MSc Mathematics

Track: Data Science and Optimization

Master thesis

Generating synthetic bank balance data with machine learning techniques

by

Veerle Dinghs

February 17, 2023

Supervisor: dr. René Bekker

Second examiner: prof. dr. Sandjai Bhulai

Department of Mathematics
Faculty of Sciences

VU UNIVERSITY AMSTERDAM

Abstract

This thesis aims to research the possibilities of machine learning models to generate synthetic time series data with properties of bank balance data. Synthetic bank balance data is useful in banks for testing purposes without encountering privacy issues and for using the data to improve risk models. When using synthetic balance data for improving risk models, features of the balance data can be implemented per customer.

Synthetic data is in this thesis generated out of self-simulated time series data sets with three different models; the Probabilistic AutoRegressive (PAR) model, the Time Series Generative Adversarial Network (TimeGAN), and the DoppelGANger (DG) model. The performance of these models is compared using three types of metrics; statistical values, machine learning techniques, and visual metrics. The model capturing the correlations and features in the simulated data sets best is the DG model. When the randomness in the input data set is increased, the number of input customers in the DG model has to be increased as well to capture all patterns and to get a synthetic data set with the same distributions as the original one.

With the synthetic data set resulting from the DG model, an application is executed to answer the second research question on how to create representative features on this data set. Clustering on static data with K-Means clustering results in clusters with features recognizable from the input data. Clustering on the dynamic data with K-Means Time Series clustering needs more research since the K-Means clustering technique does not result clearly in the input features and gives differences in the original and synthetic data set clusters.

Keywords: Synthetic time series data; Neural Networks; K-Means Clustering; PAR; GAN; TimeGAN; DoppelGANger.

Title: Generating synthetic bank balance data with machine learning techniques
Author: Veerle Dinghs, v.dinghs@student.vu.nl, 2597738
Supervisor: dr. René Bekker
Second examiner: prof. dr. Sandjai Bhulai
Date: February 17, 2023

Department of Mathematics
VU University Amsterdam
de Boelelaan 1081, 1081 HV Amsterdam
`http://www.math.vu.nl/`

Contents

1. Introduction

Banks' databases are full of customer data gathered over the last decades. A large part of this data is not used in developing new products and calculating lending risk. For example, customer balance data, which is the amount of money in a bank account over time, is not used to develop credit risk models. Balance data has some difficulties when it would be used for modeling. For example, the data is very volatile over time, it can change multiple times a day, but it is also possible that it does not change on that day. Besides that, the data is confidential, customers will not just give their transaction data to others and the bank will need permission to share it. Despite these difficulties, balance data could in theory be beneficial for training purposes in banks. As an example, balance data is not used in (credit) risk models for predicting, one of these risk models is in predicting a probability of default. It is, however, interesting to investigate if these risk models will improve if features of balance data are added. Examples of these features could be the number of times the balance is negative or the time series volatility. Banks are careful to take the step of adding balance data into the risk models since it is time-consuming and confidential. However, instances such as the European Banking Authority (EBA) want current models to be improved [17]. Balance data could enrich existing models with features for predicting and it is interesting to find out if this indeed results in improvements.

A solution for the noisy and confidential balance data could be to generate synthetic balance data with a synthetic data generator (SDG). With this generator, one can anonymize the data set and still keep the same distributions as in the original data set. Synthetic data can be used in improving models since the underlying distribution of the features in the original data set stays the same and therefore both data sets are highly correlated [12]. Besides the potential usefulness of synthetic balance data to improve financial models, it is also useful for testing statistical properties on the banks' portfolio without asking and waiting for permission. This can speed up preliminary data investigation and thus eventually speed up the process of creating a model.

This thesis aims to investigate the construction of synthetic financial bank balance data with machine learning techniques. Moreover, the synthetic data set is used in an application where representative features for the risk models are generated. To generate synthetic balance data, an input balance data set is needed. Since data in banks is very confidential, the models for this thesis have simulated data sets with properties of bank balance data as input. An advantage of simulating the data is that the distributions in the data are known, with this knowledge the underlying distributions in the synthetic data can be verified on these distributions and a comparison with the input data set can be made more easily. The simulated data represents the daily balance amount over time with a corresponding number of transactions per day. The customers' age at the

beginning of the time series is added as a static feature. Several features, groups, and events which can be found in balance data (formed based on expert-based inputs) are added to this simulated data. A clearly visible event in balance data is the monthly seasonality of salary and fixed costs. Furthermore, a portfolio of customers exists of high-risk and low-risk customers and an event of an economic crisis can be detected out of the information in balance data.

Different types of models are useful for generating synthetic data, for instance generative adversarial networks (GANs) and auto-regressive models. The GANs are successful models in generating data, for example in image processing or 3D object generation [23]. These networks use machine learning techniques to improve random data to look as much as possible like the real input data. After further investigation into GANs, it turned out that with some algorithm adjustments, these networks are not only useful for recreating images but they can also be used for generating time series data. Reproducing time series data has extra difficulties due to the noisy data, the correlations between values at different time steps, the long-term persistence, and periodicity [15]. New techniques emerged where GANs are combined with existing models that represent time series and where GANs are adjusted so they can handle long-term correlations, for example the TimeGAN and DoppelGANger models. The auto-regressive models create successive time series values based on previous values and predict the sequel of a time series with this technique. An example of these models is the PAR model which is just as time series GANs successful in generating synthetic time series data [2].

The outputs of the different machine learning models on simulated time series are synthetic data sets. Metrics can evaluate these output data sets to compare them to the input data set. Evaluation can be done with different techniques; statistical properties and measures, machine learning metrics, and visual metrics. Machine learning metrics give a score on how well the real data and the synthetic data are separable from each other by training neural nets on the data sets. Ideally, the real and synthetic data sets are hard to distinguish. Visual metrics reduce the dimensionality of the real and synthetic data, these data sets can be checked on overlap after plotting with reduced dimensionality. Based on these evaluation metrics, the question of which model performs best in generating time series data with properties of bank balance data can be answered, which is the main research question. The best-performing model captures correlations and distributions best and is therefore the most promising.

The synthetic data sets resulting from the machine learning models can be used in banking for testing purposes or to improve financial risk models. For the consideration of taking balance data into account in financial models, the data has to be investigated concerning features, patterns, and groups. Possible features to add to financial models are the number of times the balance is negative and the volatility of the balance over time. Another option is to cluster the accounts, this gives insight into the various groups of the portfolio of clients. The group label received from clustering can be an indication of high-risk or low-risk clients and can be included as a feature in the model prediction. This process answers the secondary research question and is called the modeling part. When the modeling part results in the features and groups representative of the features and groups of the input data, these techniques can also be applied to real-world bank

balance data.

1.1. Research questions and approach

This thesis investigates the following research question

1. Which model performs best in generating time series data with properties of bank balance data?

After answering this research question, an application of synthetic data in banking is being investigated and answers the second question

2. How to create representative features on generated synthetic bank balance data?

In order to answer these research questions, a literature review is done in Chapter 2. Chapter 3 describes the characteristics of the different simulated data sets. Simplified data sets and a more advanced data set are simulated. The goal is to generate a synthetic bank balance data set with the simulated advanced data set. Nevertheless, three different synthetic data generating models are tested on simplified data sets before generating. The three synthetic data generating models are PAR, TimeGAN, and DG. These models use different types of neural networks, which are explained in Chapter 4. With this knowledge, the three synthetic data generating models can be explained, which is done in Chapter 5. The synthetic results from these simplified data sets are compared with the real data with metrics, the metrics are explained in Chapter 6. From these simplified data sets, information on the model performance and metric performance is obtained and elaborated on in Chapter 7. This information results in a model with corresponding hyper parameters useful to simulate data from the advanced data set. Also, a list of useful metrics to check the performance of the model is received from this information. The next step is to apply the best-performing synthetic data generator on the simulated advanced data set to get a synthetic data set. This synthetic data set is being searched for groups in the modeling part in Chapter 8. Finally, Chapter 9 concludes and summarises this thesis.

This structure is visualized in Figure 1.1, the basic and intermediate data sets are used to check the performance of the three machine learning models. The synthetic data sets resulting from these models are used to determine which metrics are suitable for comparing the synthetic data with the original data. Out of the results, the machine learning model with the best performance is used to generate a synthetic data set out of the advanced data set, which answers the first research question. On this synthetic data set and the advanced data set, clustering techniques are applied to answer the second research question.

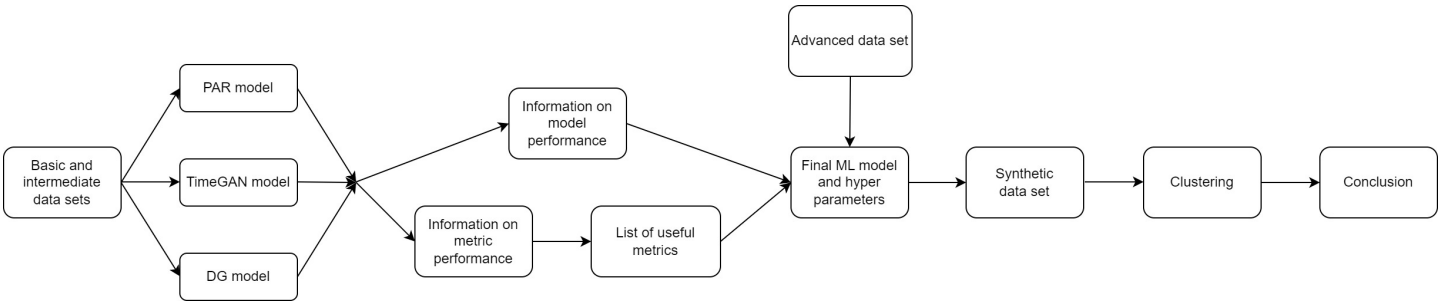


Figure 1.1.: Process of simulating synthetic data and modeling with the synthetic data

2. Literature Review

In the literature review, an overview of the current knowledge about synthetic data is analyzed and evaluated. In Section 2.1, the knowledge of synthetic data in general is discussed. Then in Section 2.2, the focus is limited to the generation of synthetic time series data. The second section provides the information necessary to answer the first research question. Finally, Section 2.3 describes the possibilities to model with the synthetic data which is needed to answer the second research question.

2.1. Synthetic data

Synthetic data is an upcoming field in data science where data is generated with machine learning techniques to gain extra data or gather missing data. It is emerging as an important element in building accurate machine learning models that can result in the correct predictions. Often the right ideas for a machine learning model are accessible but the needed data is not available, the data is biased towards a particular group, or it is private so it can not be used and shared. The synthetic data generation models are trained to keep the structure and correlations from the original data in the synthetic data [24]. Examples of fields where synthetic data is used more often are image processing where additional images can be constructed for training, health care where extra sequences of genomic data are made and detecting financial crime where synthetic fraud transactions are generated to have extra data since these are rare in the unbalanced data set [14]. Not only tabular data can be generated, but also time series data is interesting to generate to get synthetic data with the same trends and seasonalities. This data includes stock prices, sales over time, weather measurements, and heartbeats over time. Synthetic time series generation has extra difficulties because of the correlations between values at different time steps and the long-term persistence [15].

According to the European Data Protection Supervisor (EDPS), the positive impacts of synthetic data are enhancing privacy and improving fairness. Enhancing privacy is to protect personal data from being shared with third parties and improving fairness is to overcome racism and gender discrimination [18].

An ecosystem for synthetic data models is the Synthetic Data Vault (SDV) [37] which focuses mainly on tabular data but also on time series data. The SDV is introduced by Patki et al. [32] with as goal to create synthetic data with no significant differences from real data. The vault introduces techniques for generating synthetic tabular data like Gaussian Copula and AutoEncoder, and techniques for generating synthetic time series data like the Probabilistic AutoRegressive model.

2.2. Generating synthetic time series data

Various models that generate synthetic time series data can be found in the literature. The most relevant ones are described and are needed to answer the first research question.

The time series generation model described in the SDV is the Probabilistic AutoRegressive (PAR) model. This model uses the autoregressive structure of time series, namely a value at time t depending on the p previous time steps. The PAR model deals with all kinds of values (discrete, continuous, categorical) and involves static features in their model. The model is described by Zhang [50], where an experiment is applied to compare the PAR model with a synthetic data generation technique for non-time series data (CTGAN). The results of the PAR model on time series data improve compared to the results of the CTGAN model, this means that synthetic time series generation can be improved by using the autoregressive structure of time series compared to a model that does not use this structure.

Other synthetic time series generation techniques use the structure of Generative Adversarial Networks (GANs). The first paper on GANs is published by Goodfellow et al. [23], they combine the existing discriminative model and generative model into one adversarial nets framework. The authors state that the discriminative models are involved in the best successes in deep learning but that the generative models had less impact since they give difficulties in approximating probabilistic computations. The new adversarial model tackles these difficulties. This paper does not focus on time series data but mostly on generating images, audio waveforms, and symbols. The results show that the GAN models have good potential to generate synthetic data.

Research focusing on applying GANs on time series data is published by Yoon et al. [49], where the GAN model is combined with an autoregressive (AR) model and is called TimeGAN. In this new technique, the flexibility of the GAN model is combined with the autoregressive models which can capture the distribution of a time series well. In this paper, the TimeGAN model is compared with other GAN models and gives the best results in generating synthetic time series data, making it an adequate technique for generating synthetic bank balance data.

Another paper that focuses on applying GANs to time series data is published by Lin et al. [27], which focuses on long-term dependencies and multidimensional relationships in a model called DoppelGANger (DG). The problem, mentioned by the authors, of the GAN method is that in a long time series, the long-term correlations fail to be captured (e.g., annual correlations). In the DG model, this is resolved using RNN batches where the previous time step patterns are incorporated by generating the current time step. When comparing the DG model with other (time series) data-generating techniques (e.g., GAN, AR) on Google Cluster Usage Traces, the paper concludes that DG having the best results. This conclusion confirms that the DG model is sufficient to generate synthetic bank balance data.

While researching the different generation methods, an article in which the authors compare the PAR method with the GAN method was found [29]. The authors of this paper applied the DG, PAR, and TimeGAN models on time series wireless network traffic data and compared the generated time series results with each other. All three

models result in synthetic time series data with the same course as the real data, but not all equally good. The authors of this article conclude that the GAN-based models for generating time series data are superior in performance over the PAR model. However, PAR can be optimized since it was still under construction and the number of training steps can be increased.

The previously described papers do not have any relation to financial time series data. This can make a difference in results since financial time series data have differences in dependencies between time points and the data can depend on macroeconomic variables. The article from Takahashi et al. [40] investigates another kind of GAN model, namely the FIN-GAN model. The paper focuses mainly on stock markets but also mentions the usefulness of generating transaction data for fraud detection problems. However, no clear explanation architecture could be found and no proof that this method indeed works better than the other time series generating methods. This is why FIN-GAN is not tested in this thesis.

2.3. Modeling with the synthetic data

For the modeling part of this thesis, the focus is on investigating different clusters in the generated synthetic data. The goal is to find groups in the synthetic data with the same features that can be given as input in the development of a risk model for banks. The clustering method and the results received from this method can answer the second research question.

Finding groups in the large time series data set can be done with various clustering techniques, Denyse writes on this in an article [13]. The suggested method in this article is *Time Series K-means clustering* [42], which is a method that aims to partition the time series data in K clusters. This method is the easiest and most fast one, when having more computing time, more complicated methods can be used, which can handle the difficulties of clustering time series better. Examples of difficulties here are the different features at different time scales, multi-dimensionality, and high-frequency noise. These difficulties are discussed in an article by A. Alqahtani in Deep Time-Series Clustering [1], their solution is to use Deep Time-Series Clustering (DTSC) where clusters are made in the latent space with promising results. However, this method has a large computation time for large data sets, which is out of scope in this thesis.

The group labels received with clustering can be used as features in risk models, but also other features from time series data can be selected. Finding these features is a time-consuming process when the data set is large, Braun [7] writes on this in an article on Towards Data Science. The suggested method in this article is *tsfresh* [41], which is an automated feature extraction library for time series data. The input is a data frame with time series with corresponding ids, the outputs are different features with values per id. The features range from the easier ones like the minimum and maximum to the more difficult ones like trend, skewness, and autocorrelation. This method can give up to hundreds of different features per id, this is why the *tsfresh* method is probably not the most convenient method to use for feature selection in risk models for banks.

3. Characteristics bank balance data

The data chapter gives an impression of how the data sets are structured. All data sets in this thesis are simulated data sets where the time series have properties of bank balance data[1]. In this thesis, three different data sets are used, namely the *Basic data set*, the *Intermediate data set*, and the *Advanced data set*. The first two data sets are used for testing the models and metrics and the advanced data set is used to construct a synthetic bank balance data set for the modeling part of this thesis. See Table 3.1 for the customer properties per data set. In Section 3.1, the structure of the data set is explained. Then in Section 3.2, the simulation of the basic and intermediate data sets is described and at last, the simulation of the advanced data set is illustrated in Section 3.3.

Data set	**Properties of the customers**
Basic	Same trend, same seasonality moments.
Intermediate	Different trends, seasonality moments on different dates.
Advanced	Stable and unstable customers, economic crisis event, high randomness.

Table 3.1.: Overview of the three data sets

3.1. Structure data set

The three data sets are all built up in the same structure but they have numeric differences that result in differences in randomness. The structure of the data set is as in Table 3.2, where each line is the balance amount of a customer at the end of a day together with the number of transactions made that day. In the example of Table 3.2, n customers are present in the data. For each customer, there are 1827 data points representing 1827 days from 01/01/2016 to 31/12/2020. The customers' age at the start of the time series (01/01/2016) is included as a static feature since it can have some correlations with balance properties like start amount, salary height, and volatility. When adding age as a static feature, the models have an extra feature on which they can train and group. The data structure as in Table 3.2 is the input of one of the data generation models used in this thesis.

[1]See Chapter A.1 for an explanation of time series terms like trend and seasonality, which are used often in this chapter.

Account ID	Age	Date	Balance amount	Transaction count
a	29	01/01/2016	€1.208, 35	3
a	29	02/01/2016	€1.185, 12	1
a	29	03/01/2016	€1.185, 12	0
⋮	⋮	⋮	⋮	⋮
p	19	12/05/2017	€300, 55	5
p	19	13/05/2017	€288, 55	6
⋮	⋮	⋮	⋮	⋮
n	42	30/12/2020	€3.027, 18	3
n	42	31/12/2020	€2.896, 27	3

Table 3.2.: Structure of the input time series data in tabular form

However, two of the models need another input structure for training the data. The $2D$-matrix from Table 3.2 has to be converted to an $3D$-array. The three dimensions are the number of clients, the number of time steps, and the number of features and are visualized in Figure 3.1. In this visualization, the array is an $(n, T, 2)$-array with n clients, T time steps (in days), and 2 features, namely balance amount and number of transactions. The clients do not have a specific account ID anymore and the dates are also not defined, but the model can extract this information from the input dimensions. Therefore the time between two time steps all have to be equal, days in this case, which was not a requirement in the previous structure. The static feature age is implemented as a list of length n connected to the $3D$-array by the models.

3.2. Simulation basic and intermediate data sets

In the simulation of the data sets, properties of bank balance data are added to the balance amount time series. Typical properties of bank balance data are, for example, salary peaks once a month and fixed cost expenses once a month. The two data sets explained in this section are the basic and the intermediate data sets, as described in Table 3.1. The basic and intermediate data sets are used to get a better insight into the machine learning models and to check if the different models will capture the characteristics of the data sets. Next to that, the metrics are also checked on the synthetic data sets to figure out how these work and whether they give a good indication of the performance of this type of multivariate time series data.

Every time series in the data set represents a customer with a daily balance amount and a transaction count series from 2016 until 2020. The first data set is created with the seasonal peaks at the same moment and the same trend for all customers and is called *Basic data set*. The simulation is done in two steps, namely

1. a) Simulate for each customer a salary date around the 25^{th} of the month and a fixed costs date around the first of the month;

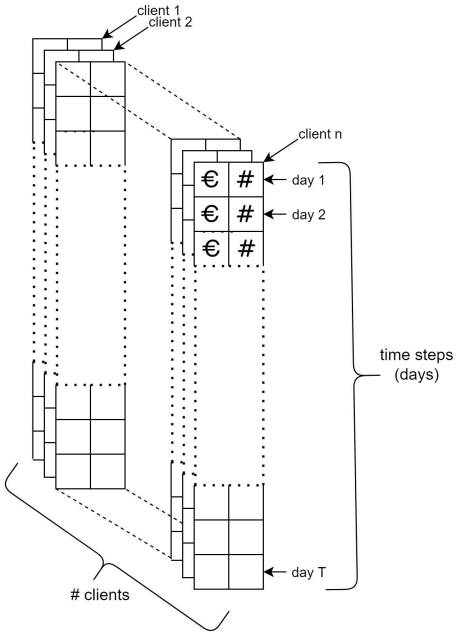


Figure 3.1.: Structure of the input data as 3D-array

b) Generate salary amount randomly between 1.000 and 4.000 and generate fixed costs amount between 80% and 90% of the salary.

2. Simulate the time series per customer. Call b_t the balance amount at time t, then $b_t = b_{t-1} + \text{amount}$, where *amount* depends on three situations.
   a) When it is salary day, $b_t = b_{t-1} + \text{salary}$;
   b) When it is fixed costs day, $b_t = b_{t-1} - \text{fixed\_costs}$;
   c) Else, a random number of transactions between 0 and 15 is simulated. For each of these i transactions, an amount between 1 and 100 is randomly drawn (c_i) from a beta distribution which is right-skewed when b_{t-1} is close to zero and left-skewed otherwise. Then $b_t = b_{t-1} - \sum_i c_i$, where 10% of the c_i are income transactions and the others expenses.

Since the total amount of transactions depends on the balance amount, all customers have the same trend. This trend is first decreasing and it stabilizes when reaching the amount of zero. In Figure 3.2, the sample paths of ten random customers of the basic data set are plotted for two years. As seen in this figure, the peaks overlap and the trends are the same for all customers. The original zero line is the zero line before normalizing, indicating a customer having a negative balance amount. It is important that the original zero line in the synthetic data set is around the same value as in the original data set. For a customer getting under this line means an overdraft which can be an extra risk when issuing a loan.

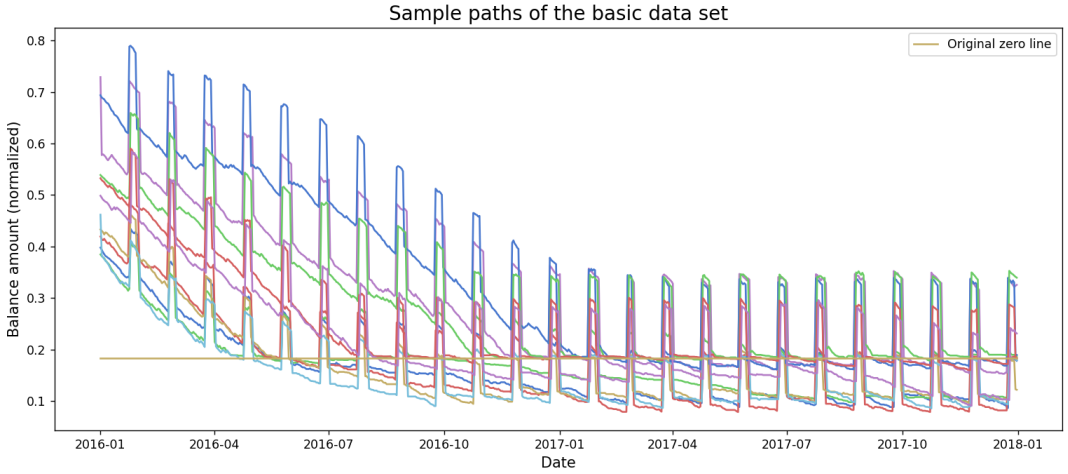


Figure 3.2.: Sample paths of ten customers of the basic data set where all customers have the same trend and seasonality moments

Since this is quite an easy pattern, extra randomness is added to the data to challenge the models more. When inserting more randomness in the data set, it will be more difficult for the data generation models to capture all the correlations and features. The extra randomness is added to the basic data set and is called the *Intermediate data set*. This simulation has the same structure as the simulation of the basic data set but with three differences:

1. The salary and fixed costs dates can be any moment in the month with a higher probability around the 25^{th} for salary and around the first for fixed costs. This will add randomness to the data since the seasonalities will no longer be at the same moments.

2. The salary is not for every customer close to the fixed costs anymore such that the decreasing trend at the beginning disappears. This is done because most customers will have a stable trend and only some of them a decreasing one. This will result in different trends in the data, so more randomness.

3. To add some extra randomness, customers can have a less stable job with the probability of losing their job and losing income. This results in customers who have no salary in a certain month, so the seasonality pattern will be interrupted at one point, making it harder for the models to find the pattern.

Again ten random sample paths of customers' balance amount from this data set are plotted in Figure 3.3.

The basic and intermediate data sets are used to check if the models can capture bank balance data properties and to decide if these models can be used to simulate synthetic data out of real bank balance data.

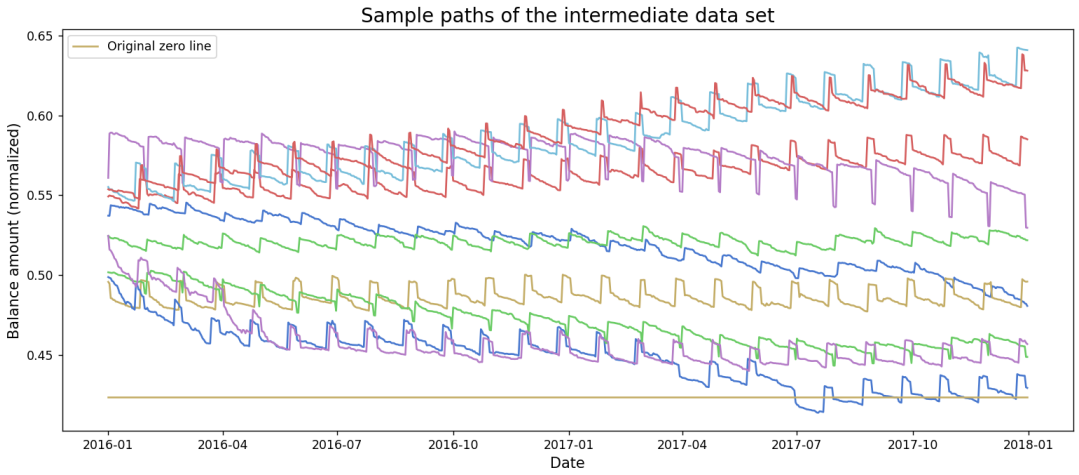


Figure 3.3.: Sample paths of ten customers of the intermediate data set where more randomness is added by varying salary and fixed costs date

3.3. Simulation advanced data set

The advanced data set is an extension of the intermediate data set with extra features, randomness, and groups. This data set is used as input in the best-performing synthetic data generation model. The synthetic data resulting from this model is then used in the second part of the thesis. To get more realistic properties of bank balance data, numbers received from research of CBS (Centraal Bureau voor de Statistiek, or Statistics Netherlands) and expert-based opinions are used[2]. Note that the focus of this thesis is not to simulate the most representative balance amount time series but to test the synthetic data generation models on simulated data sets with known distributions and properties of bank balance data.

