

Hu, W. (2016). Overdue Invoice Forecasting and Data Mining. *MASSACHUSETTS INSTITUTE OF TECHNOLOGY*.

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: Principles and Practice.* Melbourne, Australia: OTexts.

Introduction to ARIMA. (2022, 4 22). Retrieved from ARIMA models for time series forecasting: https://people.duke.edu/~rnau/411arim.htm

Ioannis E. Livieris, E. P. (2020). A CNN–LSTM model for gold price time-series forecasting . *Neural Computing and Applications*, 32:17351–17360 .

Khashei, M., & Bijari, M. (2011). A New Hybrid Methodology for Nonlinear Time Series Forecasting. *Modelling and Simulation in Engineering*.

Kingma, D. P., & Ba, J. L. (2015). ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. *ICLR*.

Lemke, C., & Gabrys, B. (2010). Meta-learning for time series forecasting and forecast combination. *Neurocomputing 73*, 2006-2016.

Namin, S. S., & Namin, A. S. (2018). *Forecasting Economic and Financial Time Series: Arima vs. LSTM.* Lubbock, TX, USA: Texas Tech University.

Navon, R. (1996). *COMPANY-LEVEL CASH-FLOW MANAGEMENT* . ASCE: JOURNAL OF CONSTRUCTION ENGINEERING AND MANAGEMENT.

Nikolenko, S. I. (2021). *Synthetic Data for Deep Learning.* San Francisco: Springer.

Oliveira, J. F., & Ludermir, T. B. (2016). A hybrid evolutionary decomposition system for time series forecasting. *Neurocomputing 180*, 27-34.

O'Malley, T. a., Long, E. a., Chollet, J. a., Jin, F. a., Invernizzi, H. a., & others, L. a. (2019). *KerasTuner*. Retrieved from Keras: \url{https://github.com/keras-team/keras-tuner

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

Parmezan, A. R., Souza, V. M., & Batista, G. E. (2019). Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model. *Information sciences 484*, 302-337.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *Proceedings of the 30th International Conference on Machine Learning*, PMLR 28(3):1310-1318.

Peiguang, H. (2013). Predicting and Improving Invoice-to-Cash Collection Through Machine Learning. *Engineering Science Programme National University of Singapore*.

PennState Eberly College of Science. (2022, 4 21). Retrieved from STAT 510 Applied Time Series Analysis: https://online.stat.psu.edu/stat510/lesson/14/14.1

Pitkänen, A. (2016). *Cash Flow Forecasting Proposal for New Long-Term Cash Flow Forecast in the Case Company.* Helsinki: Helsinki Metropolia University of Applied Sciences .

Polak, P., Nelischer, C., Guo, H., & Robertson, D. C. (2020). "Intelligent" finance and treasury management: what we can expect. *AI & SOCIETY*, 35:715–726.

Polak, P., Robertson, D. C., & Lind, M. (2011). The New Role of the Corporate Treasurer: Emerging Trends in Response to the Financial Crisis. *International Research Journal of Finance and Economics*, 1450-2887 Issue 78.

Raza, M. Q., Nadarajah, M., & Ekanayake, C. (2017). Demand forecast of PV integrated bioclimatic buildings using ensemble framework. *Applied Energy 208*, 1626-1638.

Roszkowska, P., & Prorokowski, L. (2017). THE CHANGING ROLE OF A BANK'S TREASURY.

Shira, D. (2019, July 31). *Profit Repatriation from China*. Retrieved from China Briefing: https://www.china-briefing.com/news/profit-repatriation-from-china/

Siami-Namini, S., Tavakoli, N., & Namin, A. S. (2018). A Comparison of ARIMA and LSTM in Forecasting Time Series. *Department of Computer Science*.

Tangsucheeva, R., & Prabhu, V. (2014). Stochastic financial analyticsforcash flowforecasting. *Int. J.ProductionEconomics 158*, 65-76.

