Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Master's Thesis

Diffusion based Tabular Data Synthesis

MIN-Fakultät
Fachbereich Informatik

Sven Groen
Sven.Groen@studium.uni-hamburg.de
Studiengang IT Management und Consulting
Matr.-Nr. 7388105

Erstgutachter: Dr. Fabian Panse
Zweitgutachter: Professor Dr. Wolfram Wingerath

Abstract

The growing demand for data in machine learning and specifically deep learning applications, combined with the difficulties in acquiring and gathering real-world data, has fueled the development of synthetic data generation techniques. Synthetic data, which is artificially generated but modeled on real data, can address privacy constraints and provide a cost-effective alternative for various use cases. Through the utilization of diffusion models, a novel data generation methodology, the quality of synthetic image production has advanced, surpassing the image quality of previously established Generative Adversarial Network (GAN)-based methodologies. Recently, TabDDPM, a generative diffusion model, outperformed GANs on tabular data synthesis as well. Synthesizing tabular data presents unique challenges due to the complexity of the underlying joint distributions between variables and the need to capture intricate relationships among features. While GAN based solutions have been heavily explored in the literature, diffusion-based solutions are relatively new and unexplored.

This thesis investigates whether adapting different tabular processing mechanisms from the literature positively affects the diffusion model's generative capability by encoding the tabular data into a different data format that specifically addresses known challenges of tabular data. By extending the existing tabular data generation pipeline of TabDDPM, various tabular encoding and decoding strategies are implemented. It focuses on two tabular processing mechanisms that encode the data before training and revert it back after sampling synthetic data: a static embedding technique named Feature Tokenization (FT) and a mode-specific-normalization encoding technique that makes use of a Bayesian Gaussian Mixture Model (BGM). In addition to that, this study touches upon the effects of normalization and hyperparameter optimization on diffusion models for tabular data synthesis. The pipeline was evaluated through a CatBoost machine learning efficacy test and a TabSynDex similarity metric, alongside a comparative analysis of synthetic and real data features visualized. The visualizations incorporated correlation difference plots, principal components, column distributions, and cumulative density functions.

The results demonstrate the superiority of diffusion models over other generative techniques, such as GANs and Variational Autoencoders (VAE), in generating synthetic tabular data. The addition of a BGM-based processing mechanism improves the TabDDPM model's performance in machine learning scenarios, while the FT approach fails to produce meaningful data. The importance of hyperparameter tuning and data normalization strategies is highlighted, as well as the need for a comparative visual evaluation of dataset characteristics to accurately assess synthetic data quality. Tuning the hyperparameters to optimize the TabSynDex similarity metric significantly affects diffusion models more than non-diffusion models. This indicates the diffusion models' flexibility to generate synthetic data to cater to the specific requirements of the intended use case. Finally, diffusion models produced synthetic data more indistinguishable from real data than non-diffusion models, as suggested by a non-zero propensity Mean Squared Error (pMSE) score.

Table of Contents

Glossary

GDPR The General Data Protection Regulation (GDPR) [Eur16] is a data privacy and security law passed by the European Union, which imposes standards and potential penalties on organizations worldwide that handle data related to people within the European Union.
Pages: 1, 11

Latent variable model Latent variable models are a type of probabilistic model that use latent or hidden variables to explain the observed data. They model the relationship between the observed data and the latent variables through a set of conditional probability distributions. The goal is often to use the joint distribution of the observed data and latent variables to make predictions or perform inferences about the data, which typically involves marginalizing over the latent variables. This means integrating over all possible values of the latent variables to obtain the probability distribution of the observed data [SR07; Bis98].
Pages: 26

Markov chain A Markov chain, also known as a Markov process, represents a probabilistic model where the likelihood of an event occurring is determined exclusively by the outcome of the preceding event in a series of potential occurrences [Gag17].
Pages: 26, 27, 30