The simulation of the advanced data set is executed in three steps. First, properties of the customers, like age, income, and customer type, are simulated. Second, the balance amount and transaction count time series are simulated per customer using the properties information. Third, an economic crisis event is added where the circumstances for the customers are worse.

Simulation of the customer properties Eight properties are generated for each customer in the data set, before simulating the time series.

1. Age is simulated uniformly between 18 and 90.

2. A customer is stable with a probability of 90% and unstable with a probability of 10%. Stable customers have a stable income, stable fixed costs, and less volatile

[2]The expert-based opinions are obtained by employees of RiskQuest who work with transaction data. Since this data is very confidential, only rough indications of the setup of the data set and the correlations in the data set are given.

transactions. These features are vice versa for unstable customers. In practice, one does not know if a customer is stable or unstable, so this information is not given as static feature input in the synthetic data generation models. However, it is included in the data to see if the models pick up two different types of customers and if these can be found in the modeling part (Chapter 8).

3. The salary date is simulated from a distribution where the probability of having salary around more common dates (i.e., the 25^{th}) is higher than other dates.

4. The salary amount is simulated from a beta distribution $Beta(\alpha_1, \beta_1)$ with α_1 and β_1 depending on the customer's age and the customer being stable or unstable. Young and unstable customers have a more right-skewed distribution, while old and stable customers have a more left-skewed distribution. Here, young is defined as having an age under 25 and old is defined as having an age above 75. The salary amount resulting from this beta distribution has a value between zero and one. To reverse normalize this value and get realistic values, the salary amount is changed to a value between 300 and 10.000 as $salary\_amount = salary\_amount \times (10.000 - 300) + 300$. The average amount per age group is derived from data of CBS [19].

5. The fixed costs date is simulated uniformly between 1 and 28.

6. The fixed costs amount is simulated from a beta distribution $Beta(\alpha_2, \beta_2)$ with α_2 and β_2 depending on the customer's age and the customer being stable or unstable, with the same reasoning as for the salary amount. The fixed costs amount is then reversed normalized (in the same way as in step 4) between 40% and 60% of the salary amount. According to Nibud, fixed costs are for Dutch people between 40% to 60% of their income [21].

7. The bonus indicator is simulated from a probability distribution where the probability of getting a bonus increases when the salary amount increases. When the salary is below 2.500, the probability of a bonus is 0. The bonus amount is between 20% and 60% of the salary amount, or equal to 0 when no bonus is given.

8. The start amount of the balance time series depends on the average savings per age reported by the CBS [20] and on the type of customer. Again the amount is simulated from a beta distribution with the parameters adjusted on the average savings per age and on unstable customers having lower savings.

Simulation of the time series per customer When the customers' properties are simulated, the time series with balance amounts and transaction counts are simulated for each customer. For each day, the following six steps are executed. Step 3 and step 4 are applied to break the income seasonalities and therefore add extra randomness to the data set.

1. When the day is equal to the salary day, add the salary amount to the current balance amount.

2. When the day is equal to the fixed costs day, subtract the amount of the fixed costs from the current balance amount.

3. Add holiday pay to the current balance amount on a day in the month May.

4. When the month is equal to December and the day is equal to the salary day, add the bonus to the balance amount (can also be equal to zero).

5. The number of transactions is simulated from a beta distribution $Beta(\alpha_3, \beta_3)$ with α_3 and β_3 depending on age, customer type, and current balance amount. Unstable and younger customers follow a beta distribution that is left-skewed, so with a higher transaction count, other customers follow a right-skewed beta distribution. Also, when a customer has a low balance amount, it has a higher probability of having a lower transaction count. The number of transactions is between 0 and 15.

6. The amount of a transaction is simulated from a beta distribution $Beta(\alpha_4, \beta_4)$ with α_4 and β_4 based on the number of transactions and the current balance amount. Here is assumed that a customer with many transactions in one day has more small transactions (right-skewed beta distribution) and vice versa. Another assumption on transaction amount is that a customer with almost no money left is more careful and has a higher probability of spending less money. To add some extra randomness in the simulation, random high expenses are added when a customer has enough money to spend such a high expense.

Economic crisis event Finally, a special event is added to the data since this can influence the risk models in banks when these are going to use the synthetic data. A special event in time can give another value to the customers' risk. The goal of adding this event is to see if the model can detect not only differences between customers but also differences between time ranges. The special event is an economic crisis in 2018 where the fixed costs increase, the unemployment rate increases because of the reluctance of companies to hire staff, and there will be more part-time workers with part-time WW instead of full-time workers. This information is received from a research of CBS [9]. In the data set this event can be captured by a less increasing trend and a lower average balance amount in 2018 because of the increasing fixed costs and the probability of (party) losing the job.

Example of the simulation The balance amount sample paths of five customers of the advanced data set are visualized in Figure 3.4. Customer 4 is an example of a stable customer with the same pattern and balance amount range over three years. Customers 2 and 3 are also stable customers, but they have suffered from the economic crisis. At customer 3 this can be seen by losing the salary in August of 2018 and at customer 2

by higher fixed costs in 2018. Customers 1 and 5 are unstable customers with a lower balance amount and random months where they have no or less salary.

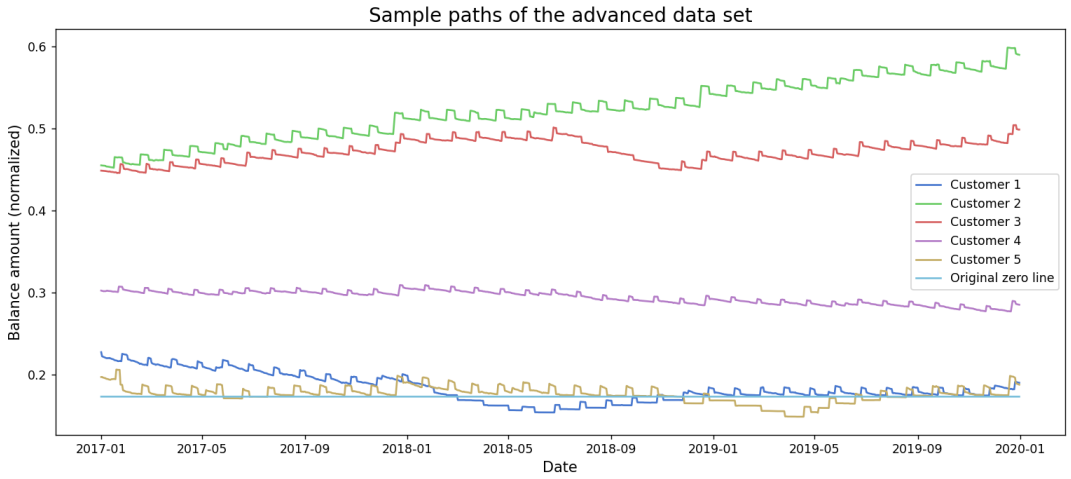


Figure 3.4.: Sample paths of five customers of the input data of the advanced data set

4. Neural Network Preliminaries

In this chapter, the theory behind neural networks and the differences between various neural networks are described. The different neural networks described are Multi-Layer Perceptron (MLP), Recurrent Neural Network (RNN), Gated Recurrent Unit (GRU), and Long Short Term Memory (LSTM). All these types of neural nets are used in the synthetic time series generating models described in Chapter 5. This chapter can be skipped if these types of neural networks are already known.

4.1. Neural Networks

The book Pattern Recognition and Machine Learning by C M Bishop [5] gives a clear mathematical explanation of the basic neural networks. A Neural Network (NN) is a unit of deep learning, which is a type of machine learning that tries to simulate the behavior of input data by learning from the data and giving as accurate as possible predictions of the test data. The architecture of an NN is displayed in Figure 4.1, where the circles are called nodes (or neurons), and a vertical line of nodes is called a layer. In this network, n_1 data points $x_1, \ldots, x_{n_1}$ are given as input in the system, the input layer passes these inputs to the next layer. An NN can have multiple hidden layers where computation is done and weights are transferred to the next hidden layer or the output layer, the example in Figure 4.1 has two hidden layers.

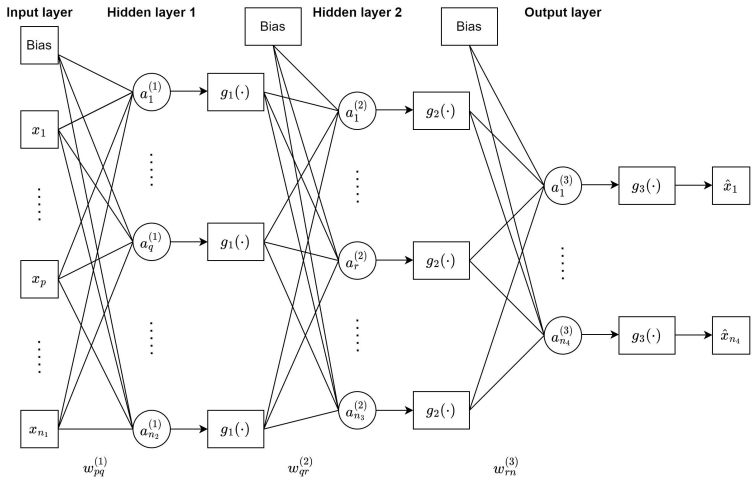


Figure 4.1.: Architecture of a Neural Network with two hidden layers

Each neuron in the first hidden layer transforms the values of the input layer with weights. These neurons are called activations and have the structure as in Equation 4.1.

Here, $a_j^{(1)}$ can be computed for $j \in \{1, \ldots, n_2\}$. The weight w_{j0} is the weight from the bias to the j^{th} hidden node, the weight w_{ji} is the weight from the i^{th} input to the j^{th} hidden node, and the superscript (1) represents the first hidden layer.

$$a_j^{(1)} = \sum_{i=1}^{n_1} w_{ji}^{(1)} x_i + w_{j0}^{(1)}. \tag{4.1}$$

This transformation is put in a non-linear activation function $g(\cdot)$ and results in the output of that layer as $z_j^{(1)} = g(a_j^{(1)})$. When there are multiple hidden layers (L) in the neural network, the activations of the l^{th} hidden layer are

$$a_j^{(l)} = \sum_{i=1}^{n} w_{ji}^{(l)} z_i^{(l-1)} + w_{j0}^{(l)}$$

for $l = 2, \ldots, L$ and n the number of nodes in the l^{th} hidden layer. The output layer transforms the values of the hidden layer as

$$a_k = \sum_{j=1}^{n_L} w_{kj}^{(L)} z_j^{(L)} + w_{k0}^{(L)},$$

with k one of the n_4 nodes in the output layer and n_L the number of nodes in the last hidden layer. The output layer uses an activation function that maps the information in the nodes in the desired output format like $\hat{x}_k = g(a_k)$. Commonly used activation functions $g(\cdot)$ are sigmoid (logistic output), tanh (output between $[-1, 1]$), ReLU (removes negative part), and leaky ReLU (lowers the magnitude of negative part). Two different activation functions are used in this thesis, the first one is the Swish activation function which is defined as $g(y) = y \cdot sigmoid(\beta y)$. This activation function is comparable with the ReLU activation function but outperforms it [33]. The Swish function propagates a small number of negative weights while ReLU sets all these negative weights to zero. The second activation function used is the softplus function which is a smoothed version of the ReLU; $g(y) = \log(1 + \exp(y))$.

The process starting from the input layer till obtaining the outputs $\hat{x}_k$ is called the forward propagation of the neural network. The next step is to measure how well the obtained output is compared to the real output x_k with a cost function $C = \text{cost}(x, \hat{x})$ which can be Mean Squared Error (MSE), cross-entropy, or another cost function. Based on the result C, the network weights and biases are adjusted to optimize the cost function. This is done in a backward pass called backward propagation. To update these values, gradients are calculated to minimize the cost function. The gradient of the cost function with respect to a parameter determines the level of adjustment of that parameter. This means that

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = \frac{\partial C}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial w_{jk}^{(l)}}$$

needs to be computed for $k = 0, \ldots, n_3$ which also includes the bias w_{j0}. Then the weights are updated by

$$w_{jk}^{(l)} := w_{jk}^{(l)} - \epsilon \frac{\partial C}{\partial w_{jk}^{(l)}}$$

with random initial values and ϵ the learning rate. The learning rate determines the influence of the gradient on the updated weight.

The neural network displayed in Figure 4.1 is also called a Multi-Layer Perceptron (MLP) which is characterized by multiple computational layers, fully connected in a feed-forward way. An MLP is able to learn non-linear representations and often has a sigmoid function as activation function defined as $\sigma(y) = \frac{1}{1+e^{-y}}$ which gives output values between 0 and 1.

There are many different types of neural networks for solving different kinds of problems. The ones used for the synthetic data generating models are described in the following subsections.

4.2. Recurrent Neural Network

One type of NN that is used for training on time series data is the Recurrent Neural Network (RNN). An RNN is used to deal with sequential data (like time series) since it does not only take the current input into consideration but also previous inputs. Chapter 4 of the book Neural Network and Deep Learning by A Géron [22] explains the mathematics behind RNNs and special kinds of RNNs discussed later in this chapter.

In Figure 4.2, an overview of an RNN is sketched where each x_t, $t = 1, \ldots, T$ represents a sequence step or a minibatch of successive sequence steps and each $H_t^{(l)}$ represents the l^{th} hidden layer at step t. The data of the first minibatch x_1 is propagated through the network with two hidden layers where each hidden layer remembers the information. For the second minibatch x_2, not only the data is given as input in the first hidden layer but the remembered information from the previous minibatch as well. So the output of a hidden layer is propagated forward as an input for that hidden layer in the next sequence step. With this, the hidden state can memorize previous inputs which is not possible in a simple NN like MLP that assumes that inputs and outputs are independent of each other. In an RNN, the number of input nodes represents the number of steps in the sequence or the number of mini-batches taken from the sequence.

Zooming into sequence step t of hidden layer l of the RNN of Figure 4.2, the architecture of this step has the structure as in Figure 4.3. Define $h_t^{(l)}$ as the output of hidden layer $H_t^{(l)}$. The two inputs shown in the figure are the output of the same hidden layer l but one time-step back ($h_{t-1}^{(l)}$) and the output of the previous hidden layer $l-1$ at the same time-step ($h_t^{(l-1)}$). These two inputs are used as input for the activation function $g_l(\cdot)$ of that layer which gives the output $h_t^{(l)} = g_l(h_{t-1}^{(l)}, h_t^{(l-1)})$. The output will be used as input in the next hidden layer and as input in the same hidden layer for the next time step.

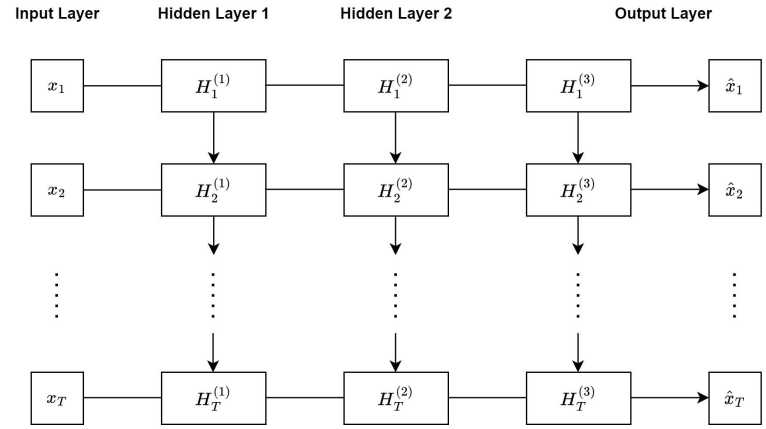


Figure 4.2.: Architecture of a Recurrent Neural Network

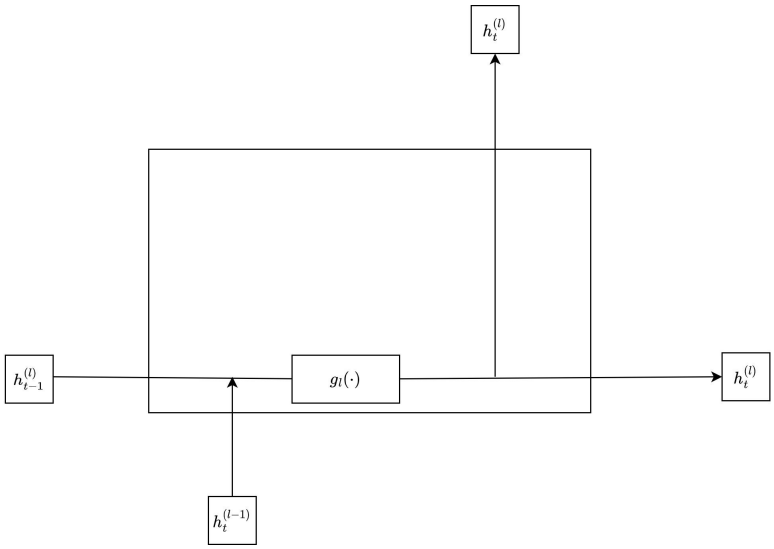


Figure 4.3.: Sequence unit t of layer l of a multi-layer RNN

A disadvantage of an RNN is that at a certain point, when a sequence is getting too long, the RNN can not connect the information at the beginning of the time series with the current time step anymore. This is due to the vanishing gradient problem which is the problem that the gradient shrinks as it back propagates through time. At each time step, the gradient gets smaller and does not contribute much learning anymore. This results in the first layers of the RNN forgetting what they have seen in longer sequences.

Two methods that are special kinds of RNNs that can handle long-term dependencies are Gated Recurrent Units and Long Short Term Memory networks. These are used in the synthetic data generation models in this thesis for better performance and are explained in the following sections.

4.3. Gated Recurrent Unit

A special kind of RNN that is able to learn long-term dependencies is a Gated Recurrent Unit (GRU). The GRUs are designed with two gates, an update gate, and a reset gate. The update gate has to decide the amount of information received from the previous time steps to be passed to the next time steps. In contrast, the reset gate decides the amount which can be forgotten. The architecture of sequence unit t of layer l of an GRU network is displayed in Figure 4.4 [22].

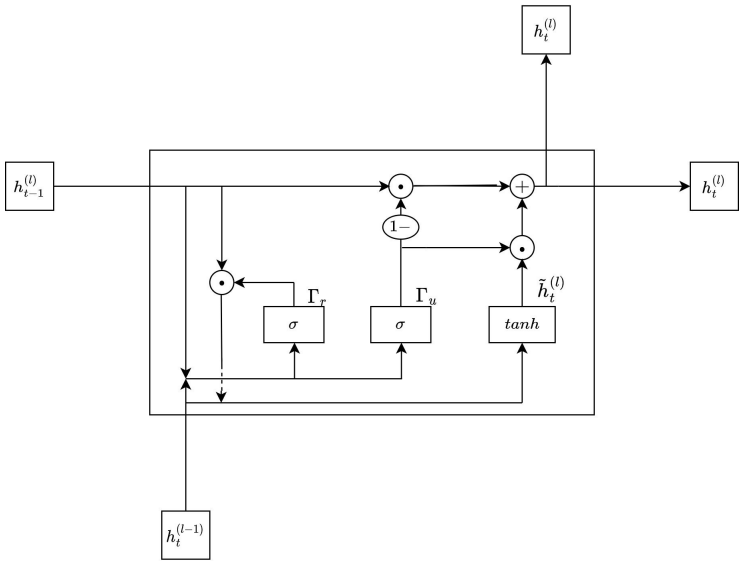


Figure 4.4.: Sequence unit t of layer l of a multi-layer GRU

The activation functions are sigmoid functions (σ) which lead to output values in the interval $[0, 1]$. The left sigmoid function in Figure 4.4 is part of the reset gate Γ_r, which controls how much information from the previous state can be forgotten. This function gets as input the output of the previous time step $h_{t-1}^{(l)}$ and the output of the previous layer $h_t^{(l)}$ and is defined as

$$\Gamma_r = \sigma(W_r[h_{t-1}^{(l)}, h_t^{(l-1)}] + b_r),$$

with W_r the weights and b_r the bias. All the values in Γ_r will be between 0 and 1 because of the sigmoid function, values closer to 0 are less important than values closer to 1. Based on trained weights, the values closest to 0 are forgotten. The second sigmoid function is part of the update gate Γ_u and decides how much of the previous state information is involved in the current state. The update gate uses the same function as the reset gate but with different weights W_u and bias b_u and is formulated as

$$\Gamma_u = \sigma(W_u[h_{t-1}^{(l)}, h_t^{(l-1)}] + b_u).$$

With the output of the reset gate Γ_u, a candidate hidden state can be generated with the part of the previous state that is not forgotten. This is done with a tanh function

resulting in outputs in the range $[-1, 1]$ to learn which information can be subtracted from the current state and which information has to be added to the current state. The candidate hidden state is

$$\tilde{h}_t^{(l)} = \tanh\left(W_h[\Gamma_r \cdot h_{t-1}^{(l)}, h_t^{(l-1)}] + b_h\right),$$

with weights W_h and bias b_h. The final action left is the incorporation of the update gate which results in the output called the hidden state. The hidden state is the old state $h_{t-1}^{(l)}$ versus how much it resembles the new candidate state. The update gate Γ_u decides how much of the previous state has to be involved and the part remaining to involve comes from the candidate state, this results in the hidden state

$$h_t^{(l)} = \Gamma_u \cdot h_{t-1}^{(l)} + (1 - \Gamma_u) \cdot \tilde{h}_t^{(l)}.$$

4.4. Long Short Term Memory

Another type of RNN that is able to learn long-term dependencies is the Long Short Term Memory (LSTM) network. It is slower than GRU and uses more memory because the GRU has fewer tensor operations, but the LSTM is overall more accurate. The architecture of sequence unit t of layer l of an LSTM network is displayed in Figure 4.5 [22].

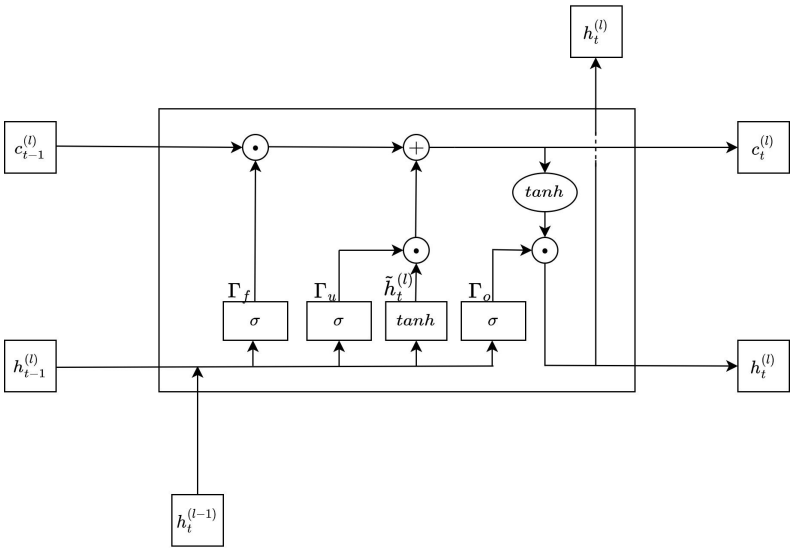


Figure 4.5.: Sequence unit t of layer l of a multi-layer LSTM

The LSTM network uses three gates (instead of two in the GRU network), namely the input gate, output gate, and forget gate. The input gate Γ_u controls how much of the input needs to be added to the current cell's internal state. The forget gate Γ_f decides whether to keep or to forget a value and the output gate Γ_o determines which information is carried as output. The outputs of these three gates are all computed with

a sigmoid activation function but with different weights and biases. The formulas for the forget gate Γ_f, input gate Γ_u, and output gate Γ_o are

$$\begin{aligned}\Gamma_f &= \sigma(W_f[h_{t-1}^{(l)}, h_t^{(l-1)}] + b_f),\\ \Gamma_u &= \sigma(W_u[h_{t-1}^{(l)}, h_t^{(l-1)}] + b_u),\\ \Gamma_o &= \sigma(W_o[h_{t-1}^{(l)}, h_t^{(l-1)}] + b_o).\end{aligned}$$

Just as in the GRU network, a candidate hidden state is computed in the range $[-1, 1]$ to learn which information can be subtracted from the cell state and which information has to be added to the cell state. The candidate hidden state is

$$\tilde{h}_t^{(l)} = \tanh\left(W_h[h_{t-1}^{(l)}, h_t^{(l-1)}] + b_h\right).$$

Another difference between the GRU and the LSTM networks is that the LSTM has three inputs instead of two. The two inputs which are the same as in an GRU network are the hidden states of the previous timestamp and previous layer, $h_{t-1}^{(l)}$ and $h_t^{(l-1)}$, which is known as the short-term memory. The extra input is the cell state of the previous timestamp, $c_{t-1}^{(l)}$, and is known as the long-term memory. The output of the cell state is everything that can not be forgotten from the previous cell state plus everything from the candidate's hidden state that can not be forgotten. In formula from this is equal to

$$c_t^{(l)} = \Gamma_u \cdot \tilde{h}_t^{(l)} + \Gamma_f \cdot c_{t-1}^{(l)}.$$

Finally, the output gate and the updated cell state together result in the current hidden state as

$$h_t^{(l)} = \Gamma_o \cdot \tanh\left(c_t^{(l)}\right).$$

5. Synthetic data generating models

In this chapter, the different types of models used for generating synthetic bank balance data are described. Two main models, the Probabilistic AutoRegressive (PAR) model and the Generative Adversarial Network (GAN), are explained together with two extensions of the GAN; TimeGAN and DoppelGANger.