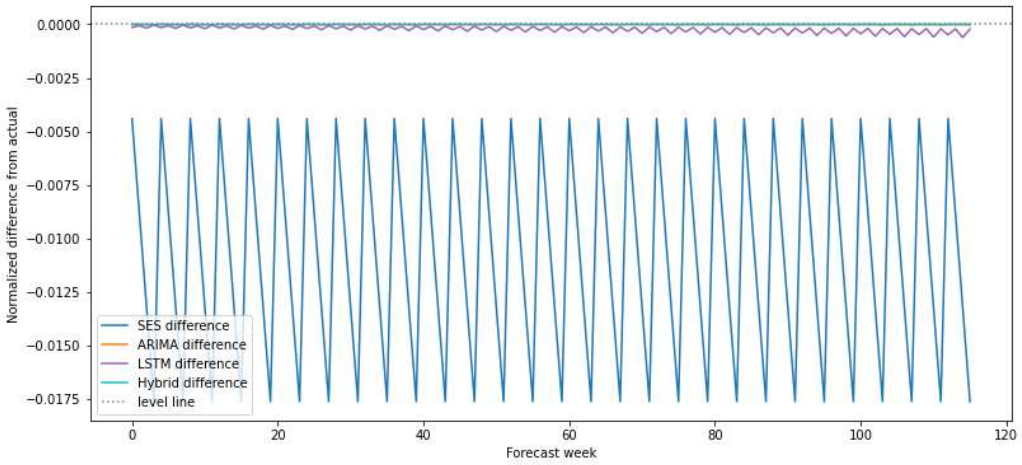
Weytjens, H., Lohmann , E., & Kleinsteuber, M. (2019). *Cash flow prediction: MLP and LSTM compared to ARIMA and Prophet.* Springer Science+Business Media.

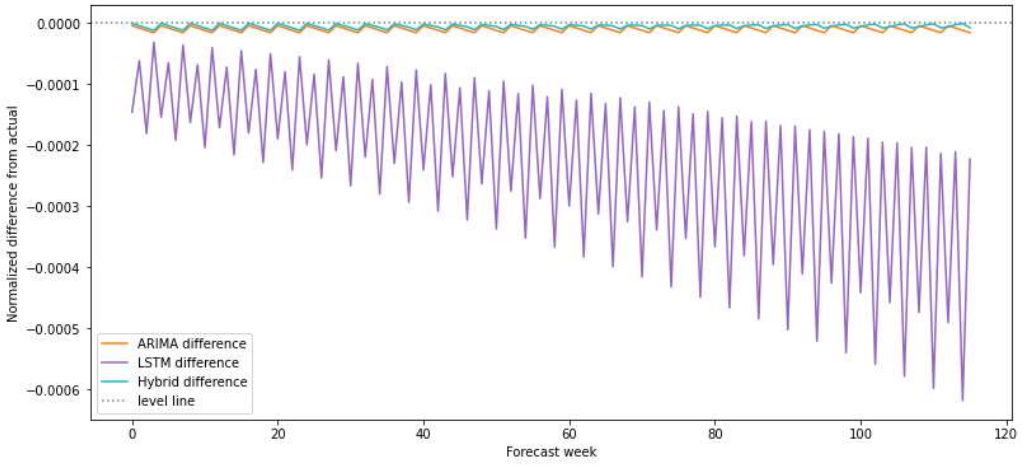
Zhang, G. P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing 50*, 159-175.