5.1. PAR model

The Probabilistic AutoRegressive (PAR) model is a model from the synthetic data vault (SDV) ecosystem of libraries for generating new synthetic data with the same statistical properties as the original data set. The SDV has models for single table data, relational data, and time series data but in this thesis the focus is on the PAR model, which is the model for time series data. In four steps, the PAR model generates time series data based on an autoregressive model, see Formula A.1 in Section A.1. The first step pre-processes the data, the second step simulates the static features, then an RNN is trained on the time series data to estimate the AR model parameters and minimize the loss, and the last step is to use these estimated parameters to reconstruct synthetic data [50].

Data pre-processing First, the PAR model pre-processes the data in the right format where the input data has the structure as in Table 3.2. Hereby it fills the missing data with the average of the column or it creates a new column that indicates if the value was missing. Besides filling in the missing values, the model creates numerical values of the categorical data with the one-hot encoding technique. Every unique value of the categorical data is a new feature in the model. The last pre-processing step is normalization, where for the continuous numerical data a z-score transformation is applied as $Z(x) = \frac{x-\mu}{\sigma}$, with μ the mean and σ the standard deviation of the column. For the discrete numerical data, the normalization applied is the min-max method as $N(x) = \frac{x-\min}{\max-\min}$.

Static feature simulation In the second step, the PAR model simulates the static feature(s) S, which is only the age of the customers in this thesis. The static features are simulated by applying a Gaussian Copula model to the real static feature data. The Gaussian Copula model is described in Section A.2.

Neural Network training In this third step, the PAR model trains an RNN to estimate the sequence parameters $\theta_1, \ldots, \theta_p$ of the AR model. As seen in Chapter 4,

RNNs are useful for generating time series data since these models also take previous time steps into consideration. The PAR model needs as input the sequence history and the static feature, and it outputs the parameters which are needed to generate the next item in the sequence, this is visualized in Figure 5.1. The static feature S does not change over time and $x_0, x_1, \ldots$ are the time series data points where each data point (x_i) depends on all previous data points $(x_0, \ldots, x_{i-1})$. In this research, x_i is a two-dimensional vector with balance amount and corresponding transaction count of one day. After estimating the parameters, the sequence of data points and the parameters are inputs for a loss function. The model adjusts the parameters based on minimizing the loss function. This is called the sequential model algorithm.

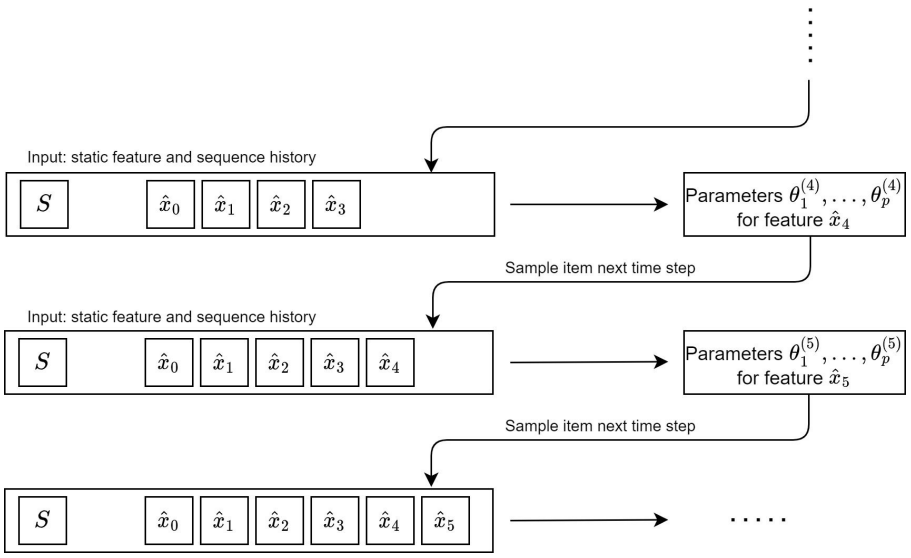


Figure 5.1.: Process of generating a time series with the PAR model [50]

To explain the loss functions, define the simulated sequences as $\hat{x}^{(0)}, \hat{x}^{(1)}, \ldots$ depending on time t. For sequence i the time dependence is defined as $\hat{x}_0^{(i)}, \hat{x}_1^{(i)}, \ldots$. Likewise, the parameters of the sequence i at time step t are $\theta_{(t,0)}^{(i)}, \theta_{(t,1)}^{(i)}, \ldots, \theta_{(t,k-1)}^{(i)}$ where k numbers describe the probability distribution (e.g., mean and standard deviation). The overall loss function depends on the losses for each data point depending on i and t. The total loss is then the sum of these individual losses which needs to be minimized and is defined as

$$\mathcal{L} = \sum_i \sum_t \sum_{j=0}^{k-1} \mathcal{L}(\hat{x}_t^{(i)}, \theta_{(t,j)}^{(i)}).$$

The loss function per sequence depends on the sequence values being continuous, discrete, or categorical. Since no categorical data is used in this thesis, only the loss functions for discrete and continuous data are described. For continuous numerical data (the balance amount), the loss function depends on three parameters, the mean $\theta_{(t,\mu)}^{(i)}$, standard deviation $\theta_{(t,\sigma)}^{(i)}$ and the probability that a value is missing $\theta_{(t,m)}^{(i)}$. Call these

three parameters μ, σ and m, respectively. The loss function is then defined as

$$\begin{cases} \mathcal{L}(x;\mu,\sigma,m) = -(\log(f_{\mu,\sigma^2}(x)) + \log(1-m)) & \text{if } x \text{ is not missing} \\ \mathcal{L}(x;\mu,\sigma,m) = -\log(m) & \text{if } x \text{ is missing} \end{cases}$$

with x the real value and f the probability density of a Gaussian distribution with mean μ and variance σ^2. The log contributes to the probability that a value is missing, where a higher probability of missing results in more contribution to the equation.

For discrete numerical data (the transaction count), the loss function depends on the parameters of a negative binomial distribution $\theta^{(i)}_{(t,r)}$ and $\theta^{(i)}_{(t,\rho)}$ and also the probability that a value is missing $\theta^{(i)}_{(t,m)}$. Call these three parameters r, ρ and m, respectively. The loss function is then defined as

$$\begin{cases} \mathcal{L}(x;r,\rho,m) = -(\log(f_{r,\rho}(x)) + \log(1-m)) & \text{if } x \text{ is not missing} \\ \mathcal{L}(x;r,\rho,m) = -\log(m) & \text{if } x \text{ is missing} \end{cases}$$

with the same reasoning as for continuous data. Here f is the probability density function of the negative binomial distributions with parameters r and ρ.

The last parameter is used for determining if the sequence is terminated and is called $\theta^{(0)}_{(t,\tau)}$ (τ in short). The loss function is

$$\mathcal{L}(\kappa;\tau) = -(\kappa\log(\tau) + (1-\kappa)\log(1-\tau))$$

where $\kappa = 0$ when the sequence has not terminated and $\kappa = 1$ when the sequence has terminated.

The architecture of the PAR model is a fully connected GRU with the Swish activation function in the fully connected layers. The output activation function depends again on the type of data, for continuous data the parameters μ and σ use the Softplus function. The parameter m uses the Sigmoid activation function with values between 0 and 1. When the data is discrete, the parameter r uses the Softplus activation function and ρ the Sigmoid function. See Chapter 4 for the explanation of GRU and the activation functions.

Generating synthetic time series After training the model, new sequences can be created by the trained NN and these sequences construct the synthetic data. Suppose t time steps of a sequence are available, the next time step is defined by first estimating the probability distribution parameters by the trained NN as follows

$$\mathbb{P}(\hat{x}_{t+1}|\hat{x}_0,\hat{x}_1,\ldots,\hat{x}_t;S) = f(\hat{x}_{t+1};\theta_{(t+1,0)},\theta_{(t+1,1)},\ldots,\theta_{(t+1,k-1)}).$$

To get a value for the next time step, a value is randomly sampled from f; $\hat{x}_{t+1} \sim f$. This next time step value is added to the sequence and used as input of the probability density function f of $\hat{x}_{t+2}$. This generates step by thep the synthetic data sequences.

5.2. GAN models

In this section, the methodology of GAN models is explained with two extensions in the subsections. The framework of generative adversarial networks (GANs) was first proposed by Goodfellow et al. [23]. This paper combines two existing techniques, discriminative models (D) and generative models (G), for better performance in deep learning. According to Goodfellow et al., the most striking successes in deep learning have involved discriminative models. On the other hand, generative models had less striking successes due to difficulties in probabilistic computations. Combining these two models resolves these difficulties and results in a model that can represent the data distributions well.

The basic idea of GAN is that the two models (D and G) compete with each other to get the best results. The generator gets as input a random data set with noise, it then captures the data distribution and generates 'fake' samples. The fake samples are together with the real samples the input of the model discriminator. The discriminator analyses the two samples and gives a probability of whether the data coming from the generator is real or synthetic data. This output is propagated through a cost function and the information received from that function is given back to the generator and discriminator via backpropagation. The two models D and G then compete against each other which drives both to improve their methods until the 'fake' samples can not be distinguished from the real samples anymore. Figure 5.2 gives a visual overview of the GAN structure. The discriminator and generator are both neural networks used for training, for example a multilayer perceptron (MLP, see Chapter 4). For different datasets, different neural networks can be used, for example convolutional neural networks (CNNs) are often used in image programming and recurrent neural networks (RNNs) in time series data.

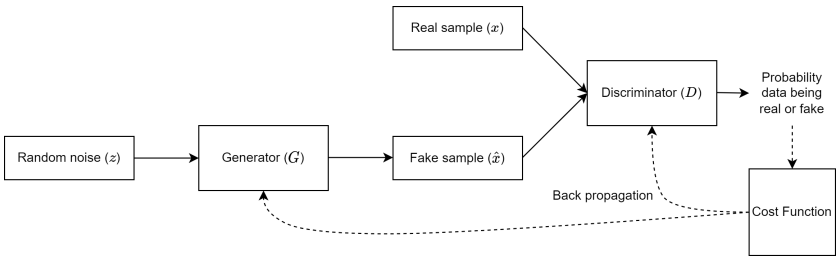


Figure 5.2.: Visual overview of the generative adversarial networks [34]

Since the generator and the discriminator compete against each other, the network becomes a minimax problem. Minimax is a decision rule in game theory for minimizing the maximum loss for a particular player when competing against other players. In the GAN structure, the minimax problem can be recognized since the generator wants to minimize the maximum performance of the discriminator. The optimal value of this problem is defined in a value function depending on the action taken by one player and the actions taken by other players. For GANs, the value function depends on the output of the generator and the output of the discriminator and is defined as $V(G, D)$.

To describe the GAN algorithm, define x as the real data, z as the input noise, and $\hat{x} = G(z)$ as the generated ('fake') data. The discriminator D outputs a probability with $y = D(x)$ representing the probability that the data is real rather than simulated and $\hat{y} = D(\hat{x}) = D(G(z))$ the probability that the data is simulated. The goal of the discriminator is to correctly label the data, so y as 1 and $\hat{y}$ as 0 [39]. With this, the loss function (which needs to be minimized) of the discriminator can be constructed as

$$\mathcal{L}_D = \text{Error}(y, 1) + \text{Error}(\hat{y}, 0),$$

where *Error* is a function that gives the distance between two parameters (i.e., cross-entropy or Kullback-Leibler divergence). The goal of the generator, however, is to confuse the discriminator, so the goal for G is to have a $\hat{y}$ as close as possible to 1. This results in the loss function of the generator as

$$\mathcal{L}_G = \text{Error}(\hat{y}, 1).$$

The error function used in GAN models is binary cross entropy which is feasible since the discriminator labels data as zero or one. The general binary cross entropy function is $H(z, \hat{z}) = -(z \log(\hat{z}) + (1 - z) \log(1 - \hat{z}))$ where z is the real label and $\hat{z}$ the generated label. Implementing this in the loss functions of D and G gives

$$\begin{aligned}\mathcal{L}_D &= -(\log(y) + \log(1 - \hat{y})) \\ \mathcal{L}_G &= -(\log(\hat{y})).\end{aligned}$$

Intuitively, the loss of D is small when y is close to 1 and $\hat{y}$ is close to zero, and the loss of G is small when $\hat{y}$ is close to 1, exactly the goals of the discriminator and generator, respectively. When minimizing the loss functions, the data is sampled in mini-batches of size m. The stochastic gradients of these two loss functions for a mini-batch are

$$\begin{aligned}&\nabla_{\theta_d} \frac{1}{m} \sum_{t=1}^{m} [\log y_t + \log(1 - \hat{y}_t)] \\ &\nabla_{\theta_g} \frac{1}{m} \sum_{t=1}^{m} \log(1 - \hat{y}_t),\end{aligned}$$

respectively. Here, θ_d and θ_g are the MLP parameters of the discriminator and the generator, respectively.

The discriminator is trying to minimize the loss function, independent from the generator, which is the same as maximizing $\log(y) + \log(1 - \hat{y})$. Since the generator wants to fool the discriminator, it tries to minimize this loss function which results in the minimax problem

$$\mathcal{L} = \min_G \max_D [\log(y) + \log(1 - \hat{y})].$$

This loss function is valid for only one data point, so to consider the whole data set the expectation is taken and the value function of the minimax function is defined as

in Equation 5.1 with p the distribution of the real data and $\hat{p}$ the distribution of the generated data.

$$\min_G \max_D V(D,G) = \min_G \max_D \mathbb{E}_{\mathbf{x}\sim p}[\sum_t \log y_t] + \mathbb{E}_{\mathbf{z}\sim \hat{p}}[\sum_t \log(1-\hat{y}_t)] \tag{5.1}$$

The algorithm of the GAN optimizes the minimax problem for a certain number of iterations (n_{iter}). It updates the discriminator and the generator by maximizing and minimizing, respectively. The number of iterations is one of the model hyper parameters and is an input of the user, without hyper parameter tuning this can result in over- or under-sampling when applying too much or too few iterations. A hyper parameter k decides the number of times to apply the discriminator per iteration, this hyper parameter can also be tuned to get the best results. The steps to implement this model are described in Algorithm 1 [23].

Algorithm 1 GAN optimalization

for n in range(n_{iter}) **do**;

for k steps **do**;

- Sample m noise samples $\{z_1, \ldots, z_m\}$ from noise prior $\hat{p}(z)$.
- Sample m samples $\{x_1, \ldots, x_m\}$ from the real data $p(x)$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{t=1}^{m} [\log y_t + \log(1-\hat{y}_t)].$$

end for

- Sample m noise samples $\{z_1, \ldots, z_m\}$ from noise prior $\hat{p}(z)$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{t=1}^{m} \log(1-\hat{y}_t).$$

end for

5.2.1. TimeGAN model

The original GAN model described in the previous part of this section does not preserve the dependencies between time points in a sequence and does not account for temporal correlations. This original model is therefore not an adequate model for synthetic time series generation but the architecture can be used and extended to get better performance for time series data.

The TimeGAN model is published by Yoon et al. [49] to create a model that can handle the difficulties of time series data. The TimeGAN model captures the difficulties of preserving dependencies between time points and accounting for temporal correlations. It combines the AR model (see Section A.1) with the GAN model to solve the negative parts of both models towards generating time series and results in a better-performing adversarial network. The AR and the GAN model both have properties needed for simulating synthetic data. The AR model can factor the distribution of the time series into a product of conditionals $\prod_t p(x_t|x_{1:t-1})$, and therefore it affords control. However, it can not randomly sample new series without external conditioning; it is not generative. For the GAN model, this is the other way around, it is generative (so flexible) but can not factor the distribution of the time series because it does not take the dependencies into account.

The setup of the TimeGAN model is a network consisting of two extra components besides the discriminator and generator (the adversarial components), namely an embedding function and a recovery function. These extra components (the autoencoding components) are trained jointly with the adversarial components. Figure 5.3 gives a block diagram of the components of the TimeGAN model.

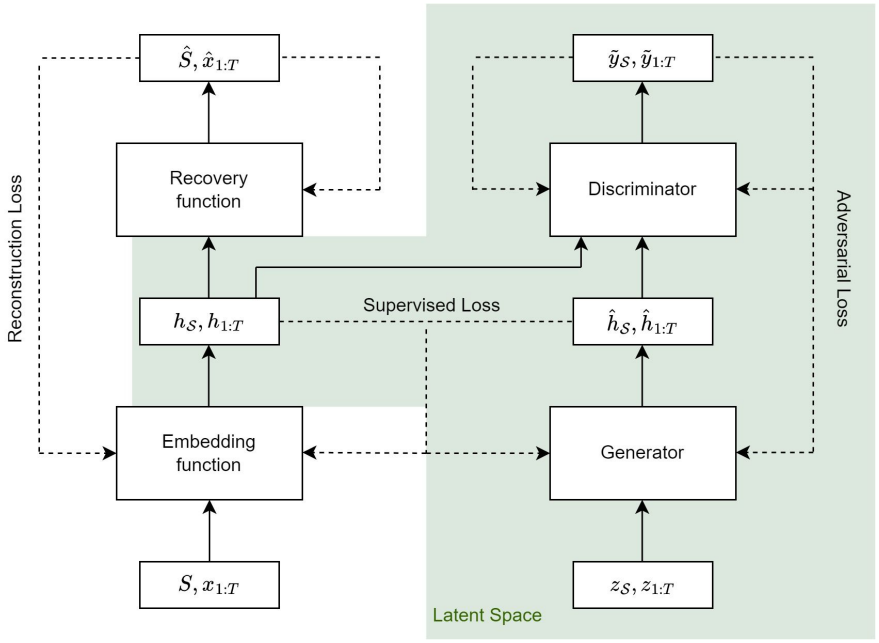


Figure 5.3.: Visual overview of the TimeGAN model [49]

The embedding and recovery functions operate between the feature and latent spaces. The feature space is the space where the data is in, so it contains all the original features of the data and has the structure as visualized in Figure 3.1. The latent space is a lower-dimensional space spanned by the latent variables which are not directly observable variables describing the data. This space contains compressed features of the data that can not be interpreted directly but these features encode meaningful internal representations of externally observed events. The network can learn temporal dynamics

via lower-dimensional representations in the latent space.

Define $\mathcal{H}_\mathcal{S}$ and $\mathcal{H}_\mathcal{X}$ as the latent spaces corresponding to the static feature space $\mathcal{S}$ with $S \in \mathcal{S}$ and the temporal feature space $\mathcal{X}$ with $x \in \mathcal{X}$ respectively. Then the latent codes $h_\mathcal{S}$ and $h_{1:T}$ are constructed by the embedding functions $e_\mathcal{S} : \mathcal{S} \to \mathcal{H}_\mathcal{S}$ and $e_\mathcal{X} : \mathcal{H}_\mathcal{S} \times \mathcal{H}_\mathcal{X} \times \mathcal{X} \to \mathcal{H}_\mathcal{X}$ as

$$h_\mathcal{S} = e_\mathcal{S}(S) \text{ and } h_t = e_\mathcal{X}(h_\mathcal{S}, h_{t-1}, \mathbf{x}_t),$$

so every time step depends on the static feature and the previous time step. The recovery functions $r_\mathcal{S} : \mathcal{H}_\mathcal{S} \to \mathcal{S}$ and $r_\mathcal{X} : \mathcal{H}_\mathcal{X} \to \mathcal{X}$ operate in opposite direction and reconstruct the feature representations as

$$\hat{S} = r_\mathcal{S}(h_S) \text{ and } \hat{x}_t = r_\mathcal{X}(h_t).$$

In theory, any architecture can be used to construct the embedding and recovery functions. In the TimeGAN model RNNs are used where an input parameter is available to choose between GRU or LSTM (see Chapter 4).

The workflow of the adversarial components in TimeGAN differs from the original GAN structure because of the autoencoding components. The output of the generator is in the TimeGAN structure first in the latent space instead of directly in the feature space. Let $\mathcal{Z}_\mathcal{S}$ and $\mathcal{Z}_\mathcal{X}$ be the vector spaces where the random samples are drawn. Then the generating functions $g_\mathcal{S} : \mathcal{Z}_\mathcal{S} \to \mathcal{H}_\mathcal{S}$ and $g_\mathcal{X} : \mathcal{H}_\mathcal{S} \times \mathcal{H}_\mathcal{X} \times \mathcal{Z}_\mathcal{X} \to \mathcal{H}_\mathcal{X}$ generate fake samples in the latent space as

$$\hat{h}_\mathcal{S} = g_\mathcal{S}(\mathbf{z}_\mathcal{S}) \text{ and } \hat{h}_t = g_\mathcal{X}(\hat{h}_\mathcal{S}, \hat{h}_{t-1}, z_t).$$

The last part is the labeling of the discriminator, which is in the latent space as well. The output layer classification functions are defined as $d_\mathcal{S}$ and $d_\mathcal{X}$ and have outputs in $[0, 1]$. The outputs are

$$\tilde{y}_\mathcal{S} = d_\mathcal{S}(\tilde{h}_\mathcal{S}) \text{ and } \tilde{y}_t = d_\mathcal{X}(\overleftarrow{u}_t, \overrightarrow{u}_t),$$

where $\overrightarrow{u}_t = \overrightarrow{c}_\mathcal{X}(\tilde{h}_\mathcal{S}, \tilde{h}_t, \overrightarrow{u}_{t-1})$ and $\overleftarrow{u}_t = \overleftarrow{c}_\mathcal{X}(\tilde{h}_\mathcal{S}, \tilde{h}_t, \overleftarrow{u}_{t-1})$ are the forward and backward hidden states respectively. The notation $\tilde{h}$ and $\tilde{y}$ represent that the embeddings are either real (h and y) or fake ($\hat{h}$ and $\hat{y}$). Again the RNN used can be chosen to be GRU of LSTM, in the model constructed by the authors this choice is the same as for the autoencoding elements but this can be different when the model is self-implemented.

As can be seen in the overview in Figure 5.3, the model relies on three different loss functions, reconstruction loss, adversarial loss, and supervised loss, which represent three objective functions. The reconstruction loss depends on the embedding and recovery functions which have to create accurate reconstructions of the original data from their latent representations, so this loss is equal to the mean squared error (MSE) between the real data (S, x_t) and the reconstructed data $(\hat{S}, \hat{x}_t)$. Mathematically this is

$$\mathcal{L}_R = \mathbb{E}_{S,x\sim p}[\|S - \hat{S}\|_2 + \sum_t \|x_t - \hat{x}_t\|_2].$$

The adversarial loss (or unsupervised loss) $\mathcal{L}_U$ depends on the generator and discriminator and has the same structure as the minimax problem seen in Equation 5.1. Again this allows the discriminator to maximize and the generator to minimize the likelihood of providing correct classifications and is structured as

$$\mathcal{L}_U = \mathbb{E}_{S,x\sim p}[\log y_{\mathcal{S}} + \sum_t \log y_t] + \mathbb{E}_{S,z\sim\hat{p}}[\log(1-\hat{y}_{\mathcal{S}}) + \sum_t \log(1-\hat{y}_t)].$$

The supervised loss connects the adversarial components with the autoencoding components. The loss function is the mean squared error of the latent representation of the data after embedding (h_t) and the next sequence based on the latent representation of the generator ($\hat{h}_t$). Mathematically this is

$$\mathcal{L}_S = \mathbb{E}_{S,x\sim p}[\sum_t \|h_t - g_{\mathcal{X}}(h_{\mathcal{S}}, h_{t-1}, z_t)\|_2].$$

For optimization of the model, the components are trained on two objectives. Denote $\theta_e, \theta_r, \theta_g, \theta_d$ as the parameters of the embedding, recovery, generator, and discriminator networks, respectively. The embedding and recovery functions are trained on both the supervised loss and the reconstruction loss, which have to be minimized as

$$\min_{\theta_e,\theta_r}(\lambda\mathcal{L}_S + \mathcal{L}_R),$$

with $\lambda \geq 0$ a hyperparameter that takes care of the balance between the two losses. The second objective accounts for the generator and discriminator and is equal to

$$\min_{\theta_g}(\eta\mathcal{L}_S + \max_{\theta_d}\mathcal{L}_U),$$

with $\eta \geq 0$ is also a hyperparameter that takes care of the balance between the two losses and this objective is again a minimax game.

By training on these two objectives in n steps, which is again a hyper parameter, the synthetic data is optimized to have the same characteristics as much as possible as the input data.

5.2.2. DoppelGANger model

Another extension of the GAN model is generated because the original GAN model, described at the beginning of this chapter, does not preserve long-term dependencies and multidimensional relationships. The reason for this is that the MLP must jointly learn all of the cross-correlations in the data set and therefore cannot learn the patterns that mainly occur in time series data sets. The DoppelGANger (DG) model solves the problem of not finding patterns by including recurrent neural networks (RNNs) next to the MLP; these RNNs incorporate previous patterns when generating a subsequent time step.

The overall overview of the DG model is outlined in Figure 5.4, with customer data input as in Figure 3.1. In contrast to the GAN and TimeGAN models, this model has

three generator functions. Each of these generators solves a problem that the GAN model encounters in generating time series. Two of the generators are for generating static features, solving the problems of complex multidimensional relationships and mode collapse. The other generator is for the time series features generation which captures the difficulties in finding long-term dependencies.

What also differs from the original GAN model is that the DG model has two different discriminators. The auxiliary discriminator in the yellow block is used for discriminating the static features separately from the dynamic features. The discriminator in the grey block discriminates the static and dynamic features together [27].

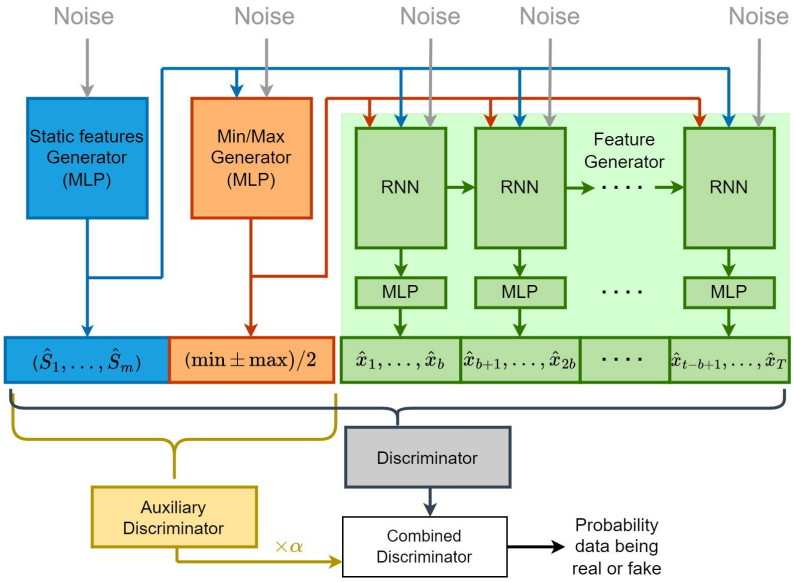


Figure 5.4.: Visual overview of the doppelGANger model [47]

First, focus on the methodology of the three generators. The static feature generator is visualized in the blue blocks in Figure 5.4. This generator is an MLP that connects the static features with the dynamic data. The static feature generator has two sub-tasks, namely first generating static features ($\hat{S}_i$) and then generating the measurements time series conditioned on the generated static feature ($\hat{x}_i$) as $P(\hat{S}_i, \hat{x}_i) = P(\hat{S}_i) \cdot P(\hat{x}_i|\hat{S}_i)$, capturing the complex multidimensional relationships. Since the static features are non-time series data, an MLP is a suitable model to generate the static features. The generated static feature $\hat{S}_i$ is then used as an input to the RNN in the feature generator.

The min/max generator is visualized in the orange blocks in Figure 5.4. This generator will tackle the mode collapse problem. This is a known problem for GANs where the generator generates data distributions in the most common range of values and notices that this is enough to fool the discriminator. The extreme values are not generated in the synthetic data as a result of the discriminator already being fooled, which results in a mode collapse. To solve this problem, the DG model normalizes the balance amount not for the whole data set but per customer and remembers the minimum and the maximum values as static features. In this case, there are no extremes anymore, which solves the

mode collapse problem. The min/max generator generates the static minimum and maximum features per customer with an MLP. The values in the generated time series can be converted back to the original range by using these synthetic maximum and minimum features to get output data in the same order of values.

The dynamic feature generator is visualized in the green blocks in Figure 5.4. As described in Chapter 4, an RNN incorporates the time steps $1, \ldots, t-1$ when a new data point at time t is generated. The RNNs for feature generating can be chosen by the user of being GRUs or LSTMs (see Chapter 4). The MLPs connected to the outputs of the RNNs are inserted to fix the desired dimensionality of the output. This is necessary since the dimensionality of an RNN depends on the number of RNN units but this is not always the dimensionality wanted as output. The MLPs inserted have the desired dimensionality output. From the block diagram of the DG model, it seems like multiple RNNs generate a batch of the full time series, in fact this is the same RNN used for many time batches. These time batches are included in the model since an RNN generator can still struggle with temporal correlations if the time series is too long (e.g., more than 500 time steps). Therefore the MLP reads the output from the RNN and generates the sample batch of a chosen size (b in the block diagram).

Even though RNNs can capture long-term relationships in data, using the described architecture for training on the static feature and time series data generates poorly synthetic data. This is due to the discriminator judging the samples and giving feedback for the generator to improve, this is hard when the dimension of the time series data is large. To solve this problem, an auxiliary discriminator is added to the network which discriminates only on the static features. The generator also learns from this auxiliary discriminator to generate trustworthy static features. Together, these static features and the discriminator generate high-fidelity measurements of the dynamic features. The two losses from the two discriminators are combined and with a weighting parameter α the influence of the auxiliary discriminator can be decided. The total loss function is

$$\mathcal{L} = \min_{G} \max_{D_1, D_2} \mathcal{L}_1(G, D_1) + \alpha \mathcal{L}_2(G, D_2),$$

where $\mathcal{L}_1$ is the Wasserstein loss for the discriminator and $\mathcal{L}_2$ the Wasserstein loss for the auxiliary discriminator [48]. The Wasserstein loss is defined as

$$\mathcal{L}_i(G, D_i) = \mathbb{E}_{x \sim p}[T_i(D_i(x))] - \mathbb{E}_{z \sim \hat{p}}[D_i(T_i(G(z)))] - \lambda \mathbb{E}_{x^* \sim p_{x^*}}[(\nabla_{x^*} \| D_i(T_i(x^*)) \|_2 - 1)^2],$$

where T_1 describes the static and dynamic features of the data, T_2 describes only the static feature part of the data, and $x^* = tx + (1-t)(G(z))$ where t is a value from the uniform distribution $t \sim U[0, 1]$. The hyperparameter $\lambda \geq 0$ takes care of the balance between the discriminator and generator loss [27].

6. Evaluation metrics

In this chapter, the methodology of the metrics to compare the generated synthetic data with the input data is described. The comparison of the two data sets can be done in different ways, three main metric types are described in this chapter. In Section 6.1, the statistical properties and measures will be described, then in Section 6.2 two kinds of scores that derive the error between the two sets with machine learning are explained. Next, visualization techniques where the overlap of the distributions of the data sets can be compared are described in Section 6.3. Finally, Section 6.4 gives an overview of all these metrics.

6.1. Statistical properties and measures

In this section, a couple of statistical properties are explained to compare the real data with the synthetic data. The easiest properties are the mean (μ) and the variance (σ^2) of a data set. These will show whether the two data sets will have balance data in the same range of values. Then depending on the mean and the variance, the correlation between the two data sets can be computed with Pearson correlation

$$\rho_{x,\hat{x}} = \frac{\mathbb{E}[(x - \mu_x)(\hat{x} - \mu_{\hat{x}})]}{\sigma_x \sigma_{\hat{x}}},$$

where x is the real data and $\hat{x}$ is the generated data [31]. When the correlation between the two data sets is close to one, the synthetic data is more similar to the real data compared to a correlation close to zero. The opposite of this is to compute the error between the two data sets, which is desired to be as low as possible with a minimum of zero. For this, the Root Mean Squared Error (RMSE) will be used and is defined as

$$RMSE = \sqrt{\frac{1}{t}\sum_{i=1}^{t}(x_i - \hat{x}_i)^2},$$

with t the number of time steps [28].

With the mean and the variance the range of values is simply described, but with Shannon entropy (E) the degree of variation in a time series can be described. When E is high, the frequency of values is more equally distributed and when E is low, there is a larger concentration around certain values [46]. The Shannon entropy can be computed by

$$E(x) = -\sum_{i=1}^{t} x_i \cdot \log(x_i).$$

To compare the original with the synthetic data, $E(x)$ and $E(\hat{x})$ have to be computed. The variations in the two data sets are alike when $E(x)$ and $E(\hat{x})$ are alike.

Since time series are generated, it is also interesting to look at the autocorrelations of the two different data sets. The autocorrelation of a time series describes the degree of similarity between a time series' current value and its past values over successive time intervals. When the autocorrelation of the synthetic data set has the same pattern as the autocorrelation of the real data set, the two sets have the same row interdependencies which some models claim to capture. Also, the seasonalities are visible in the autocorrelation plot, which makes it visible whether the synthetic data generation models have captured the seasonalities well.

The data sets have two time series per customer, the balance amount and the transaction count. In practice, these two time series are not necessarily correlated; one customer can have one big transaction on one day but also ten small transactions. A significant difference in balance amount between two days does not necessarily mean many transactions. Most of the customers in the advanced data set have a low correlation between the two time series but there will also be a group that has a somewhat higher correlation between the two time series. That is why one of the metrics is the Cumulative Distribution Function (CDF) of the correlation coefficients between the balance amount and the transaction count. The CDF is a distribution function that shows the probability that a variable Z takes on a value less than or equal to z: $\mathbb{P}(Z \leq z)$. The CDF of the inter correlation of every client in the data set is computed and plotted for the whole set, this will show the distribution of the inter correlation values of all the customers. By showing a distribution instead of taking an average of all the correlations, it becomes clear whether a specific range of values is captured by the synthetic data generation model.

Finally, the generated synthetic static features will be compared to the original static features in two ways. First, the Wasserstein distance between age groups is computed. This means that the two data sets are both split up into three groups (age $\leq$ 25, 25 $<$ age $<$ 75, age $\geq$ 75), and each real data set group is compared with the synthetic data set group by the Wasserstein distance. This distance is defined as

$$W(x, \hat{x}) = \inf_{\gamma \sim \Pi(x, \hat{x})} \mathbb{E}_{(u,v)\sim\gamma}[\|u - v\|],$$

where the infimum is taken over the distance between all pairs (u, v), which are samples from the product distribution of the real and the synthetic data $\Pi(x, \hat{x})$ [10]. This distance is optimally as close to zero for each age group.

The second way is to plot the distribution of the ages on top of each other to see whether these are equally divided. The goal for the age distributions is to not be exactly the same since this contradicts the privacy statements, but to have approximately the same shape.

6.2. Discriminative and Predictive score

Next to the statistical properties and measures, two other numerical scores obtained with machine learning techniques are described. The scores used in this thesis are the discriminative score and the predictive score, both techniques are introduced in Yoon et al. [49]. The difference between these metrics and more usual metrics (mean squared error, mean absolute error) is that the discriminative and predictive scores are based on training neural networks instead of applying a formula to the data.

6.2.1. Discriminative Score

Broadly speaking, the discriminative score is the accuracy between the correctly labeled data sets and the labeled data sets by an optimized 2-layer LSTM, see Section 4.4. The first step is to label the original and synthetic data as 1 and 0, respectively. Then split the data in a train set to train the LSTM and in a test set to check the performance of the LSTM. The optimized LSTM is trained on the train set trying to recognize the real data and the synthetic data. The goal is to have a synthetic data set with the same distributions as the original data set, so the synthetic data generation model performs best when the synthetic and the real data sets are indistinguishable. This performance is obtained when half of the test set is labeled correctly and the other half is labeled wrong. So the synthetic data generating model performs best when the accuracy is 0.5. The discriminative score then is $|0.5 - \text{accuracy}|$, and the synthetic data generating model for which the discriminative score of the real data versus the synthetic data is the lowest, is the best performing model.

6.2.2. Predictive Score

Next to the discriminative score, a predictive score can be estimated by training an 2-layer LSTM on the generated synthetic data while learning the conditional distribution of the data over time. After training the LSTM on the data distribution, the model predicts next-step values over the synthetic data sequences. In the best case, these predicted next-step values have the same conditional distribution as the real data. After generating the predicted next-step values, the output data of the LSTM is compared with the real data. This comparison is made by measuring the performance with the mean absolute error (MAE), which is $|1 - \text{probability that the two data sets have the same conditional distribution}|$. The synthetic data generating model for which the probability that the real and the synthetic data set have the same conditional distribution is the highest (so with the lowest predictive score) is the best performing model.

6.3. PCA and t-SNE visualizations

Another method to check how closely the real samples and the generated samples are distributed is by visualizations. This section describes two different dimensionality re-

duction techniques that can be used for visualizations. When both the real and synthetic data are reduced to two-dimensional data sets using the same techniques, they can be plotted in one figure. When the synthetic data and real data overlap, they follow the same distribution. With this method, it is not evident to rank the performance of the different synthetic data generating models since it is not a numeric metric. However, it checks the extent to which the two data sets overlap.

6.3.1. Principal Component Analysis (PCA)

Dimensionality reduction with PCA relies on standardizing the range of the variables and the singular value decomposition (SVD) of a matrix [25]. Say that the matrix in the situation of this thesis is an $m \times n$ matrix A, where m is the number of time steps, and n is the number of customers. For standardization, the mean μ and standard deviation σ of the features are used. The formula for this is

$$A^i_{\text{standard}} = \frac{A - \mu}{\sigma}$$

where A^i is one customer of A which has in total n customers. The SVD is a method where the matrix A is decomposed into three other matrices. Mathematically, this is given by

$$A = USV^T,$$

where U is an $m \times m$ matrix whose columns represent eigenvectors of AA^T, S is an $m \times n$ diagonal matrix with the singular values (square-root of corresponding eigenvalue denoted as d_i) on the diagonal and V is an $n \times n$ matrix whose columns represent eigenvectors of A^TA.

The SVD is applied on the covariance matrix of the standardized matrix A which is

$$cov(A_{\text{standard}}) = \frac{1}{n}\sum_{i=1}^{n}(A^i_{\text{standard}})(A^i_{\text{standard}})^T = A^TA.$$

To reduce the dimensionality of the matrix A, the k eigenvectors which represent the data best are kept and the others are dismissed. First, order the singular values d_i and to keep 99% of the variance choose k such that

$$\frac{\sum_{i=1}^{k} d_i}{\sum_{i=1}^{n} d_i} \geq 0.99.$$

In Figure 6.1, the PCA technique is illustrated. The red arrows are the two principal components, where the first principal component describes the best direction, which means that it retains the maximum variance. This is the same as the eigenvector with the highest eigenvalue d_1. The second component reflects the direction with the second maximum variance while being orthogonal to the first principal component. These two principal components form a plane that reduces dimensionality like in the second plot of Figure 6.1.

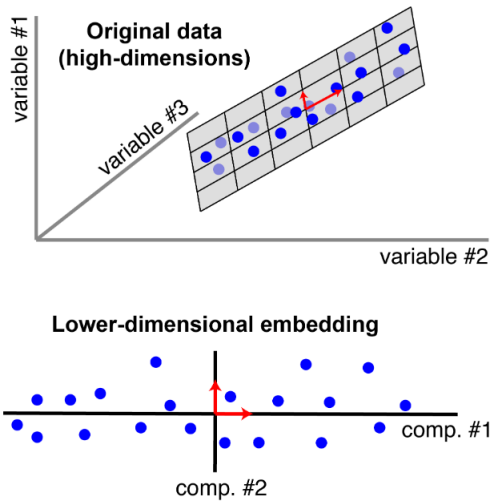


Figure 6.1.: Dimensionality reduction with PCA technique [26]

6.3.2. t-Distributed Stochastic Neighbor Embedding (t-SNE)

The method to visualize data using t-SNE is presented by van der Maaten et al. [44]. It is a technique to visualize high-dimensional data in a two-dimensional data frame by reducing the dimensionality of the sequences based on Stochastic Neighbor Embedding (SNE). This technique maps high-dimensional data in the low-dimensional space by reflecting the true distance of two points in the low-dimensional space. The aim is to match the distributions of points of the high-dimensional space and the low-dimensional space as well as possible by minimizing a cost function. A similar method of SNE is the t-Distributed SNE (t-SNE) with a cost function that differs from the one of SNE. By changing the cost function, t-SNE solves two problems of the SNE process. The first problem of SNE is that the cost function is difficult to optimize. The second problem is that SNE has a crowding problem which means that it is impossible to preserve the real distances in smaller dimensions so they have to be cramped in a smaller area. To solve this, t-SNE assumes a Student t-distribution with one degree of freedom instead of a Gaussian distribution for the computation of the distance between the low-dimensional data points [8]. This Student t-distribution is a heavy-tailed distribution that assigns higher probabilities to points that are in the low-dimensional space further away than they were in the original high-dimensional space.

To explain the algorithm of t-SNE, call x_i a data point in the high-dimensional space and $\check{x}_i$ a data point in the low-dimensional space. The joint probabilities between two data points i and j are called in the high-dimensional space p_{ij} and in the low-dimensional space q_{ij}. The conditional probability that data point x_i would pick data point x_j as its neighbor is $p_{j|i}$ and is chosen following a Gaussian distribution centered around x_i with variance σ_i

$$p_{j|i} = \frac{exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-\|x_i - x_k\|^2/2\sigma_i^2)}.$$

The variance σ_i induces a probability distribution P_i over all of the other data points and SNE searches for σ_i that produces a P_i with fixed perplexity (a measure of the number of neighbors). With this conditional probability, the high-dimensional joint probability p_{ij} is defined as the symmetrized condition probabilities

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

with n the number of data points.

As already mentioned, the low-dimensional joint probability q_{ij} is based on the Student t-distribution with one degree of freedom. The PDF of this distribution with one degree of freedom is

$$f(t) \propto C \times (1 + t^2)^{-1},$$

with C a scaling term independent of t which will be canceled out when normalizing. So the normalized joint probability between point i and j is obtained by normalizing $f(\check{x}_i - \check{x}_j)$ and is defined as

$$q_{ij} = \frac{(1 + \|\check{x}_i - \check{x}_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|\check{x}_k - \check{x}_l\|^2)^{-1}}.$$

The cost function which now has to be minimized is the Kullback-Leibler divergence between p_{ij} and q_{ij}

$$C = KL(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

with derivative

$$\frac{\delta C}{\delta \check{x}_i} = 4 \sum_j (p_{ij} - q_{ij})(\check{x}_i - \check{x}_j)(1 + \|\check{x}_i - \check{x}_j\|^2)^{-1}.$$

Then the gradient is updated with a momentum term as

$$\mathcal{X}^{(t)} = \mathcal{X}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{X}} + \alpha(t)(\mathcal{X}^{(t-1)} - \mathcal{X}^{(t-2)}), \tag{6.1}$$

with η the learning rate and $\alpha(t)$ the momentum which makes sure that the sum of previous gradients is decaying in order to find in each iteration the changes in the coordinates.

The updating step in Equation 6.1 is done T times to get a low-dimensional data representation $\mathcal{X}^{(T)} = \{\check{x}_1, \ldots, \check{x}_n\}$.

6.4. Overview metrics

In this section, an overview of all the described metrics in this chapter is given. In Table 6.1, the description per metric and a quick reminder of which values have to be achieved per metric to get the optimal model performance are summarized.

Metric	Description	Optimal value	Type
Mean and Variance	Mean and variance of the balance amount	As close to the original values as possible	Statistical measures and properties
Correlation	Correlation between balance amount time series	Closest to 1	
Entropy	Describes the amount of information and variation	As close to the original value as possible	
RMSE	Average error between balance amounts	Closest to 0	
Inter correlation	Average correlation between balance amount and count	As close to the original values as possible	
Wasserstein distance per age group	Distance function between balance amount time series per age group	Closest to 0	
CDF of the inter correlation	Probability that the inter correlation is smaller than x for $-1 \leq x \leq 1$	Same structure as original	
Autocorrelation	Degree of correlation between successive intervals	Same structure as original	
Age distribution	Distribution of the age values per data set	Same structure as original	
Discriminative score	Accuracy of labeled data sets	Closest to 0	Numerical metrics with machine learning techniques
Predictive score	Accuracy of predicted next-step values by LSTM	Closest to 0	
PCA	Dimensionality reduction with SVD	Same structure as original	Visual metrics by dimensionality reduction
t-SNE	Dimensionality reduction with t-SNE	Same structure as original	

Table 6.1.: Summary of the statistical metrics

7. Synthetic data results

In this chapter, the initial findings in the models and the results of the three machine learning models, compared to the three simulated data sets, are described. In Section 7.1, the initial findings are described after applying the models on the simulated data sets. Section 7.2 describes the results of applying the three models on the basic and intermediate data sets, this results in a conclusion about the best synthetic bank balance data generating model for the scope of this thesis. Finally, Section 7.3 describes the results of applying this best generating model to the advanced data set.

7.1. Initial Findings

Applying the three synthetic data generation models to the simulated data sets gives insight into the performance and running times of these models. First, the run time will be discussed, then some initial findings on the performance of the models will be explained.

The run time of both the PAR model and the GANs is large since these models need to train neural networks with multiple layers, and therefore, many parameters need to be determined. The GANs even have two processes to train, the discriminator and the generator. Training the models on a standard computer can take multiple weeks for the large simulated data sets in this thesis, therefore access to the Lisa Compute Cluster of SURFsara is obtained via the VU [36]. The Lisa Compute Cluster speeds up the process such that the training can be done in a couple of days. However, there are still some limitations for the Lisa cluster, these are a maximum of five days (120 hours) of training in a row and a maximum of 50.000 CPU (Central Processing Unit) training hours. The maximum of 50.000 CPU training hours is not a fundamental limitation, as 1 hour of training is equal to 8 CPU hours which means that the limit is 6.250 hours of training. The maximum of five training days in a row is a fundamental limitation when the number of training steps is getting big and when the number of input customers increases.

A summary of the run times per model with a shared partition on the Lisa Compute Cluster is provided in Table 7.1. Here, 1.000 customers are given as input to all three synthetic data generating models. The number of training steps is chosen based on the best performance in the limited time. The PAR model has a total run time of 75 hours for 2.000 training steps. From this run time, the training time takes 63 hours and the generation of 1.000 synthetic time series takes 12 hours. This large sequence generating time is a disadvantage compared with the two GAN models since these models can generate thousands of synthetic time series in a matter of seconds. The TimeGAN model

already has a considerable training time for 24.000 training steps while the documentation recommends 50.000 training steps, this is a disadvantage of TimeGAN compared to the other models. The number 24.000 is chosen since applying the model with more training steps on the simulated data sets resulted in an error. Finally, the DG model has a clear difference in training time compared to the PAR and the TimeGAN models and has therefore an advantage over those two models.

Model	**Training steps**	**Run time with 1.000 input customers**
PAR	2.000	75 hours
TimeGAN	24.000	96 hours
DG	13.000	10 hours

Table 7.1.: Training time per synthetic data generation model

The first model finding after applying the TimeGAN model to the simulated data sets is that it is not possible to add static features in the TimeGAN model created by Yoon et al. [49]. The authors explain the methodology of the general case where static features are a part of the data, but since they apply the model to stock data, which has no static features, they do not implement this in the code.

Second, the TimeGAN model does not capture the right correlations between the time series with continuous and discrete values. The course of the two time series is approximately equal, and the model did not pick up the discrete transaction count values. An example of this finding is visualized in Section B.1.

Because of these two findings of the TimeGAN model, the results of this model will only be applied to the balance amount and not to the transaction count and the static feature age. For this thesis, the code is not adjusted to these issues, which makes it harder to compare the TimeGAN model to the other two models. For now, the most important part is whether the model can capture the bank balance data structure with the seasonalities. The static features and transaction count then are only additions to improve overall performance.

The last finding is that more training steps are needed to train the PAR model than the 128 steps described in the model methodology [32]. Training with 128 training steps resulted in synthetic balance data looking like random noise time series, increasing the number of epochs increases the performance. An example of the rate at which the result increases when giving a data set with sine waves as inout in the PAR model is visualized and described in Section B.2.

Besides these three findings for the TimeGAN and PAR models, all three models perform well on trend and seasonality tests. These tests led to the used hyper parameters described in the sub-tables in Table C.1. The default values are taken of the hyper parameters not defined in these tables. Due to running time limitations, the hyper parameters are not tuned for the data set but are gathered from trial and error and other research papers ([27], [29], and [49]). The next step is to apply these models to the simulated data sets with bank balance properties.

7.2. Results basic and intermediate data sets

The performance of the three described models in Chapter 5 is tested by training these models on the *Basic data set* and the *Intermediate data set*. The performance is received by checking the results of the synthetic data sets with the metrics described in Chapter 6.

7.2.1. Basic data set

The first results are described on the data set visualized in Figure 3.2 and is called the *Basic data set*. The synthetic data results of all three models applied to this data set are first visualized with the balance data of five random clients over two years. The goal is to see the downward trend in the first year and the monthly seasonalities in the synthetic data, just as in the input data. It is important to see the monthly seasonalities since these are the salary peaks and fixed costs expenses present also in real balance data. Furthermore, the statistical, numerical, and visual metrics are checked to determine whether the distributions and correlations are identical.

The sample paths of five random customers of the synthetic data sets are visualized in Figures 7.1, 7.2, and 7.3 for the PAR model, TimeGAN model, and DG model, respectively. All three models did capture the downward trend in the first year of the time series. The PAR model seems to have difficulties in capturing the seasonalities as these are not clearly visible. The seasonalities in the synthetic data set resulting from the two GAN models are visible, here the seasonalities of the synthetic data set of the DG model are very similar to the seasonalities in the input data set.

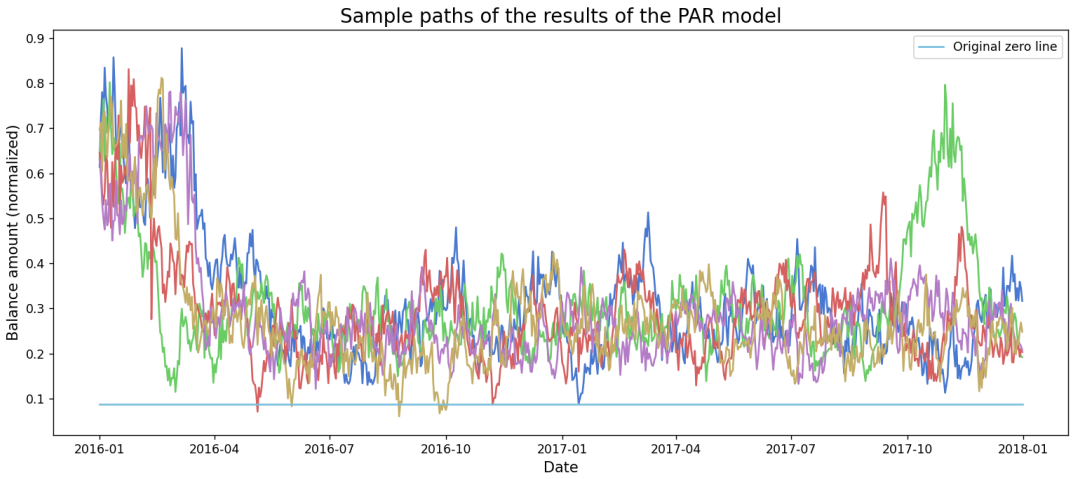


Figure 7.1.: Sample paths of five customers of the synthetic data set after applying the PAR model to the basic data set

Next to the visual inspection of the sample paths, the other metrics are presented to ensure that the models capture the distributions and correlations.