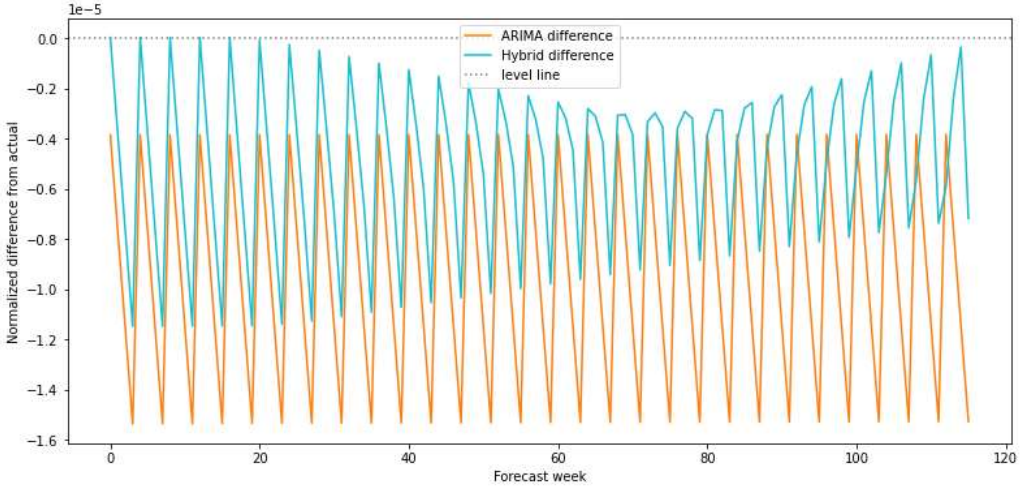
Appendix

1. Model difference graphs

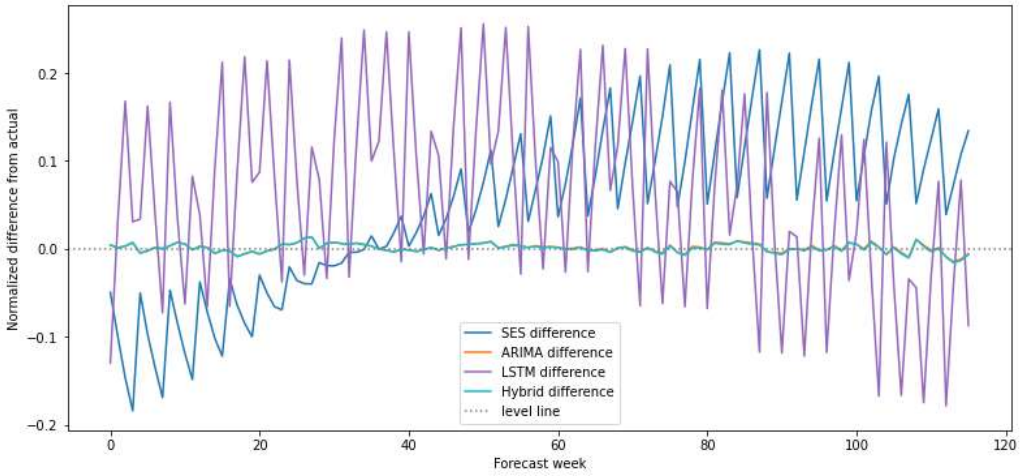
1.1 Linear difference graphs

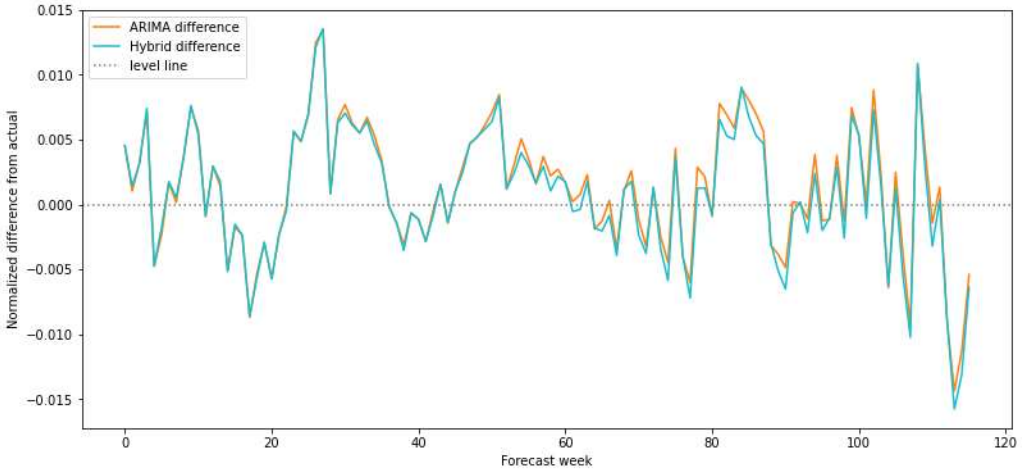