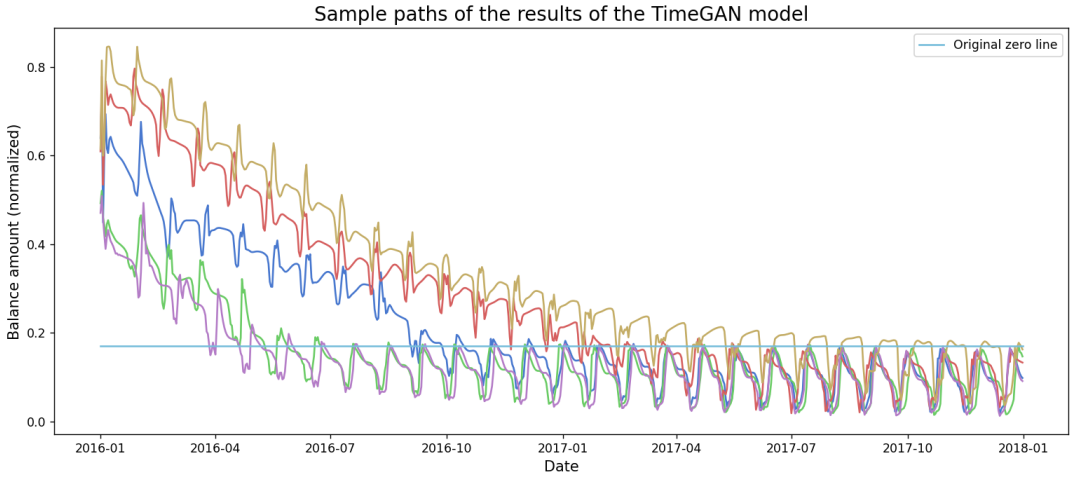


Figure 7.2.: Sample paths of five customers of the synthetic data set after applying the TimeGAN model to the basic data set

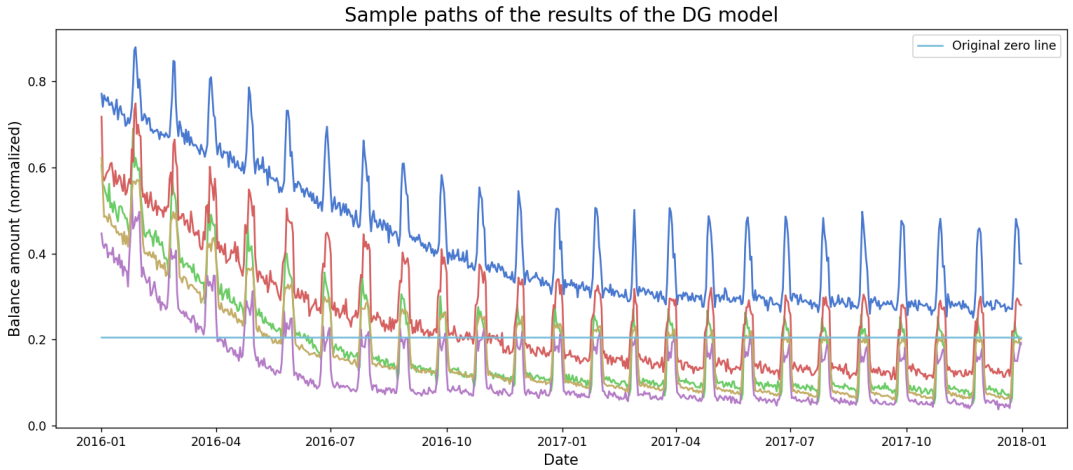


Figure 7.3.: Sample paths of five customers of the synthetic data set after applying the DG model to the basic data set

An overview of the statistical metrics per data set is given in Table 7.2. The best performing model here is the DG model since it has a high correlation and low RMSE compared to the input data set. Furthermore, the entropy is close to the entropy of the input data set. The PAR model captures the inter correlations, so the correlation between balance amount and transaction count, best. This is also visualized in Figure 7.4 where the CDF of the inter correlations resulting from the PAR model in Figure 7.4a is not perfectly the same as those of the real data, but it is in the same range of values as the CDF of the inter correlations resulting from the real data. In Figure 7.4b can be seen that the synthetic data of the DG model has higher correlations between balance amount and transaction count than the input data.

The Wasserstein distance between real and synthetic data split into age groups is

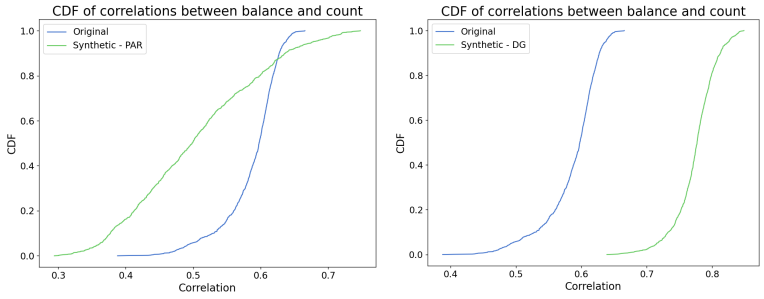
computed and denoted in Table 7.3. The distance for all three age groups is the smallest between the real data and the synthetic data of the DG model. This implies that the DG model captures the age group properties best. Next, the autocorrelation of the synthetic data sets of all three models is visualized in Figure 7.5. From these plots, the performance on seasonality can be visualized. As already seen in Figures 7.1, 7.2, and 7.3, the DG model captures the seasonalities best and the PAR model worst. Finally, the normalized age distribution of the synthetic data from the PAR model and the DG model is visualized in Figure 7.6. These plots show that the PAR model performs better in generating synthetic static feature data. In contrast, the DG model is biased towards the extremes of the static feature data.

Model	**Mean**	**Variance**	**Correlation**	**Entropy**	**RMSE**	**Inter correlation**
Original	0.1646	0.0099	1	0.4415	0	0.5865
PAR	0.2721	0.0113	0.3241	0.5974	0.1611	0.5036
TimeGAN	0.1219	0.0109	0.6969	0.4550	0.0965	-
DG	0.1834	0.0165	0.9069	0.4288	0.0872	0.7743

Table 7.2.: Statistical metrics of the basic data set and the results of the synthetic data sets per model

Wasserstein distance per age group			
Model	**age** < 25	$25 \leq$ **age** ≤ 75	**age** > 75
PAR	0.1126	0.1039	0.1173
DG	0.0564	0.0240	0.0336

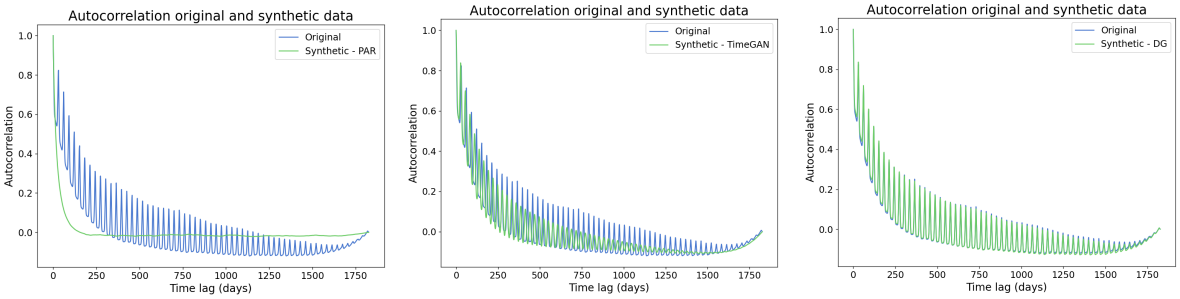
Table 7.3.: Wasserstein distance between original and synthetic data per age group



(a) PAR model (b) DG model

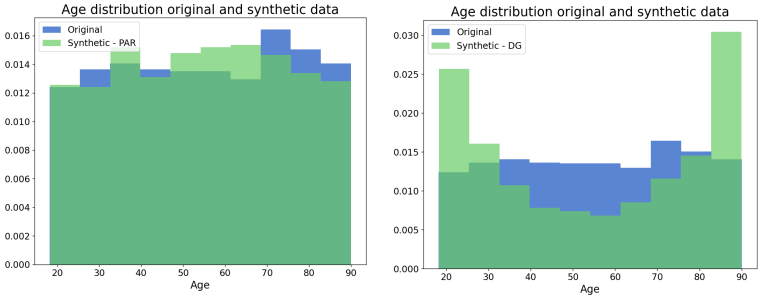
Figure 7.4.: The CDF of the correlations between balance amount and transaction count for the original data set and the synthetic data sets of the PAR and DG models

The predictive and discriminative scores between the original and synthetic data sets are denoted per data generation model in Table 7.4. The model with the lowest scores



(a) PAR model (b) TimeGAN model (c) DG model

Figure 7.5.: The autocorrelations for the original data set and the synthetic data sets of the PAR, TimeGAN, and DG models



(a) PAR model (b) DG model

Figure 7.6.: Normalized age distribution of the original data set and the synthetic data sets of the PAR and DG models

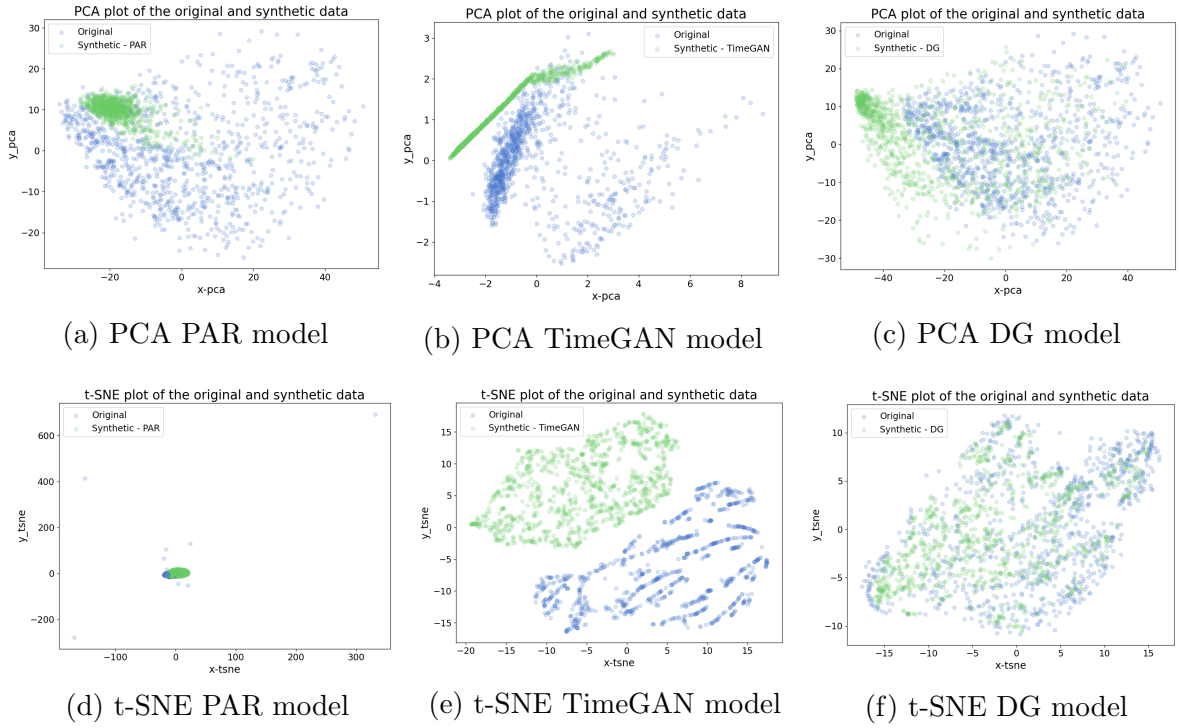
performs the best in creating synthetic data representing the input data. Again, the two GAN models have the lowest scores and thus represent the input data best since the trained neural networks had the most difficulty distinguishing between the real and the synthetic data of these two GAN models.

Model	Predictive Score	Discriminative Score
PAR	0.0564	0.4950
TimeGAN	0.0561	0.3850
DG	0.0319	0.3625

Table 7.4.: Numerical metrics of the basic data set and the results of the synthetic data sets per model

The visual metrics PCA and t-SNE of the synthetic data sets of all three models are visualized in Figure 7.7. The synthetic data sets (green dots) have the same distribution as the original data set (blue dots) if the green and blue dots are fully overlapping. In Figures 7.7a and 7.7d, the visual metrics of the PAR model are plotted; the spread of the synthetic data set is not like the spread of the original data set. For the TimeGAN

model, the distribution of values of the synthetic data does not overlap the original data set, as can be seen in Figures 7.7b and 7.7e. The DG model does return the right distribution of values in the synthetic data set as the original and synthetic data points are closely overlapping in Figures 7.7c and 7.7f.



(a) PCA PAR model (b) PCA TimeGAN model (c) PCA DG model

(d) t-SNE PAR model (e) t-SNE TimeGAN model (f) t-SNE DG model

Figure 7.7.: Visual metrics applied to the synthetic data sets of all three models against the original data set

To summarize, the synthetic data sets of the basic data set resulting from the PAR, TimeGAN, and DG models are checked on performance and compared to each other. The DG and TimeGAN models perform better than the PAR model, with DG the best results based on visual and numerical metrics. The next step is to repeat this on a data set with increased randomness, namely the intermediate data set, and check if the models can still capture correlations and distributions.

7.2.2. Intermediate data set

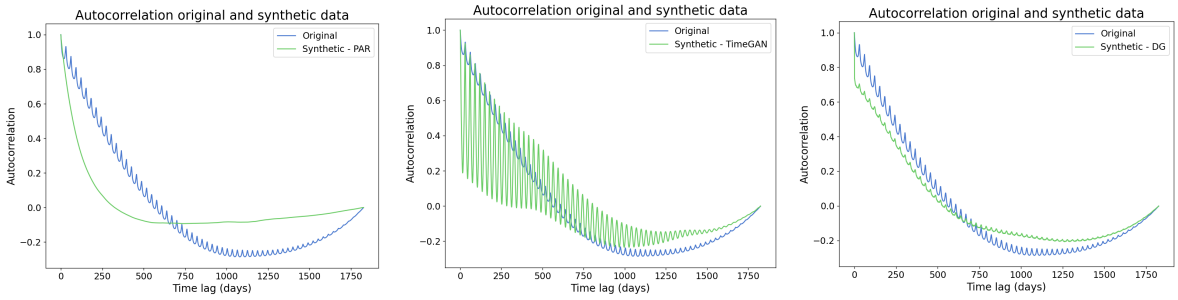
The second results are described on the data set visualized in Figure 3.3 and is called the *Intermediate data set*. The sample paths of five random customers of the synthetic data sets are visualized in the appendix in Figures D.1, D.2, and D.3. The statistical, numerical, and visual metrics are checked on synthetic data sets versus the input data set to determine if the distributions and correlations are the same. The analysis done in this subsection is exactly the same as in Subsection 7.2.1. Only the metrics where the decision of best performing model is based on are described here. These are correlation,

entropy, RMSE, predictive and discriminative score, and the autocorrelation plot. The other statistical and visual metrics can be found in Appendix D.1.

In Table 7.5, the correlation, entropy, and RMSE for the synthetic data sets of the three different data generating models and the original data are denoted. Tables D.1 and D.2 describe the remaining statistical metrics. Again, the TimeGAN and DG models perform better than the PAR model based on correlation and RMSE, and the DG model has an entropy closest to the original data set. With more randomness added, the inter correlations are captured worse by the PAR model than by the DG model. In Figure 7.8, the autocorrelations of the original data set and the three synthetic data sets are plotted. The two GAN models capture the seasonalities of the data set, while the PAR model does not capture any seasonality.

Model	**Correlation**	**Entropy**	**RMSE**
Original	1	0.2493	0
PAR	0.0517	0.3570	0.2072
TimeGAN	0.3578	0.4169	0.1622
DG	0.2662	0.2676	0.1549

Table 7.5.: Statistical metrics of the intermediate data set and the results of the synthetic data sets per model



(a) PAR model (b) TimeGAN model (c) DG model

Figure 7.8.: The autocorrelations for the original data set and the synthetic data sets of the PAR, TimeGAN, and DG models

Last, the numerical metrics predictive score and discriminative score are computed for the three models, and the results are described in Table 7.6. The DG model performs best, but the values are close to the results of the TimeGAN model. The visual metrics PCA and t-SNE are visualized in the appendix in Figure D.6. Again, the PCA and t-SNE visualizations of the DG model are the most overlapping.

The results described until here conclude that both GAN models have superior performance in simulating bank balance time series data over the PAR model. The DG model is performing best out of the two GAN models, this can be a result of the shorter runtime of the DG model, which results in the opportunity to implement enough training steps. Also, the TimeGAN model can not cope with the static and discrete dynamic

Model	Predictive Score	Discriminative Score
PAR	0.1428	0.4925
TimeGAN	0.0873	0.4800
DG	0.0734	0.4425

Table 7.6.: Numerical metrics of the basic data set and the results of the synthetic data sets per model

features in this case, making the model less useful. When the code is adjusted such that the TimeGAN model can cope with these features, and more computing time is available to add extra training steps and extra input data, the TimeGAN model could also be of interest for performing synthetic bank balance data. Since the computing time to improve TimeGAN is limited for this research and the DG model has the best results, the DG model is the model applied to the advanced data set with the groups and economic crisis event in the next section.

Before applying the DG model to the advanced data set, a couple of tests are done to see if the performance of the DG model can be improved. In Figure 7.9, the balance amount of five random customers of the synthetic data set resulting from the DG model is visualized over time. As seen in this plot and also in the autocorrelation plot in Figure 7.8c, the seasonalities are not clearly visible, so an essential property of bank balance data is not captured well by the model.

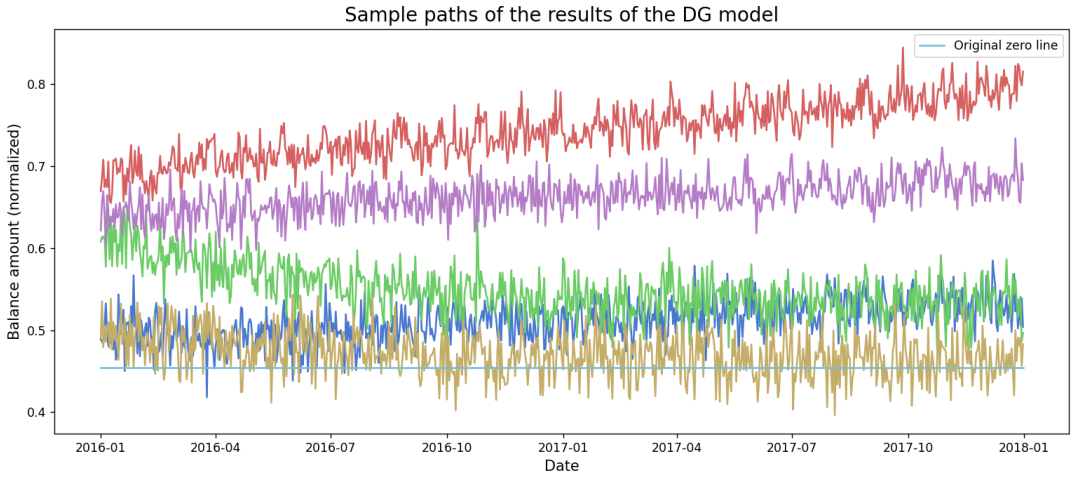


Figure 7.9.: Sample paths of five customers of the synthetic data set after applying the DG model to the intermediate data set

Three adjustments to the DG model are made to check if the performance improves. First, the number of input customers is increased to 10.000, call this model DG - 10.000 and the previous one DG - 1.000. Increasing the number of customers can compensate for the increase in randomness. More customers means more groups in the data with the same pattern and more data for the model to learn from. The second adjustment is changing one of the parameters, namely increasing the number of layers in the LSTM

to generate dynamic features. The number of layers is increased from default value 1 to 3. The aim of more LSTM layers is to create more accuracy in learning since the neural network will be deeper. The last adjustment is adding extra training steps, from 13.000 to 20.000. Extra training steps means extra gradient updates performed in the neural network, which means extra learning about the data distribution, this could improve performance if the current number of training steps is not yet enough. In each adjusted DG model, all features and hyperparameters are the same as in the original model (DG - 1.000) except for the one feature changed.

In Table 7.7, the correlation, entropy, and RMSE are denoted to compare the results of the synthetic data from the adjusted models to the results of the DG - 1.000 model to see if any adjustment improves the synthetic data results. Tables D.3 and D.4 describe the remaining statistical metrics. The predictive and discriminative scores of the adjusted DG models are described in Table 7.8. In Figure 7.10, the autocorrelation of the original data and of the synthetic data sets are visualized. The other visualizations can be found in Appendix D.2.

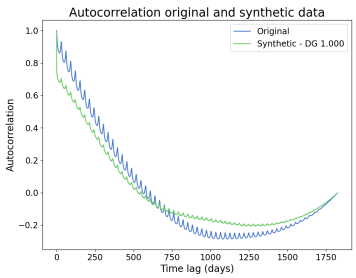
Model	**Correlation**	**Entropy**	**RMSE**
Original	1	0.2493	0
DG - 1.000	0.2662	0.4676	0.1549
DG - 10.000	0.3412	0.2706	0.1524
DG - extra layers	0.1770	0.4093	0.2222
DG - extra steps	0.1820	0.5195	0.1640

Table 7.7.: Statistical metrics of the intermediate data set and the results of the synthetic data sets per different DG model

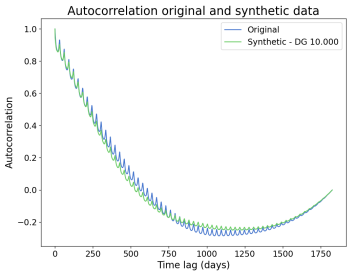
Model	**Predictive Score**	**Discriminative Score**
DG - 1.000	0.0734	0.4425
DG - 10.000	0.0674	0.1700
DG - extra layers	0.0716	0.5000
DG - extra steps	0.0757	0.4775

Table 7.8.: Numerical metrics of the basic data set and the results of the synthetic data sets per different DG model

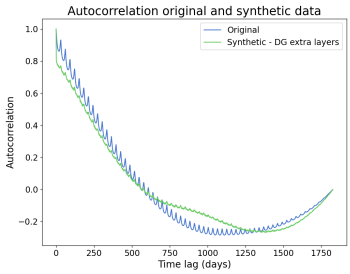
From these results can be concluded that adding extra customers as input data improves the synthetic data generation performance of the DG model as the correlation, RMSE, entropy, autocorrelation, and the predictive and discriminative scores improve. The adjusted DG model with extra layers in the LSTM network did not improve the performance, this is a result of more parameters that must be trained by the model when adding extra layers. More training steps could be needed when more parameters have to be trained. The adjusted DG model with extra training steps also does not improve the performance, in each training step a loss is computed and if the loss is already minimized



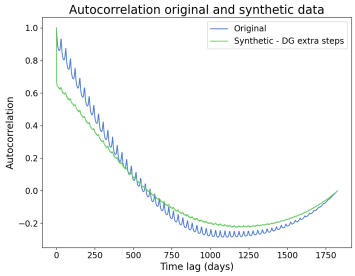
(a) DG - 1.000



(b) DG - 10.000



(c) DG - extra layers



(d) DG - extra steps

Figure 7.10.: The autocorrelations for the original data set and the synthetic data sets of the adjusted DG models

after a certain amount of training steps, the performance will not improve anymore by increasing the number of training steps.

In Figure 7.11, the balance amount of five random customers from the DG - 10.000 synthetic data set is visualized over time. The seasonalities in this plot are clearly visible, confirming the performance increase when increasing the number of input customers. The most evident difference between the input data set and the synthetic data set is the high volatility in the synthetic data set, the volatility decreases when the number of input customers increases. When increasing the number of input customers from 1.000 to 10.000, the training time increases with factor 10 as well, so it becomes 100 hours of training (see Table 7.1).

Because of the good performance of the DG - 10.000 model, the synthetic data of the advanced data set is generated with this model and will be decomposed in the modeling part to find features and groups. However, first the results of the advanced data set are described in the next section.

7.3. Results advanced data set

The only model applied to the *Advanced data set* visualized in Figure 3.4 is the DG model with 12.000 customers as input. As shown in the previous section, increasing the number of input customers increases the model performance. Because of that, the number of input customers is chosen to be as high as possible in the limited training

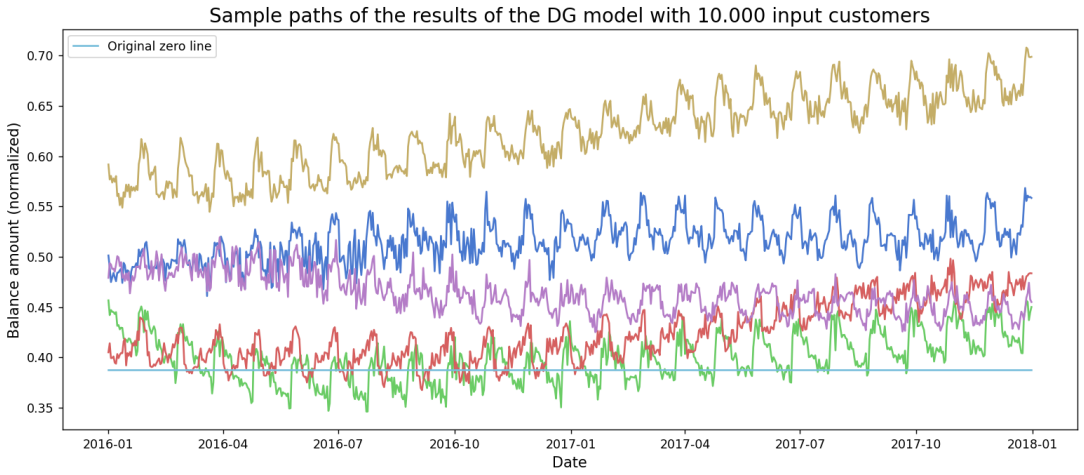


Figure 7.11.: Sample paths of five customers of the synthetic data set after applying the DG model to the intermediate data set with 10.000 customers as input

time, which is equal to 12.000 customers. The same metrics as in the previous sections of this chapter are applied to the synthetic data set and can be compared to the results of Section 7.2 as a reference if the model works well on this last data set.

In Figure 7.12, the sample paths of the balance amount of five random synthetic customers are plotted for two years. The seasonalities are visible, but the volatility is again clearly higher than in the original data set in Figure 3.4.

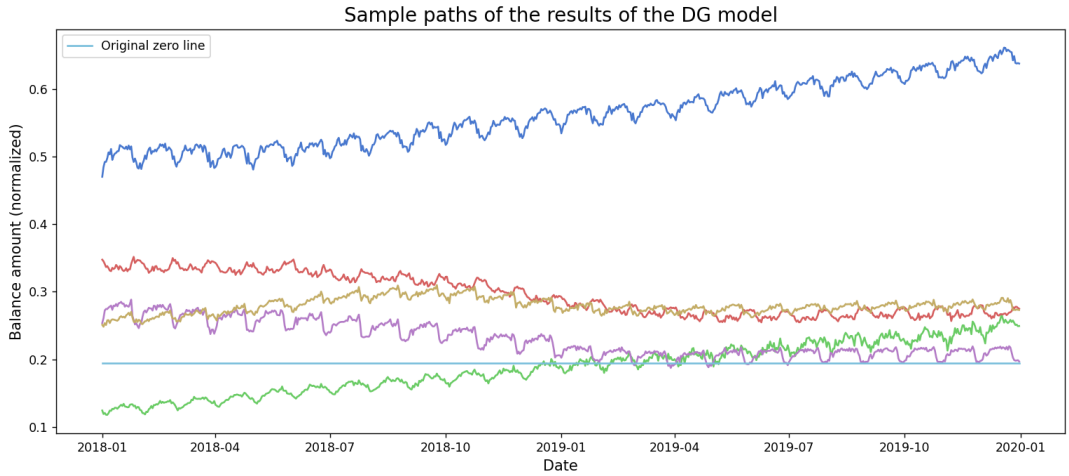


Figure 7.12.: Sample paths of five customers of the synthetic data set after applying the DG model to the advanced data set with 12.000 customers as input

The statistical metrics are summarized in Table 7.9. Compared with the synthetic data of the DG - 10.000 model of Section 7.2, the correlation and RMSE of the DG model in Table 7.9 are of the same order of performance. Also, the entropy of the original and synthetic sets are almost alike. The Wasserstein distance per age group in Table 7.10

is low for each group which means that the DG model captures the differences in each group well.

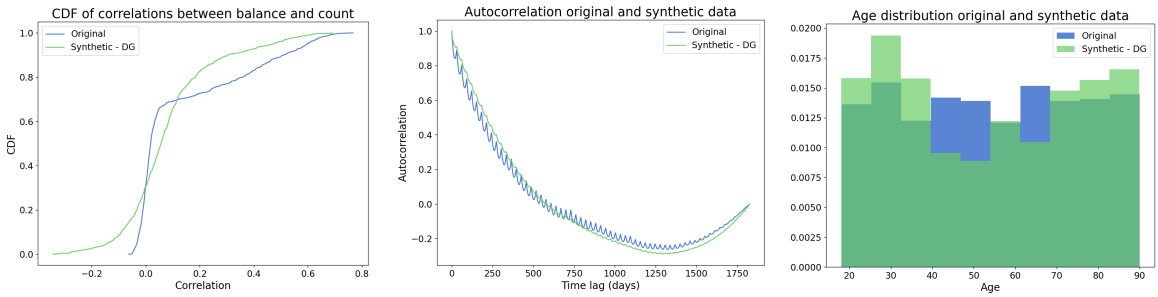
The visual statistical metrics are visualized in Figure 7.13. From these plots, it can be concluded that extra performance can be gained. The CDF of the correlations between the balance amount and the transaction count visualized in Figure 7.13a are in a broader range for the synthetic data compared with the original data. The autocorrelation visualized in Figure 7.13b has clearer seasonalities for the original data than for the synthetic data. Finally, the age distribution in Figure 7.13c shows that the age distribution of the synthetic data is again slightly biased towards the extremes. In contrast, the age distribution of the original data is more uniformly distributed.

Model	**Mean**	**Variance**	**Correlation**	**Entropy**	**RMSE**	**Inter correlation**
Original	0.3069	0.0162	1	0.2862	0	0.1284
DG	0.3106	0.0215	0.3189	0.2766	0.1614	0.1543

Table 7.9.: Statistical metrics of the advanced data set and the results of the synthetic data set from the DG model

Wasserstein distance per age group			
Model	**age** < 25	$25 \leq$ **age** ≤ 75	**age** > 75
DG	0.0617	0.0198	0.0423

Table 7.10.: Wasserstein distance between original and synthetic data per age group



(a) CDF of correlations (b) Autocorrelations (c) Age distributions

Figure 7.13.: Visuals of the statistical metrics for the original data set and the synthetic data set of the DG model

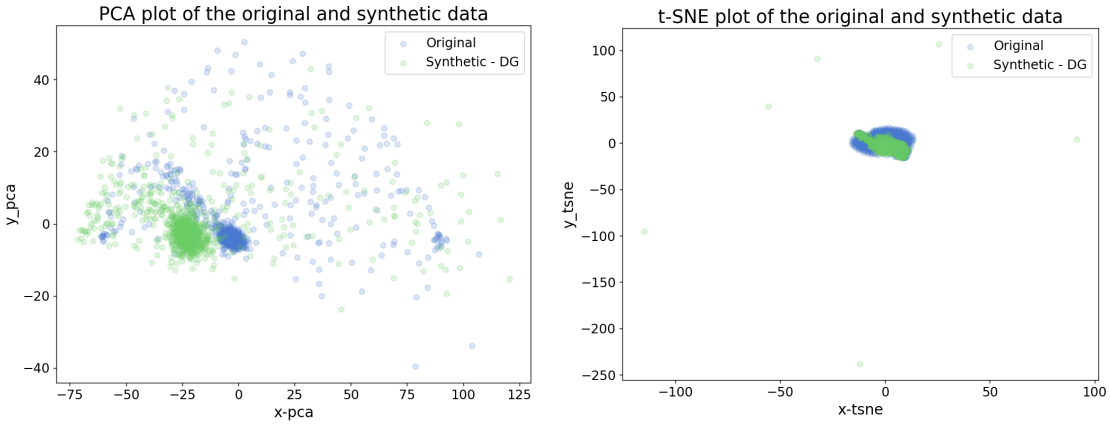
The predictive and discriminative scores representing the differences found by machine learning models between the advanced data set and the synthetic data set generated by the DG model are shown in Table 7.11. These scores are better compared to the scores of the DG - 1.000 model in Table 7.8. However, the discriminative score of the DG - 10.000 in Table 7.8 is much better than the discriminative score in Table 7.11, so improvement

can still be gained by adding more customers as input in the DG model when having enough computing time since this data set has again more randomness.

In Figure 7.14, the dimensionality reductions of the original and synthetic data sets are visualized in a PCA and a t-SNE plot. The PCA plot shows that both data sets are in the same range of values with approximately the same spread. The t-SNE plot shows the same but the synthetic data has some significant outliers, which makes it harder to compare the two data sets.

Model	Predictive Score	Discriminative Score
DG	0.0688	0.4257

Table 7.11.: Numerical metrics of the advanced data set and the results of the synthetic data set from the DG model



(a) PCA DG model

(b) t-SNE DG model

Figure 7.14.: Visual metrics applied to the synthetic data set of the DG model against the original data set

Finally, an economic crisis event was added to the advanced data set, described in Section 3.3. Features of this economic crisis event are a downward trend in 2018 compared to the years 2017 and 2019 since the fixed costs in 2018 increased due to the economic crisis. Also, the average balance amount is smaller in 2018 and 2019 compared to 2017 and 2020 since customers have more chance to lose (partly) their job in 2018 and that can still be going on in 2019. It is important for the model to capture differences in a specific time period since bad economic times can give another value to the risk of a customer.

Tables 7.12 and 7.13 summarize the average trend and balance amount per year of the original data set and the synthetic data set, respectively. The average trend in 2018 is going down for both data sets and the average balance amount is lower in 2018 and 2019 compared to 2017 and 2020 again for both data sets. These results show that the DG model captures the differences between the years due to the simulated economic crisis event and implements this in the synthetic data set as well.

Original data	**2017**	**2018**	**2019**	**2020**
Average trend	0.000204	−0.000358	0.000305	0.000261
Average balance amount	0.4171	0.3742	0.3661	0.4290

Table 7.12.: Average trend and balance amount per year of the original data set

Synthetic data	**2017**	**2018**	**2019**	**2020**
Average trend	0.000187	−0.000306	0.000202	0.000182
Average balance amount	0.4527	0.4057	0.3964	0.4567

Table 7.13.: Average trend and balance amount per year of the synthetic data set

8. Synthetic data modeling

Until this point, only the theoretical side of generating synthetic data with properties of bank balance data is exhibited. An interesting question left is how banks can use synthetic bank balance data to improve their models and whether synthetic data is representative enough to use in this situation. Therefore, an application of the use of synthetic bank balance data for a bank is investigated in this chapter, thereby giving an answer to research question 2 in Section 1.1. Section 8.1 summarizes how synthetic data can be used in improving risk models in banks. Sections 8.2 and 8.3 describe the results of two methods of clustering data to label the customers.

8.1. Improving risk models with synthetic data

As already mentioned before, banks want their models to be improved and one of the options to do this is to find out whether balance data can be a part of that. One of the possibilities for improving the models is to add features based on balance data. To give insights into the features of the synthetic data set, modeling adjustments are made. One of the many possibilities of modeling adjustments is to cluster the customers in groups based on the bank balance data and to give each customer a label based on their group. This label is then given as a feature as model input and ideally can indicate the level of risk of a customer. The clustering of time series data can be done with multiple techniques. It is, however, time-consuming when many customers are in the data set with multi-dimensional time series. Other possibilities of modeling adjustments indicating risk are giving customers an indicator of a negative balance amount and giving each customer a volatility level. In this thesis, the focus is on clustering the full data set and experimenting with the possibilities of clustering.

The clustering is done on both the advanced data set and the synthetic data set resulting from the DG model in Section 7.3. The clusters of these two data sets will be compared to determine whether the same clusters will be found in the original data set as in the synthetic data set and what the properties of the clusters are. The results of two clustering techniques to find representative groups are described in this chapter. The two clustering techniques are *TimeSeriesKMeans* for the time series data set and *Kmeans* after creating a static data set out of the time series data set. Both techniques use the K-Means clustering technique, which is explained in Section A.3.

The number of clusters can be chosen as input in both techniques. Normally, the number of clusters in the data set is unknown. However, for illustration purposes the knowledge of the original data set is used and the number of clusters chosen are two and six. These two options are chosen since the advanced data set simulated in Section 3.3 is

simulated to have two customer groups indicating low and high risk, respectively; stable and unstable. Also, differences are made between age groups, so differences between customers younger than 25, between 25 and 75, and older than 75 can be found. Since these age groups also have stable and unstable customers, a total of six groups can be created.

8.2. K-Means clustering on static data set

First, a static data set is created out of the synthetic data set resulting from the DG model by aggregating features per customer. The static data set has 12.000 customers with four features each, namely age, average transaction count, average volatility, and the correlation between the balance amount and the transaction count. These features can differ in practice when the goal of grouping the data set differs from the goal of grouping the data set based on the level of risk of the customers in this case. Grouping is done with K-Means clustering without considering age. Age is left out in this process since clustering with age as a feature resulted in equally divided groups only based on age, which is a result of age being uniformly distributed in the data. With these results, the stable and unstable customers could not be identified. Hence, age is left out as a feature for the K-Means algorithm. However, age is used as a statistic to determine if the different clusters have differences in age.

The results are split into two paragraphs, first the results of clustering into two groups are described and then the results of clustering into six groups.

Results two clusters The results of clustering the static original data set and the static synthetic data set into two groups are summarized in Tables 8.1 and 8.2, respectively. Both data sets have one big cluster and one small cluster where the small cluster has a high average transaction count and high volatility. The two clusters of the original data set have a more clear difference in volatility while the two clusters of the synthetic data set have volatilities close to each other. This is a result of the synthetic data still being very volatile overall because of the lack of enough input customers.

The features of the unstable customers are a high average transaction count and high volatility. The high volatility is a result of unstable customers having a higher probability of losing their job and because of more transactions each day. Also, the unstable customers are approximately 10% of the total portfolio. The latter is not exactly the case after clustering but it can be hard to find all the unstable customers since they only have more chance to lose their job and more chance to have many transactions. Therefore the differences between stable and unstable customers can, in some cases, be really small and hard to distinguish for the clustering algorithm.

The clusters are visualized in t-SNE plots (see Chapter 6.3) for the original data in Figure 8.1a and for the synthetic data in Figure 8.1b. The two small clusters are for both data sets in the same range and have the same shape. Note that the data dimensionality is reduced, making it intuitively harder to see why the algorithm makes these two clusters. The big difference is the spread of the points between the two data

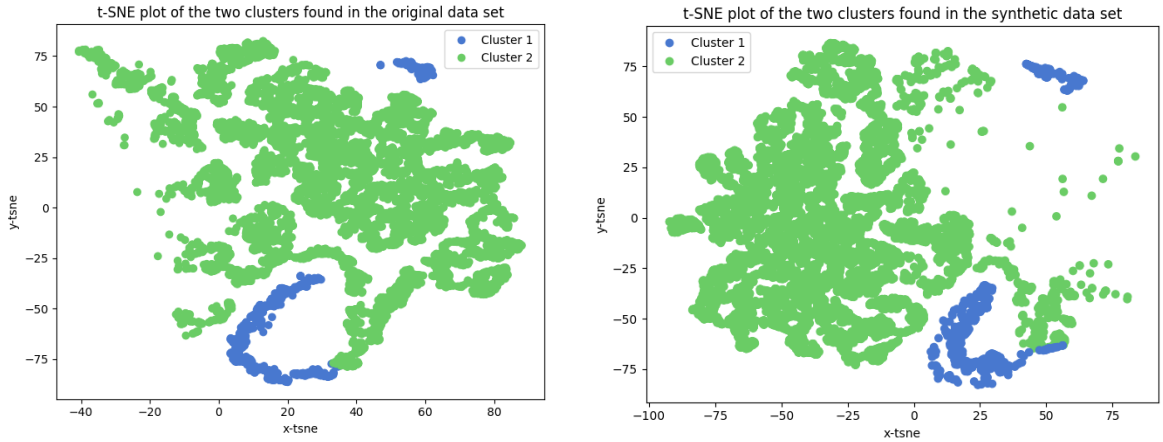
sets, the synthetic data has a broader range of values on the x-axis which is a result of the high volatility in the data set.

Group	Size	Average age	Volatility	Average transaction count
1	950	32.2	0.0112	6.02
2	11050	56.4	0.0050	2.92

Table 8.1.: Two clusters of the static original data set

Group	Size	Average age	Volatility	Average transaction count
1	844	42.1	0.0216	6.03
2	11156	63.2	0.0192	2.06

Table 8.2.: Two clusters of the static synthetic data set



(a) t-SNE of the original data

(b) t-SNE of the synthetic data

Figure 8.1.: t-SNE plots of the original data and the synthetic data per cluster

The two clusters created with the K-Means algorithm have differences in age, volatility, and average transaction count in both data sets. The small cluster with low average age, high volatility, and high average transaction count meets the features used to simulate the group with unstable customers. It can not be concluded with full certainty that this cluster is precisely the unstable customer group. Some extra data analysis is done to give more insight into how these clusters are created and whether these clusters can be labeled as unstable and stable.

One of the features of unstable customers is high volatility, so the original static data set is sorted by volatility and the 1.000 customers with the highest volatility are selected. This is done to check whether the statistics of this high volatility data set deviate from the full data set and if this confirms the thoughts of the small cluster being the group with unstable customers. The statistics of the high volatility customer group compared

to the full data set are described and visualized in Table E.1 and Figure E.1. The result is that customers with high volatility have an average transaction count of 3.6 and that this count is less concentrated around the total average value 3. This is not as high as the average transaction count in the cluster (which is 6.02). The age is for the top 1.000 volatile customers on average 35 with less variation, this is also seen in the cluster created by the K-Means algorithm.

Another feature of unstable customers is a high average transaction count, so the original static data set is again sorted, but this time by average transaction count. The 1.000 customers with the highest average transaction count are selected and checked on statistics compared to the full data set. These statistics are described and visualized in Table E.2 and Figure E.2. The customers with high transaction counts have more spread in the volatility with an average of 0.0097, while the full data set has a volatility centered around 0.0051. These volatility values are again not as extreme as those found in the clusters in Table 8.1. Again the age is for the top 1.000 highest transaction count customers on average 34 with less variation, this is also seen in the cluster created by the K-Means algorithm.

This concludes that the clustering of the static data depends on both the volatility and the transaction count having extreme values and leaving out the customers with extreme volatility but a less extreme average transaction count or the other way around. This gives more certainty of the clusters being separated into risky and less risky customers since the risky customers have features that are both extreme in volatility and average transaction count.

Results six clusters The results of clustering the static original data set and the static synthetic data set into six groups are summarized in Tables 8.3 and 8.4, respectively. When clustering the two data sets into six groups, the results of the two data sets are not that alike anymore compared to the two clusters seen before. Even though, features of the stable and unstable customers can be found here as well.

The results of the original data set summarized in Table 8.3 are discussed first. What is clear in these results is that the three groups with the highest volatility (groups 1, 2, and 3) also have the highest average transaction count. These results are less clear in the average age split, the expected split is between customers younger than 25, between 25 and 75, and older than 75 because of the differences between those age groups in the simulation. The clusters found in Table 8.3 are two groups younger than 35, two groups between 35 and 60, and two groups older than 60. Each of these age groups has one cluster with high volatility and high transaction count and one group with the opposite.

Next, the results of the synthetic data set are summarized in Table 8.4. In these results, the differences between the volatilities of the groups are less clear but again the three groups with the highest volatility (groups 1, 3, and 4) also have the highest average transaction count. Also, the age differences are not as expected but there can be made a distinction between young (younger than 35), average (between 35 and 70), and old (older than 70), with each age group again one cluster with high volatility and high transaction count and one group with the opposite. The split in ages is different

from the split in ages of the original data set.

When comparing Tables 8.3 and 8.4, it again stands out that the volatility differences in the synthetic data are not as clear as in the original data set. For visual comparison, the clusters are plotted again in t-SNE plots. Figure 8.2a visualizes the six clusters in the original data set and Figure 8.2b visualizes the six clusters in the synthetic data set. The clusters in the synthetic data set are not as clean as the clusters in the original data set as can be seen by the mixed light-blue and red points.

Group	Size	Average age	Volatility	Average transaction count
1	355	27.5	0.0094	7.20
2	456	44.1	0.0127	5.52
3	594	73.8	0.0104	4.19
4	736	34.2	0.0084	1.15
5	1291	50.4	0.0061	2.27
6	8568	60.1	0.0043	3.11

Table 8.3.: Six clusters of the static original data set

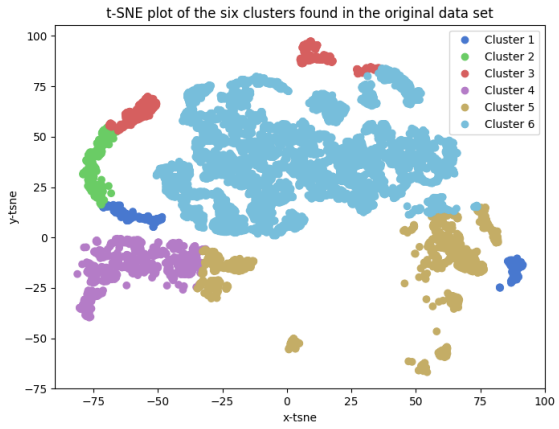
Group	Size	Average age	Volatility	Average transaction count
1	378	23.4	0.0220	7.25
2	389	84.1	0.0157	0.71
3	589	72.8	0.0199	3.45
4	1063	60.1	0.0217	5.20
5	1164	33.6	0.0178	1.55
6	8417	47.9	0.0198	2.22

Table 8.4.: Six clusters of the static synthetic data set

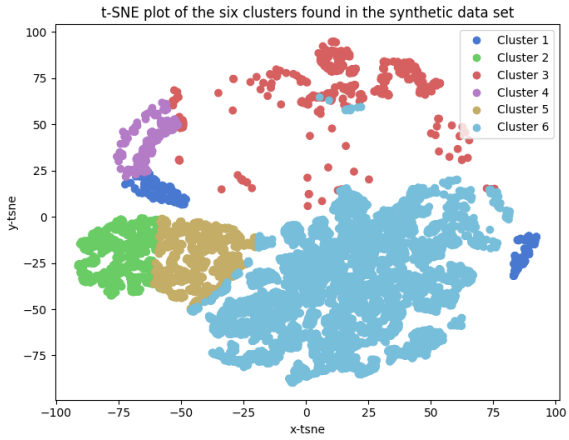
8.3. K-Means Time Series clustering

In this section, the advanced data set and the synthetic data set resulting from the DG model are both given as inputs in the K-Means Time Series clustering model. This type of clustering is more difficult since way more information to create clusters is available when the data set is not modified. Again, age is left out in the clustering process because the resulting equally divided groups are only based on age. The results are again split into two paragraphs, first the results of clustering into two groups are described and then the results of clustering into six groups.

Results two clusters The results of clustering the dynamic original data set and the dynamic synthetic data set into two groups are summarized in Tables 8.5 and 8.6, respectively. Again, in these two tables, one large cluster and one small cluster for



(a) t-SNE of the original data



(b) t-SNE of the synthetic data

Figure 8.2.: t-SNE plots of the original data and the synthetic data per cluster

both data sets are found. However, a remarkable finding is that the small cluster of the original data is larger than that one of the original data in Section 8.2. Further, the differences between the groups based on volatility and average transaction count are less clear than in the previous section. Also, the differences between the clusters of the original data and the clusters of the synthetic data are more significant. The differences between the clusters in the previous section and the clusters in this section result from more information being available in time series data compared to the static data. To get more insight into these differences, some metrics are visualized per data set per cluster. These metrics are autocorrelation, the CDF of the correlations between balance and transaction count, and a t-SNE plot, described in Chapter 6. The visualizations per cluster of the original data set can be found in Figure 8.3 and those of the synthetic data set in Figure 8.4.

Also from these figures can be concluded that the clusters of the original data set and those of the synthetic data set are not the same. In the original data set clusters, the two clusters have clear differences between the autocorrelations and the inter-correlations. In the synthetic data set clusters, the autocorrelations and inter-correlations of the two clusters are almost equal.

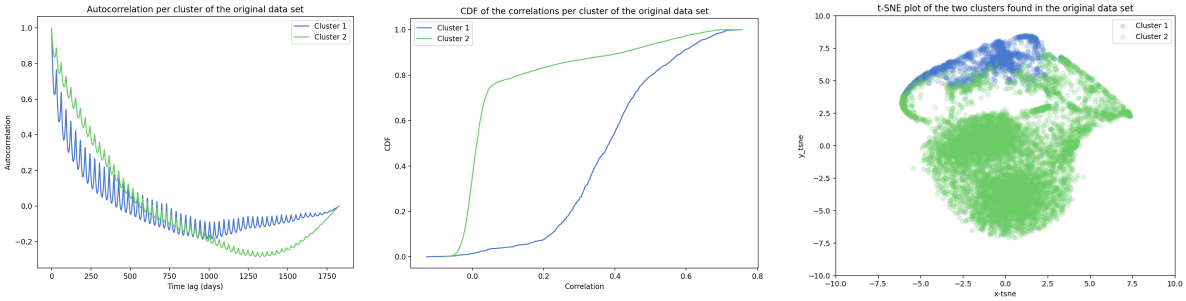
These differences between the original and the synthetic data set are a result of the synthetic data set resulting from the DG model needing to be trained better to represent the original data.

Group	Size	Average age	Volatility	Average transaction count
1	1894	44.1	0.0068	3.42
2	10106	56.4	0.0053	1.81

Table 8.5.: Two clusters of the original data set

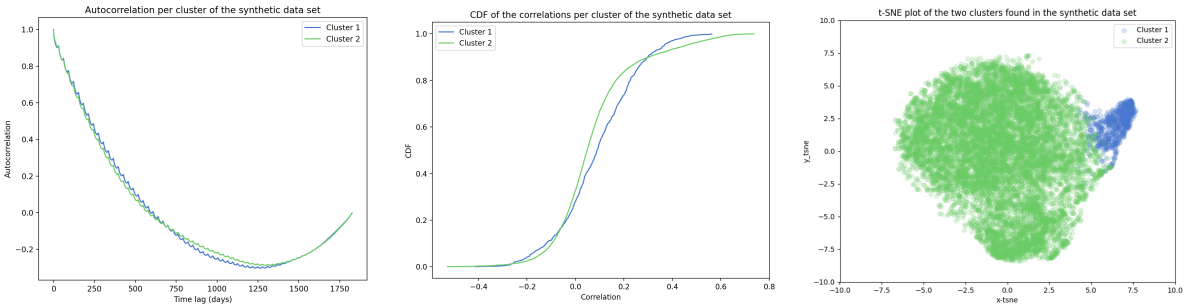
Group	Size	Average age	Volatility	Average transaction count
1	876	40.6	0.0216	5.95
2	11124	65.5	0.0192	2.05

Table 8.6.: Two clusters of the synthetic data set



(a) Autocorrelations (b) CDF of correlations (c) t-SNE

Figure 8.3.: Visuals of the differences between the two clusters in the dynamic original data set



(a) Autocorrelations (b) CDF of correlations (c) t-SNE

Figure 8.4.: Visuals of the differences between the two clusters in the dynamic synthetic data set

Results six clusters The results of clustering the dynamic original data set and the dynamic synthetic data set into six groups are summarized in Tables 8.7 and 8.8, respectively. The two groups with the highest volatility also have the highest transaction count, which are groups 1 and 2 for both data sets. However, the average age of these two groups is different for the original data compared to the synthetic data. Again, the age distribution between the clusters is not clearly separated into young, average, and old, as expected. The groups of the original data set are more average aged while those of the synthetic data set are higher. The visualizations of the clusters of the original data set in Figure 8.5 show the differences in autocorrelation and inter correlations of the different clusters. Just as for the two clusters in Figure 8.4, the clusters of the synthetic data set visualized in Figure 8.6 do not show clear differences in autocorrelation and inter correlations. Again it turns out that clustering into six groups does not give the

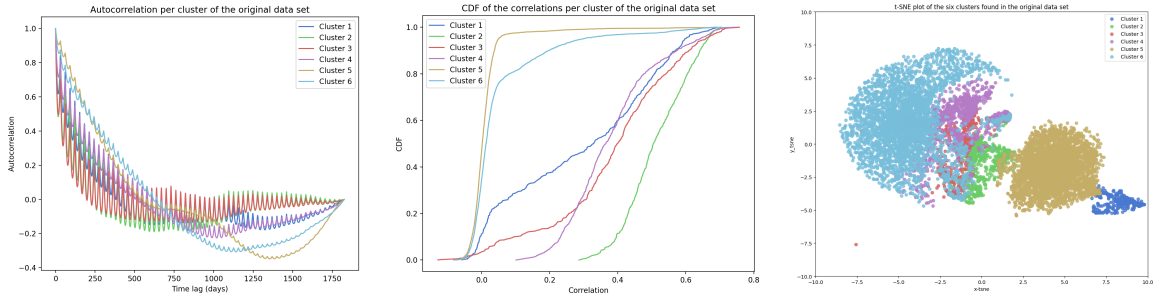
expected groups and that creating groups with clear differences for the synthetic data set is hard.