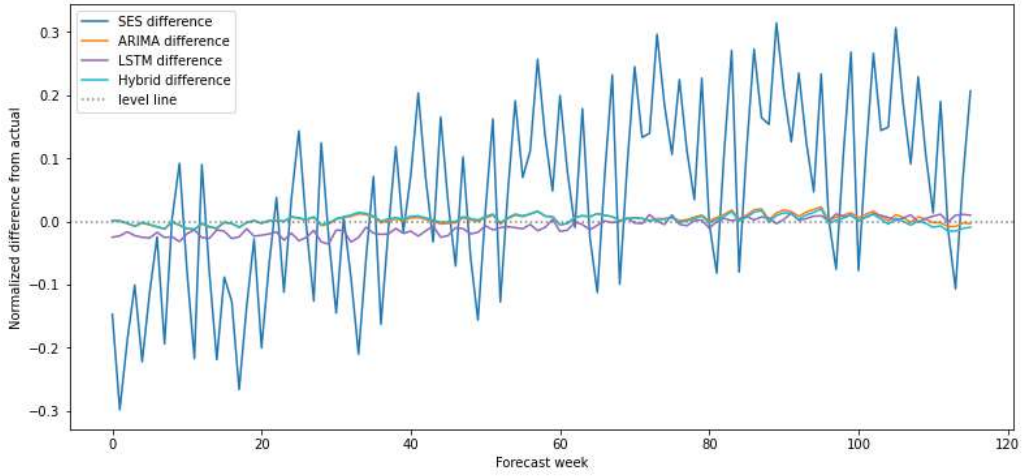


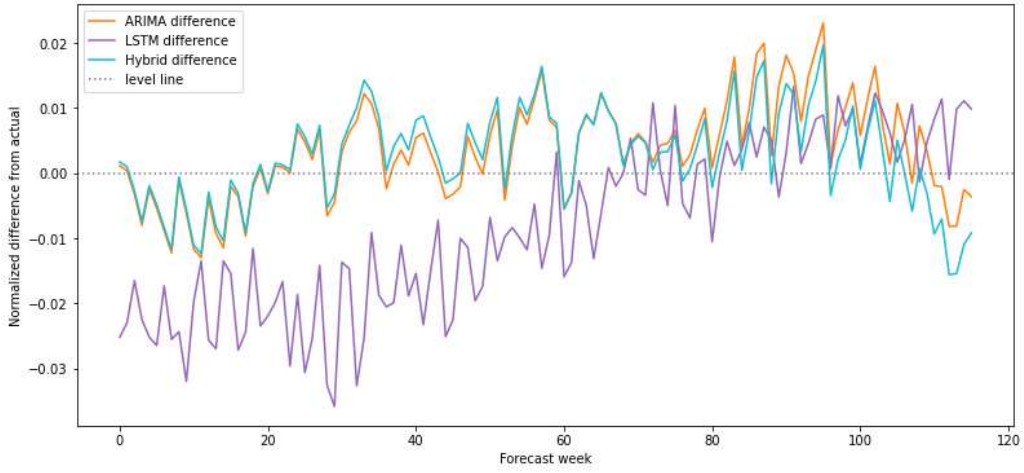
1.2 Yearly difference graphs

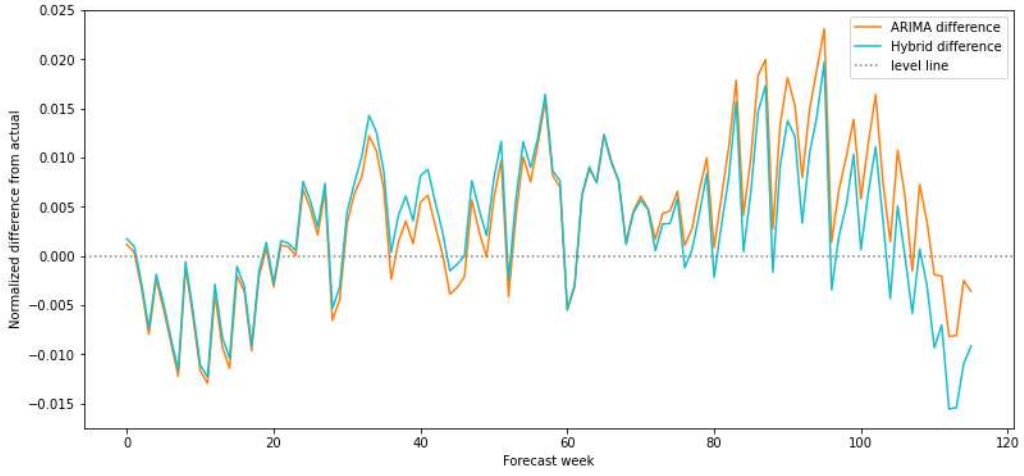




1.3 Monthly difference graphs







1.4 Company difference graphs