Group	Size	Average age	Volatility	Average transaction count
1	544	28.9	0.0109	6.77
2	738	36.2	0.0110	4.55
3	808	34.1	0.0084	1.22
4	1228	50.8	0.0058	2.32
5	4311	58.1	0.0047	3.12
6	4372	62.0	0.0041	3.12

Table 8.7.: Six clusters of the original data set

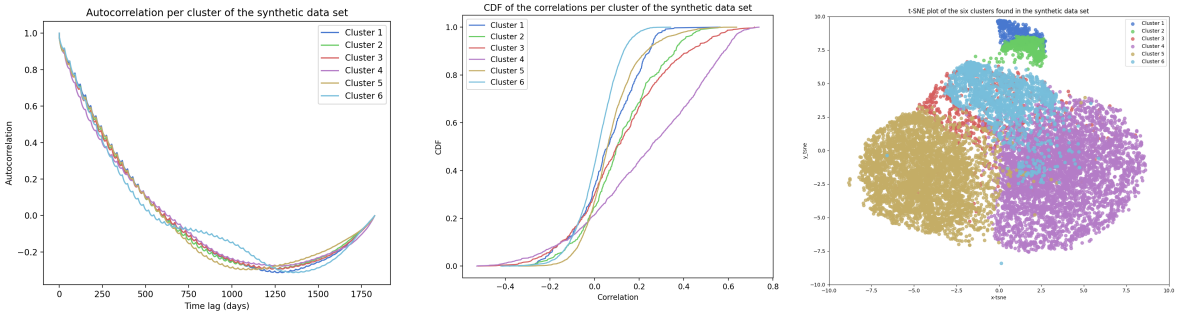
Group	Size	Average age	Volatility	Average transaction count
1	372	25.3	0.0218	7.25
2	423	79.9	0.0216	5.13
3	621	70.2	0.0199	3.30
4	1719	59.8	0.0164	0.98
5	4250	40.2	0.0206	2.18
6	4615	44.8	0.0189	2.20

Table 8.8.: Six clusters of the synthetic data set



(a) Autocorrelations (b) CDF of correlations (c) t-SNE

Figure 8.5.: Visuals of the differences between the six clusters in the dynamic original data set



(a) Autocorrelations (b) CDF of correlations (c) t-SNE

Figure 8.6.: Visuals of the differences between the six clusters in the dynamic synthetic data set

9. Conclusion

In this thesis, research on the generation of synthetic time series data with properties of bank balance data is done. Synthetic data solves the problem of restrictions in sharing the confidential data of banks. Three machine learning models, the PAR, TimeGAN, and DG models, are trained on self-simulated bank balance data sets and the synthetic data results are compared to each other with various metrics. Out of the three machine learning models, the DG model performs best in this scope based on statistical, numerical machine learning, and visual metrics. The PAR model performs best in simulating the static features and the properties corresponding to these static features. However, the model has difficulties in capturing the monthly salary and fixed costs seasonalities. Since these seasonalities are an important feature of balance data, the final synthetic data set is not simulated with the PAR model. The TimeGAN model does capture the seasonalities well but can not handle the static features and discrete transaction count data. The discrete transaction count data is generated as a continuous time series with the same course as the synthetic balance time series. This limits the possibilities to generate synthetic data with the TimeGAN model to only balance data which is the reason that the final data set is not simulated with the TimeGAN model. The DG model has the advantages that it can handle the static features and discrete transaction count data, and it can capture the seasonalities well. Because of this, and because of the best metric results, the DG model performs best in this scope and is used to simulate the final data set. Especially with less randomness in the data, the DG model can capture correlations, distributions, and seasonalities well. However, when the randomness is increased, it is harder for the model to find patterns in the data. A way to solve this is to increase the number of input customers and thereby increase the information for the model to learn from. Increasing the input customers from 1.000 to 10.000 clearly improved performance in correlation, seasonalities, and entropy, but still, more customers are needed as input to generate a proper synthetic data set. The exact number of input data points needed depends on the randomness in the data set, the length of the time series, and the number of input data points. To determine how much input customers are needed, one needs more computing time than what was available for this thesis.

After generating a synthetic data set out of the advanced data set, an application with balance data is done to find out how synthetic balance data could be used in risk models of banks. Multiple applications of synthetic balance data could be applied in banks, one of them is clustering the synthetic data set into groups. Based on the generated clusters, the customers get a label that can be given as input in a financial risk model. When this label gives an extra indication of risk, the current risk models can be improved by this clustering. The synthetic data is clustered with two different techniques, namely K-Means clustering for a static data set and K-Means Time Series clustering for the

dynamic data set. Ideally, the groups are relatable to the simulated groups in the original data set, and stable and unstable customers can be found. It turned out that the clustering of the static data sets gave two groups with features comparable to the features of stable and unstable customers, while clustering of the static data into six groups resulted in groups that could not be linked to the simulation well. The clustering of the dynamic data in two groups did not have these clear differences for stable and unstable customers, and again the six groups could not be linked to the simulation well. This results from the K-Means Time Series clustering method not finding correct differences between the time series in a data set. Also, the synthetic data set is not developed well enough to have the same features as the original data set, which resulted in clusters with no clear differences in autocorrelation and inter-correlations for the synthetic data set.

9.1. Further research possibilities

Both extensions of the GAN model (TimeGAN and DG) have in a certain way good results in generating synthetic time series data with bank balance properties. For the TimeGAN model, improvements can be obtained by carrying out three different improvement steps. The first one is to implement more training steps trying to minimize the loss, this is possible when enough computing time is available. Second, implementing the static features in the code can increase performance. Last, hyper parameter tuning can be done to find out which combination of hyper parameters results in the minimum loss. This last opportunity holds for the DG model as well, improvements could be obtained when the hyper parameters are tuned and optimized. Another possible improvement for the DG model is adding more customers as input data. As seen in Section 7.2.2, adding more customers to the data set improves the model performance since more data is available to learn from and to find patterns in. The next step would be to apply the DG model on real bank balance data to see how this model reacts to the correlations and patterns in this, with probably high randomness, data set.

When the development of the synthetic data set is improved, the differences in clusters between the original and the synthetic data sets are also dissolved. To cluster a time series data set in groups, more complicated techniques could be tried to improve performance. Because of the lack of computing time, only K-Means Time Series clustering is applied to the dynamic data set. Research into machine learning techniques such as Reservoir Computing [4], Growing Neural Gas, and Spectral Clustering [16] can be done to find out whether these techniques can improve the performance of clustering these large multivariate time series data sets.

Bibliography

[1] A. Alqahtani, M. Ali, X. Xie, and M. Jones. Deep Time-Series Clustering: A Review. *Electronics*, 10:3001, 12 2021.

[2] Altexsoft. Synthetic Data for Machine Learning: Its Nature, Types, and Means of Generation. https://www.altexsoft.com/blog/synthetic-data-generation/.

[3] A. Amidon. How to Apply K-means Clustering to Time Series Data. https://towardsdatascience.com/how-to-apply-k-means-clustering-to-time-series-data-28d04a8f7da3, 2020.

[4] F. M. Bianchi, S. Scardapane, S. Løkse, and R. Jenssen. Reservoir Computing Approaches for Representation and Classification of Multivariate Time Series. *IEEE Transactions on Neural Networks and Learning Systems*, PP, 03 2018.

[5] C. Bishop. chapter 5. 10 2007.

[6] C. Bishop. chapter 9.1. 10 2007.

[7] N. Braun. Time Series Feature Extraction on (Really) Large Data Samples. https://towardsdatascience.com/time-series-feature-extraction-on-really-large-data-samples-b732f805ba0e.

[8] F. Castillo. SNE vs. t-SNE vs. UMAP: An Evolutionary Guide. https://arize.com/blog/t-sne-vs-umap/, 2022.

[9] Centraal Bureau voor de Statistiek (CBS). De Nederlandse economie 2008.

[10] K. Chilamkurthy. Wasserstein Distance, Contraction Mapping, and Modern RL Theory. https://kowshikchilamkurthy.medium.com/wasserstein-distance-contraction-mapping-and-modern-rl-theory-93ef740ae867.

[11] J. Cryer and K.-S. Chan. *Time Series Analysis: With Applications in R*. 01 2008.

[12] Datagen. Synthetic Data: The Complete Guide. http://datagen.tech/guides/synthetic-data/synthetic-data/.

[13] Denyse. Time Series Clustering - Deriving Trends and Archetypes from Sequential D]ata, howpublished="https://towardsdatascience.com/time-series-clustering-deriving-trends-and-archetypes-from-sequential-data-bb877 year = 2021.

[14] E. Devaux. Types of Synthetic Data and 4 examples of Real-Life Applications. https://www.statice.ai/post/types-synthetic-data-examples-real-life-examples, 2022.

[15] A. Efstratiadis, Y. Dialynas, S. Kozanis, and D. Koutsoyiannis. A multivariate stochastic model for the generation of synthetic time series at multiple time scales reproducing long-term persistence. *Environmental Modelling Software*, 62:139–152, 12 2014.

[16] H. Ertan. Multivariate Time Series Clustering Using Growing Neural Gas and Spectral Clustering. https://towardsdatascience.com/multivariate-time-series-clustering-using-growing-neural-gas-and-spectral-cluste 2022.

[17] European Banking Authority. EBA Discussion Paper on Machine Learning for IRB Models, 2021.

[18] European Data Protection Supervisor. Synthetic data. https://edps.europa.eu/press-publications/publications/techsonar/synthetic-data_en.

[19] Finansjaal. Gemiddeld salaris per leeftijd in Nederland. https://finansjaal.nl/gemiddeld-salaris-per-leeftijd/.

[20] Finansjaal. Gemiddeld vermogen per leeftijd in Nederland. https://finansjaal.nl/gemiddeld-vermogen-per-leeftijd//.

[21] Finansjaal. Vaste lasten maken nu meer dan helft inkomen Nederlanders uit. http://www.huizenmarkt-zeepbel.nl/22-03-2019/vaste-lasten-maken-nu-meer-dan-helft-inkomen-nederlanders-uit/.

[22] A. Géron. chapter 4. 2018.

[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *Advances in Neural Information Processing Systems*, 3, 06 2014.

[24] D. Guitierrez. The Expanding Importance of Synthetic Data, 2022.

[25] J. Hui. Machine Learning — Singular Value Decomposition (SVD) Principal Component Analysis (PCA). https://jonathan-hui.medium.com/machine-learning-singular-value-decomposition-svd-principal-component-analysis-p

[26] Y. Joshi. Applications of Principal Component Analysis (PCA). https://iq.opengenus.org/applications-of-pca/.

[27] Z. Lin, A. Jain, C. Wang, G. Fanti, and V. Sekar. Using GANs for Sharing Networked Time Series Data: Challenges, Initial Promise, and Open Questions. pages 464–483, 10 2020.

[28] J. Moody. What does RMSE really mean? https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e.

[29] M. Naveed, U. Hashmi, N. Tajved, N. Sultan, and A. Imran. Assessing Deep Generative Models on Time-Series Network Data, 04 2022.

[30] R. Nelsen. Properties and applications of copulas: A brief survey. *Proc. 1st Braz. Conf. on Stat. Modeling in Insurance and Finance*, 01 2003.

[31] A. Patil. Beginner's Guide to Pearson's Correlation Coefficient. https://www.analyticsvidhya.com/blog/2021/01/beginners-guide-to-pearsons-correlation-coefficient/.

[32] N. Patki, R. Wedge, and K. Veeramachaneni. The Synthetic Data Vault. pages 399–410, 10 2016.

[33] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for Activation Functions. *CoRR*, abs/1710.05941, 2017.

[34] M. Sit, B. Demiray, Z. Xiang, G. Ewing, Y. Sermet, and I. Demir. A comprehensive review of deep learning applications in hydrology and water resources. *Water Science and Technology*, 82, 08 2020.

[35] StackOverflow. https://stackoverflow.com/questions/52870252/decompose-a-time-series-only-in-trend-and-residual-with-python.

[36] SURFsara. Lisa Compute Cluster: extra processing power for research. https://www.surf.nl/en/lisa-compute-cluster-extra-processing-power-for-research.

[37] Synthetic Data Vault. https://sdv.dev/.

[38] Synthetic Data Vault. Gaussian Copula. https://sdv.dev/SDV/user_guides/single_table/gaussian_copula.html.

[39] J. Tae. The Math Behind GANs. https://jaketae.github.io/study/gan-math/.

[40] S. Takahashi, Y. Chen, and K. Tanaka-Ishii. Modeling financial time-series with generative adversarial networks. *Physica A: Statistical Mechanics and its Applications*, 527:121261, 04 2019.

[41] tsfresh. Data Formats. https://tslearn.readthedocs.io/en/stable/auto_examples/clustering/plot_kmeans.html.

[42] tslearn. k-means. https://tslearn.readthedocs.io/en/stable/auto_examples/clustering/plot_kmeans.html.

[43] T. Ullrich. On the Autoregressive Time Series Model Using Real and Complex Analysis. *Forecasting*, 3:716–728, 10 2021.

[44] L. van der Maaten and G. Hinton. Viualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 11 2008.

[45] J. van der Plas. In Depth: k-Means Clustering. https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html.

[46] A. Verma. Do You Know What is Shannon's Entropy? https://towardsdatascience.com/what-is-shannons-entropy-5ad1b5a83ce1.

[47] A. Vieira. Generating Synthetic Sequential Data using GANs. https://pub.towardsai.net/generating-synthetic-sequential-data-using-gans-a1d67a7752ac.

[48] C. Wang. Generating High-fidelity, Synthetic Time Series Datasets with DoppelGANger. 09 2019.

[49] J. Yoon, D. Jarrett, and M. Schaar. Time-series Generative Adversarial Networks. 12 2019.

[50] K. Zhang, N. Patki, and K. Veeramachaneni. Sequential Models in the Synthetic Data Vault, 07 2022.

A. Preliminaries

In this chapter, the preliminary methodology of three topics is explained. First, a description of time series analysis is given, then the theory of a Gaussian Copula is explained, and last two clustering techniques are described. This chapter can be skipped if there is already knowledge of these subjects.

A.1. Time Series Analysis

In this research, time series are essential since the data consists of simulated balance and transaction count time series. Terms like trend and seasonality are much used, so in this section the basic terms of time series are explained and one kind of time series used in one of the data generating models is illustrated.

A time series is a discrete series of observations collected sequentially over time where time can be in any metric from seconds to days to years. Time series analysis is done to understand the stochastic mechanism of a series or to forecast the future or the course of a time series. The model for a time series is a stochastic process $\{Y_t : t = 0, \pm 1, \pm 2, \dots\}$, which is a sequence of random variables. The set of distributions of the Y's is determined by the probabilistic structure of the stochastic process.

Patterns can be found in time series, for example a trend in the values or seasonalities after a returning period. This is why a time series can be split up in trend, seasonality, and residual as $Y(t) = T(t) + S(t) + R(t)$ where Y is the observed time series, T the trend, S the seasonal values and R the residual values [11]. The residual values represent the time series without trend and seasonality and is called the stationary time series. An example of this time series decomposition is visualized in Figure A.1. In this example, the observed time series initially does not seem to have any seasonality, but after removing the trend and the residuals, a clear seasonality is found. Note that this will be a flat horizontal line when no seasonality is observed.

In time series analysis, an autoregressive (AR) model is used to describe the time series data and to use it for modeling time series. The AR(p) model is a model which describes the value of a process at time t based on the last p time steps $(t-1, \dots, t-p)$. Call the balance amount at time t x_t, and the previous balance amounts $x_{t-1}, \dots, x_{t-p}$, then equation A.1 describes the AR structure.

$$x_t = c + \theta_1 x_{t-1} + \theta_2 x_{t-2} + \cdots + \theta_p x_{t-p} + \epsilon_t. \tag{A.1}$$

Here, p is the order of the AR model, and θ_i, $i = t-1, \dots, t-p$ are the model parameters (the weights of the linear combination) and are constant. The error term ϵ_t is assumed to be white noise, which means that the ϵ_t are uncorrelated with each other in time and

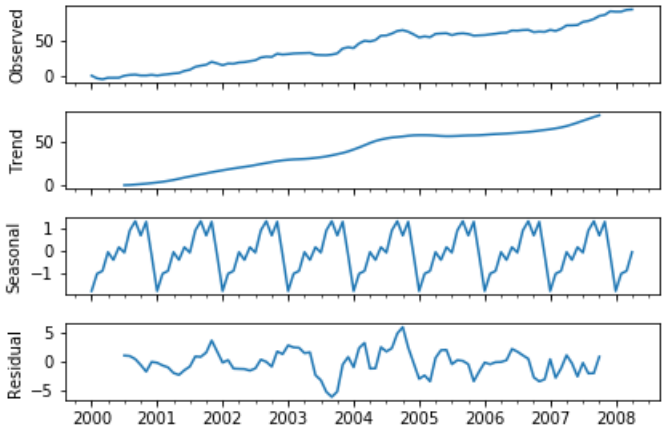


Figure A.1.: Example of time series decomposition [35]

identically distributed with an expected value of zero and finite variance [43]. The AR model describes a time series without trends and seasonalities, these have to be added to the AR model when necessary.

A.2. Gaussian Copula Model

The Gaussian Copula Model is a model of the Synthetic Data Vault (SDV) ecosystem of libraries for generating synthetic single table data [38] and is based on copula functions. This model is used in one of the machine learning models for generating synthetic static features by using a joint distribution.

A copula is a function that describes a multivariate distribution function as a one-dimensional marginal distribution function by describing the joint distribution. Formally, a (two-dimensional) copula in domain $\mathbf{I}$ is a function $C : \mathbf{I}^2 \to \mathbf{I}$ such that

- $C(0, x) = C(x, 0) = 0$ and $C(1, x) = C(x, 1) = x$ for all $x \in \mathbf{I}$;
- C is 2-increasing: for $a, b, c, d \in \mathbf{I}$ with $a \leq b$ and $c \leq d$, $V_C([a, b] \times [c, d]) = C(b, d) - C(a, d) - C(b, c) + C(a, c) \geq 0$.

Where the function V_C is called the C-volume of the rectangle $[a, b] \times [c, d]$ [30].

Equation A.2 describes the copula of the multivariate normal distribution and is called the Gaussian Copula. In this formula, Φ_ρ is the multi-dimensional standard normal distribution of the joint distribution with ρ the linear correlation coefficient.

$$C_{\text{Gaussian}}(u_1, u_2; \rho) = \Phi_\rho(\Phi^{-1}(u_1), \Phi^{-1}(u_2)). \tag{A.2}$$

A.3. Clustering

To answer the second research question, the data has to be grouped into clusters. Different techniques can be used and are described in Section 2.3. Since the computing

time of most neural network-based clustering techniques is too high in this scope, only K-Means clustering and K-Means time series clustering are described.

K-Means Clustering K-Means Clustering is the technique of clustering data in K non-overlapping clusters based on cluster centroids [6]. This method is used to cluster tabular data (non-time series data). Suppose the data set has N data points x_n that are $d-$dimensional. In the scope of this thesis, these data points represent N customers with d features based on aggregating the time series data, such as average transaction count and volatility. The first step to partition the data into K clusters (where K is an input of the user) is to assign K random data points as cluster centroids where c_k is the centroid of cluster k. The goal is to find clusters c_k, $k = 1, \ldots, K$ such that a loss function is minimized. To define the loss function, first define $r_{nk} \in \{0, 1\}$, $k = 1, \ldots, K$ for each data point x_n describing to which cluster x_n is assigned. When data point x_n is assigned to cluster k, $r_{nk} = 1$ and $r_{nj} = 0$ for $j \neq k$. The loss function is then defined as

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - c_k\|^2.$$

The loss function is the sum of squares of the distances of each data point to the corresponding cluster centroid, which has to be minimized for optimal clusters. The minimization is done iteratively, where each iteration consist of two optimization steps. First, minimize J with respect to r_{nk} and keep c_k fixed, this is called the E-step (expectation step). Second, minimize J with respect to c_k and keep r_{nk} fixed, this is called the M-step (maximization step).

The stopping criteria of this iterative process can be chosen as a certain number of iterations or convergence of the loss function. In Figure A.2, an example of K-Means Clustering with three iterations is visualized.

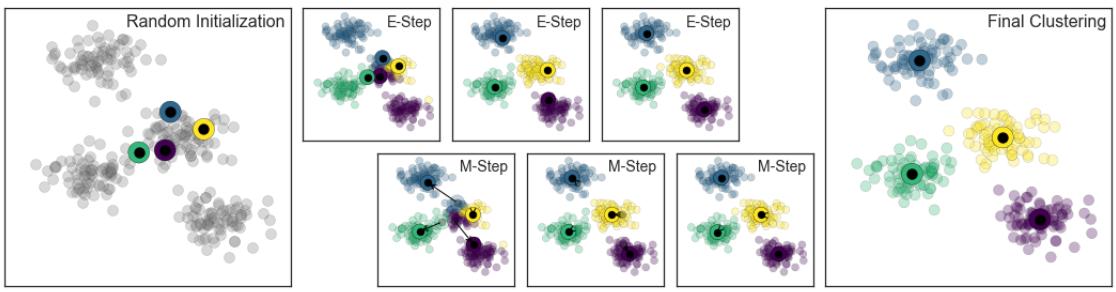


Figure A.2.: Example of K-Means Clustering with three iterations [45]

K-Means Time Series Clustering Time series clustering has as extra difficulty the dependencies over time which can be shifted in time for different series. Two time series with minimum Euclidean distance are not necessarily two time series that are alike. An

example is two time series which are exactly the same but shifted by one or two time steps, can have a large Euclidean distance.

A solution for this is to use K-Means time series clustering with Dynamic Time Warping (DTW). The DTW technique measures similarity between two time series that are not exactly the same in terms of time, speed, or length. To visualize the difference between Euclidean distance and DTW, the matching of two series which are alike is shown in Figure A.3.

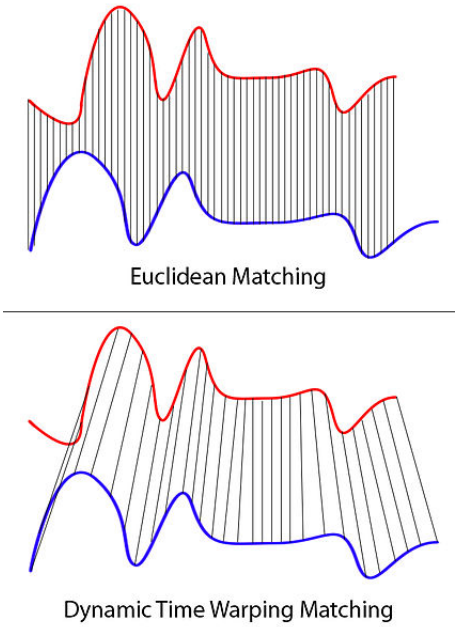


Figure A.3.: Illustration of the differences between Euclidean matching and DTW matching [3]

The DTW distance from a series $X = (x_0, \ldots, x_n)$ to a series $Y = (y_0, \ldots, y_m)$ if defined as

$$DTW(x, y) = \min_{\pi} \sqrt{\sum_{(i,j)\in\pi} (x_i - y_j)^2},$$

where π is a list of possible index pairs. This creates a path that represents for each point in X the nearest point in Y.

B. Initial model findings

In this chapter, an explanation of the two model findings described in Section 7.1 is given with visualizations. The first finding is on the TimeGAN model, where the finding is that this model can not capture the discrete valued time series. The second finding is on the PAR model, where the finding is that more than 128 training steps are needed for good performance. To illustrate both these findings, a data set with sine wave time series is used. The sine waves are visualized in Figure B.1.

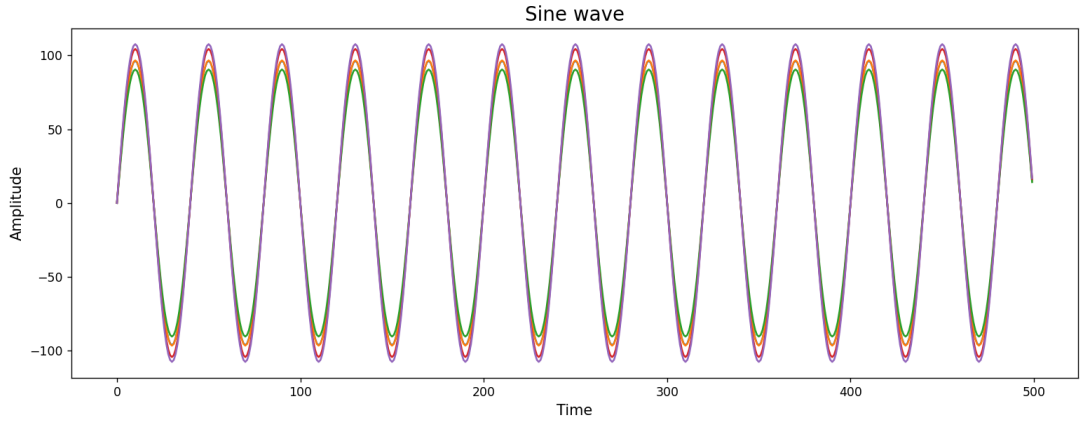


Figure B.1.: Sine wave

B.1. TimeGAN discrete values

The TimeGAN initial finding regards the discrete values, which are the transaction count values in this thesis. To give a visualization of how the model generates a synthetic time series with discrete values, a second time series is added to each of the sine waves with discrete values between 1 and 15, not depending on the values of the sine wave. The synthetic sine waves resulting from the TimeGAN model are visualized in Figure B.2, which are similar to the input data. The synthetic discrete 'transaction count' values resulting from the TimeGAN model are visualized in Figure B.3. As seen in this plot, these time series are more volatile and in the right range of values, but the course is the same as the course of the sine wave. This result of the discrete values being dependent of the continuous values appeared frequently in the results of the TimeGAN model, which led to the decision of not implying the transaction count time series as input for the TimeGAN model.

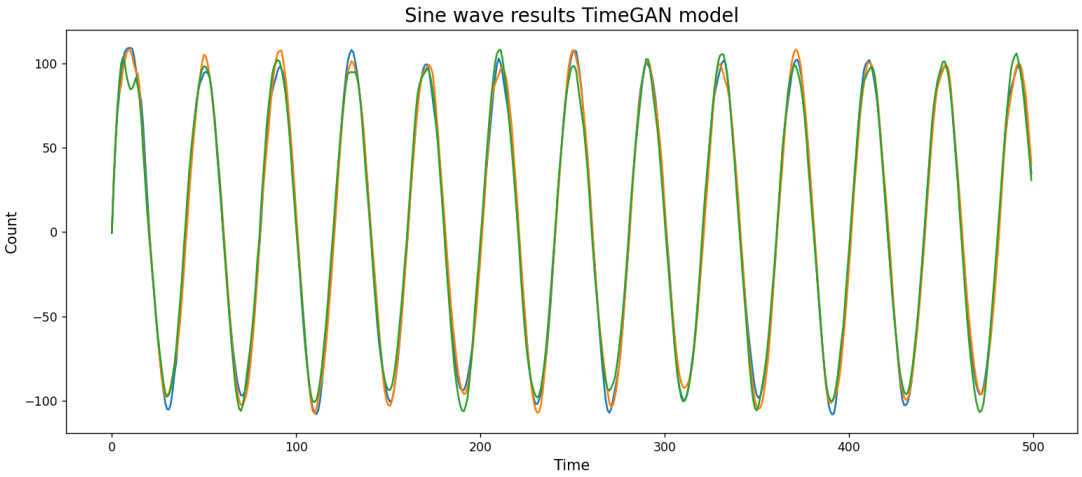


Figure B.2.: Results of the sine wave from the TimeGAN model

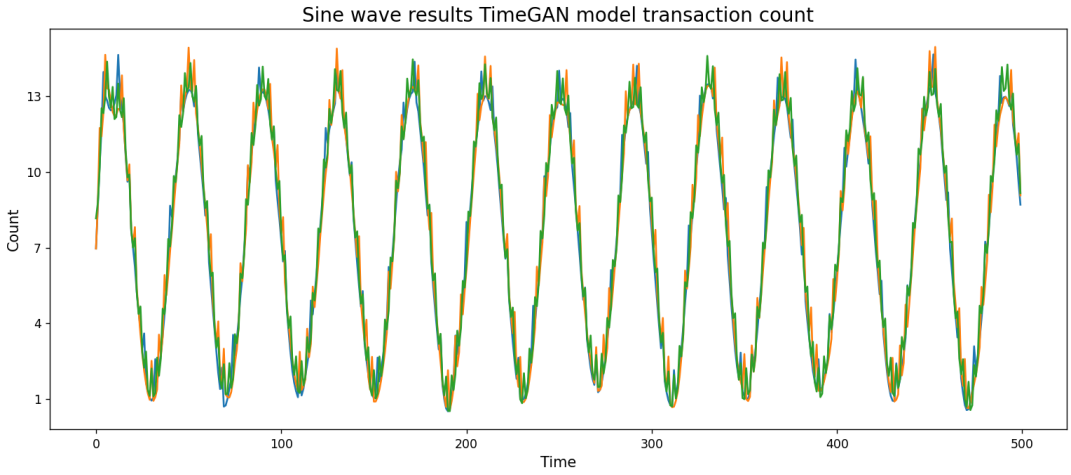


Figure B.3.: Results of the sine wave transaction count from the TimeGAN model

B.2. PAR training steps

The initial finding of the PAR model regards the number of training steps. The model methodology claims that 128 training steps are sufficient for training the PAR model. When training the PAR model with 128 training steps on the simple sine wave data set, the results are as in Figure B.4. These synthetic sine waves do not look like the input sine waves. When increasing the number of training steps to 1.000, the performance also increases, as seen in Figure B.5. Here, the PAR model gives results where the seasonalities and the structure of the sine waves are recognizable. For the simulated balance data sets, the number of training steps is increased and chosen to be 2.000 while keeping the maximum run time in mind.

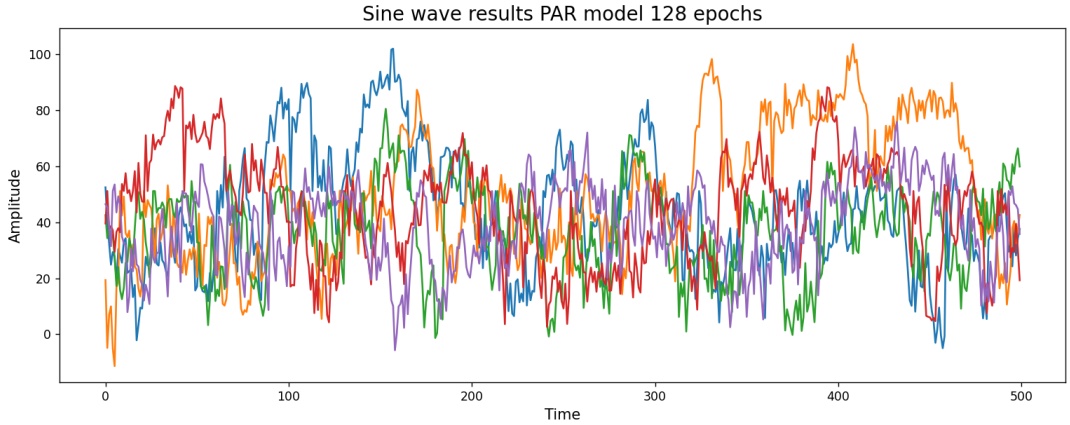


Figure B.4.: Results of the sine wave from the PAR model with 128 training steps

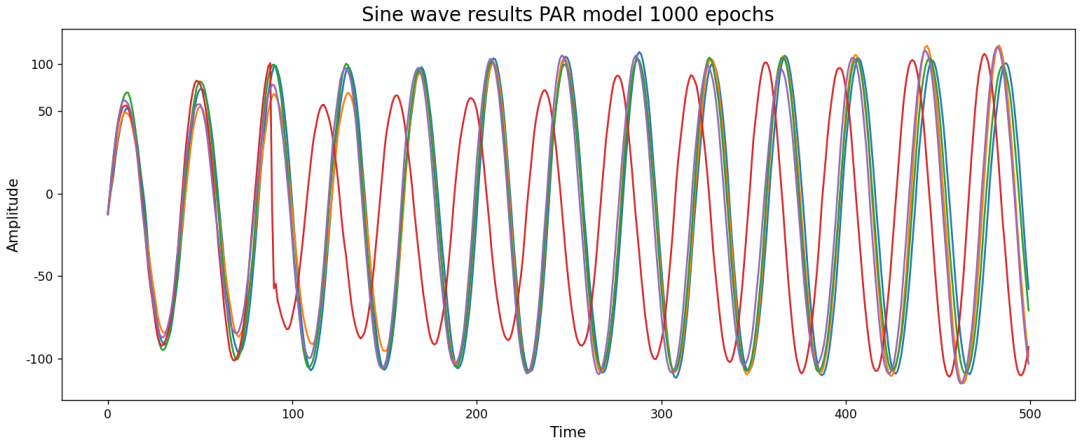


Figure B.5.: Results of the sine wave from the PAR model with 1000 training steps

C. Hyperparameter settings

Hyper parameter	Value
Training steps	2.000

(a) Hyper parameters PAR model

Hyper parameter	Value
Hidden dimensions	28
Number of layers	3
Iterations	24.000
Batch size	128
Module	GRU

(b) Hyper parameters TimeGAN model

Hyper parameter	Value
Max sequence length	1.827
Sample length	1.827
Batch size	1.000
Apply feature scaling	*True*
Apply example scaling	*False*
Use attribute discriminator	*True*
Attribute loss coefficient	1.0
Generator learning rate	$3e^{-5}$
Discriminator learning rate	$3e^{-5}$
Attribute discriminator learning rate	$3e^{-5}$
Training steps	13.000

(c) Hyper parameters DG model

Table C.1.: Hyper parameters used for training the three models

D. Results intermediate data set

D.1. Results intermediate data set PAR, TimeGAN, and DG

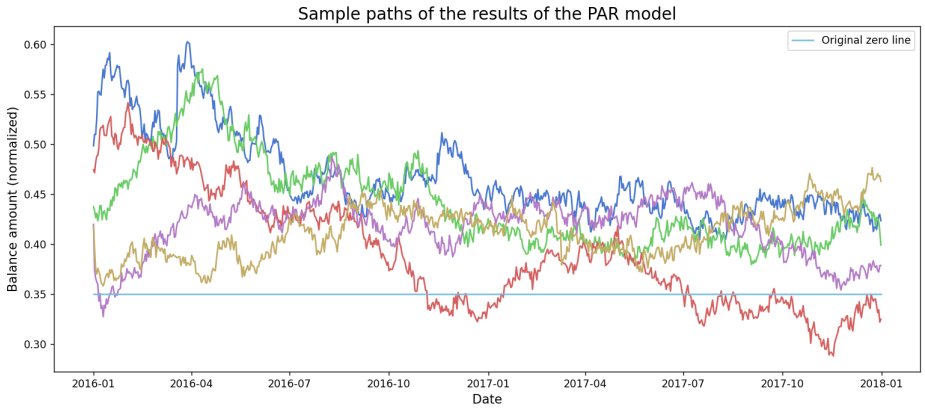


Figure D.1.: Sample paths of five customers of the synthetic data set after applying the PAR model to the intermediate data set

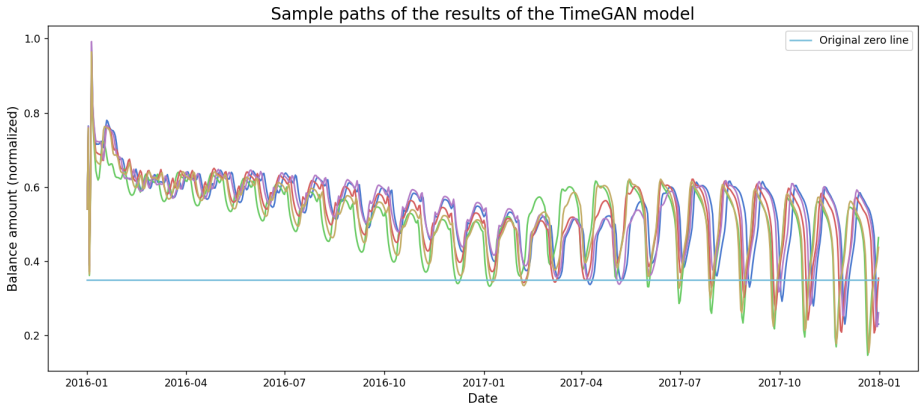


Figure D.2.: Sample paths of five customers of the synthetic data set after applying the TimeGAN model to the intermediate data set

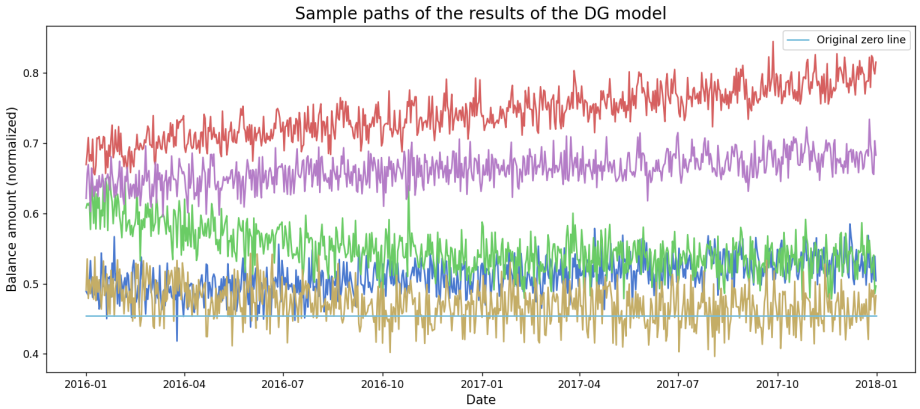


Figure D.3.: Sample paths of five customers of the synthetic data set after applying the DG model to the intermediate data set

Model	**Mean**	**Variance**	**Inter correlation**
Original	0.4877	0.0091	0.1904
PAR	0.5306	0.0046	0.0606
TimeGAN	0.4195	0.0196	-
DG	0.5048	0.0241	0.1977

Table D.1.: Statistical metrics of the intermediate data set and the results of the synthetic data sets per model

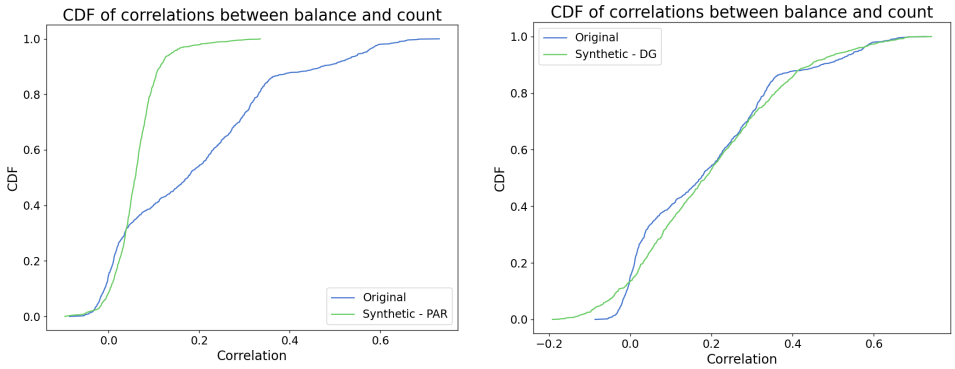
Wasserstein distance per age group			
Model	**age** < 25	$25 \leq$ **age** ≤ 75	**age** > 75
PAR	0.0275	0.0566	0.0708
DG	0.0883	0.0579	0.0805

Table D.2.: Wasserstein distance between original and synthetic data per age group

D.2. Results intermediate data set DG adjustments

Model	**Mean**	**Variance**	**Inter correlation**
Original	0.4877	0.0091	0.1904
DG - 1.000	0.5048	0.0241	0.1977
DG - 10.000	0.4708	0.0128	0.1531
DG - extra layers	0.4950	0.0535	0.1460
DG - extra steps	0.4898	0.0163	0.1840

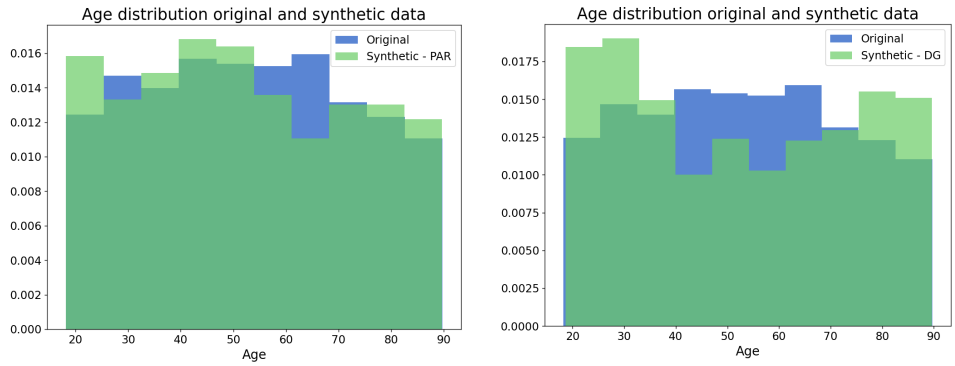
Table D.3.: Statistical metrics of the intermediate data set and the results of the synthetic data sets per different DG model



(a) PAR model

(b) DG model

Figure D.4.: The CDF of the correlations between balance amount and transaction count for the original data set and the synthetic data sets of the PAR and DG models



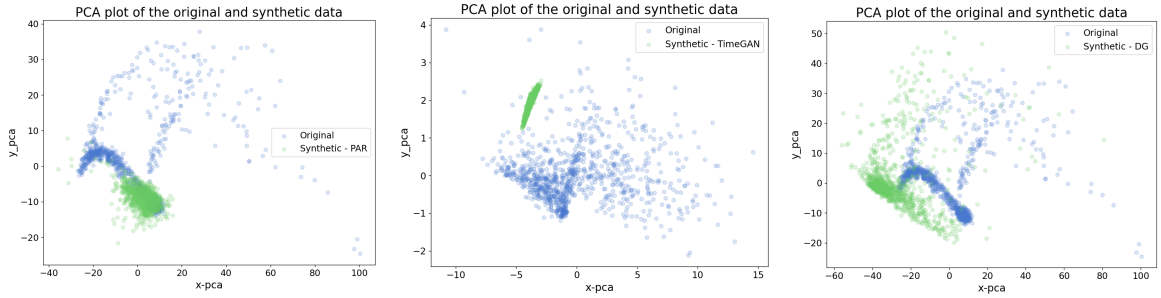
(a) PAR model

(b) DG model

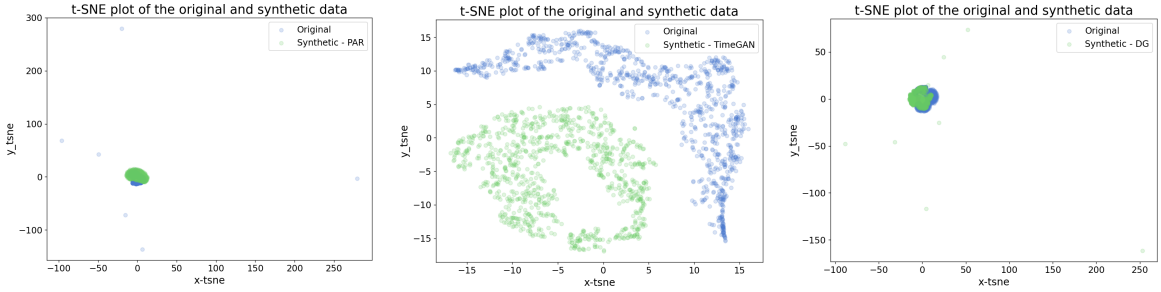
Figure D.5.: Normalized age distribution of the original data set and the synthetic data sets of the PAR and DG models

Wasserstein distance per age group			
Model	**age** < 25	$25 \leq$ **age** ≤ 75	**age** > 75
DG - 1.000	0.0883	0.0579	0.0805
DG - 10.000	0.0538	0.0672	0.0490
DG - extra layers	0.1627	0.1317	0.1467
DG - extra steps	0.0938	0.0576	0.0801

Table D.4.: Wasserstein distance between original and synthetic data per age group

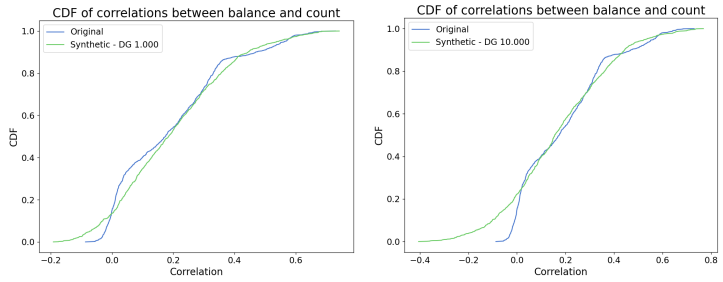


(a) PCA PAR model (b) PCA TimeGAN model (c) PCA DG model

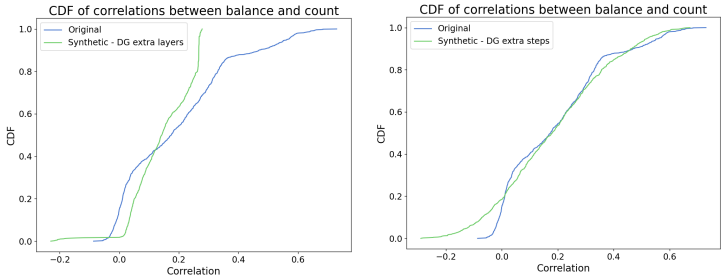


(d) t-SNE PAR model (e) t-SNE TimeGAN model (f) t-SNE DG model

Figure D.6.: Visual metrics applied on the synthetic data sets of all three models against the original data set

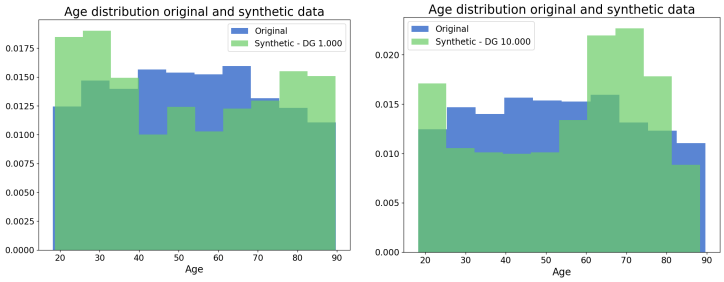


(a) DG - 1.000 (b) DG - 10.000

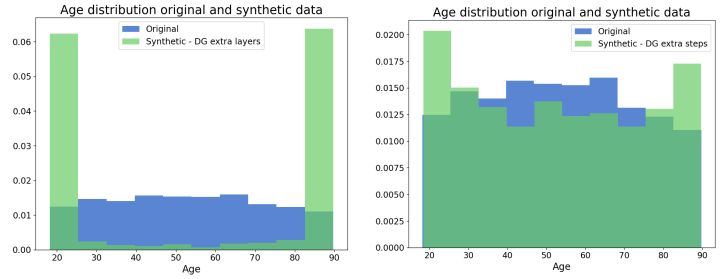


(c) DG - extra layers (d) DG - extra steps

Figure D.7.: The CDF of the correlations between balance amount and transaction count for the original data set and the synthetic data sets of the adjusted DG models

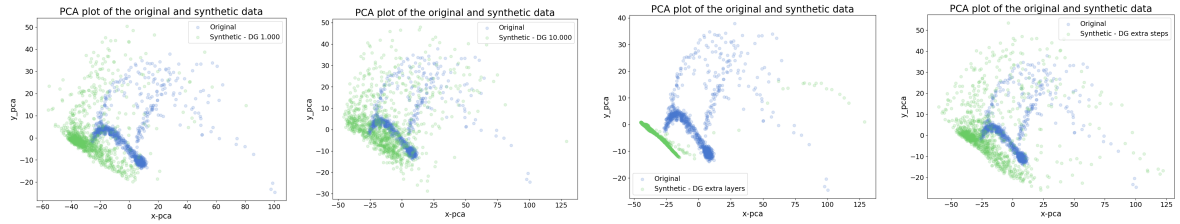


(a) DG - 1.000 (b) DG - 10.000

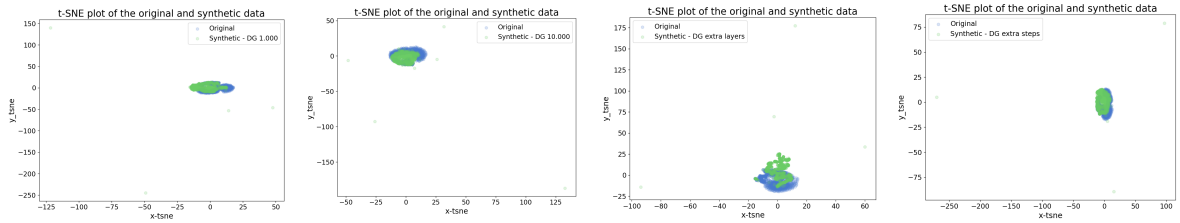


(c) DG - extra layers (d) DG - extra steps

Figure D.8.: Normalized age distribution of the original data set and the synthetic data sets of the adjusted DG models



(a) PCA DG - 1.000 (b) PCA DG - 10.000 (c) PCA DG - extra layers (d) PCA DG - extra steps



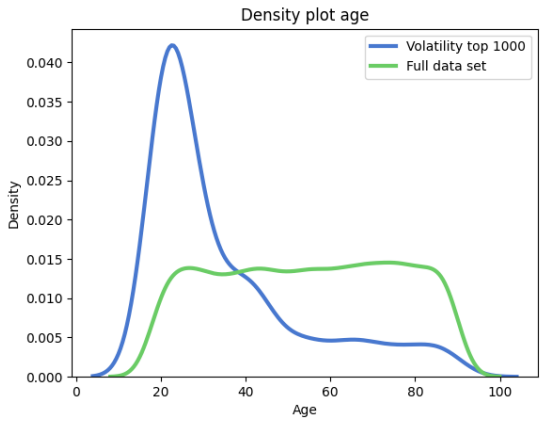
(e) t-SNE DG - 1.000 (f) t-SNE DG - 10.000 (g) t-SNE DG - extra layers (h) PCA DG - extra steps

Figure D.9.: Visual metrics applied on the synthetic data sets of the adjusted DG models against the original data set

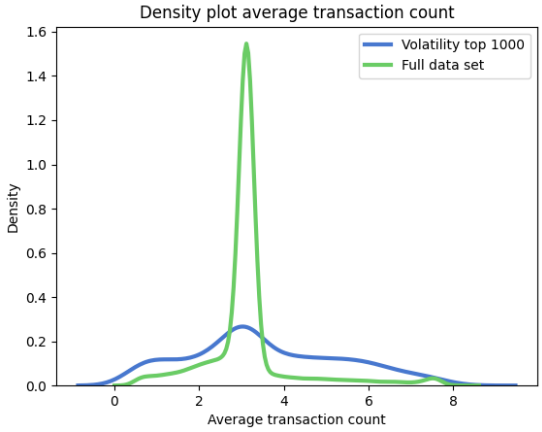
E. Data analysis clusters

Data set	Avg age	Std age	Avg count	Std count
Top 1.000 volatile customers	34.9	18.7	3.6	1.8
Full data set	54.5	20.9	3.1	1.1

Table E.1.: Spread of the statistics for the top 1.000 volatile customers and the full data set



(a) Density age

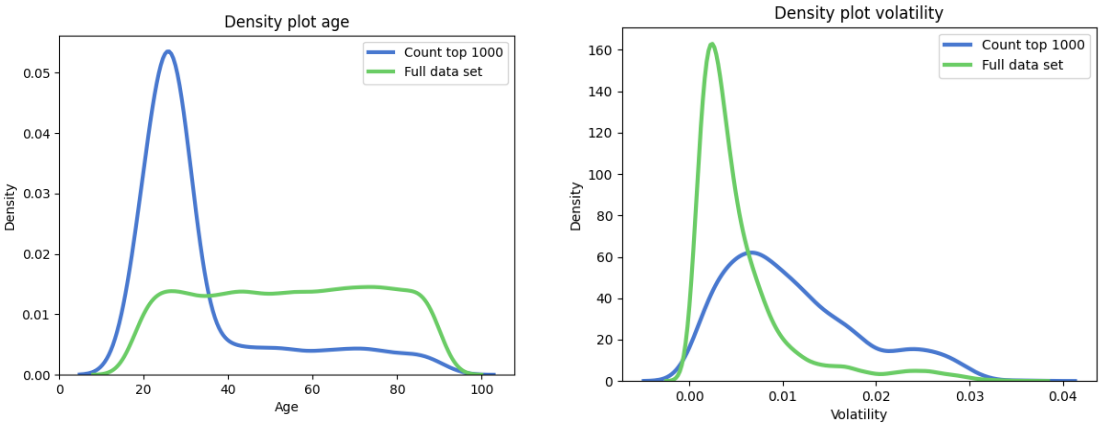


(b) Density average count

Figure E.1.: Density plots of the age and the average transaction count for the 1.000 customers with the highest volatility (blue), and for the full data set (green)

Data set	Avg age	Std age	Avg volatility	Std volatility
Top 1.000 average count	33.7	17.4	0.0097	0.0073
Full data set	54.5	20.9	0.0051	0.0055

Table E.2.: Spread of the statistics for the top 1.000 average count and the full data set



(a) Density age

(b) Density volatility

Figure E.2.: Density plots of the age and the volatility for the 1.000 customers with the highest average transaction count (blue), and for the full data set (green)