Model A model in machine learning or deep learning is a mathematical representation that captures the relationship between inputs and outputs in data. It is used to make predictions about new, unseen data by applying mathematical operations to the inputs. Deep learning models are a subset of parametric models in which the parameters are represented as weights in a neural network and are changed during training to reduce the discrepancy between expected and actual outcomes. Machine learning models include but are not limited to decision trees, logistic regression, random forests, and linear regression. Convolutional neural networks (CNNs), recurrent neural networks (RNNs), and autoencoders are a few examples of deep learning models [Par21].
Pages: 1

One-Hot A One-Hot vector is a binary vector used to represent categorical data in machine learning and deep learning algorithms. It is characterized by having exactly one

element set to 1 (the "hot" element) and all other elements set to 0. Suppose a dataset contains a "color" feature with three distinct categories: red, green, and blue. Using one-hot encoding, a unique one-hot vector for each color would look like the following:
Red: [1, 0, 0]
Green: [0, 1, 0]
Blue: [0, 0, 1].
Pages: 8, 10, 31, 42, 47, 48, 66, 92

Pandas Pandas [Wes10] is an open-source Python library that provides high-performance, easy-to-use data structures, and data analysis tools. It is designed to facilitate data manipulation, cleaning, analysis, and visualization in Python.
Pages: 69

TOML TOML, which stands for Tom's Obvious, Minimal Language [Pre21], is an open-source configuration file format that aims to be easy to read and write. TOML documents contain sections, indicated with square brackets which in turn contain key-value pairs, specifying the overall configuration.
Pages: 57

List of Abbreviations

BGM Bayesian Gaussian Mixture Model.
Pages: 0, 3, 65, 66, 68, 77, 78, 84, 85, 88, 89, 91–94, 97

CDF Cumulative Density Function.
Pages: 87, 88, 91–93

CNN Convolutional Neural Network.
Pages: 9, 14–16, 41

CPU Central Processing Unit.
Pages: 72

DCR Distance to Closest Record.
Pages: 49

DDPM Denoising Diffusion Probabilistic Model.
Pages: 26, 30, 31, 44

DP Differential Privacy.
Pages: 35

FT Feature Tokenization.
Pages: 0, 3, 65, 68, 78, 84, 87, 89, 92, 93, 97

GAN Generative Adversarial Network.
Pages: 0, 1, 3, 18, 22, 41–43, 45, 47, 66, 77, 91, 94, 98, 99

GDPR General Data Protection Regulation.
Pages: 1, 11, *Glossary:* GDPR

GMM Gaussian Mixture Model.
Pages: 42

GPU Graphics Processing Unit.
Pages: 45, 72

GRU Gated Recurrent Unit.
Pages: 16

KL Kullback-Leibler.
Pages: 19, 20, 27, 29, 32, 33, 48

LSTM Long Short-Term Memory.
Pages: 16, 35, 42

MLP Multilayer Perceptron.
Pages: 14, 15, 39, 48, 49, 58, 61

MSE Mean-Squared Error.
Pages: 29

NLP Natural Language Processing.
Pages: 9, 14, 18, 23, 43

PCA Principal Component Analysis.
Pages: 19, 36, 37, 82–84, 93

PCD Pairwise Correlation Difference.
Pages: 33

pMSE propensity Mean Squared Error.
Pages: 0, 38, 39, 77–80, 90–94, 97, 98

ReLU Rectified Linear Activation Unit.
Pages: 15, 48

RMSE Root Mean Squared Error.
Pages: 34, 39, 47

RNN Recurrent Neural Network.
Pages: 16–18, 23

RQ Research Question.
Pages: 2, 89, 93, 94

SMOTE Synthetic Minority Oversampling Technique.
Pages: 43, 48, 49, 81, 90, 92

TOML Tom's Obvious, Minimal Language.
Pages: 57, *Glossary:* TOML

VAE Variational Autoencoders.
Pages: 0, 3, 18, 19, 21, 27, 29, 43, 47, 94, 98

VGM Variational Gaussian Mixture model.
Pages: 10, 42, 66

List of Figures

List of Tables

1 Introduction

1.1 Problem Statement and Motivation

Tabular data is essential in our society due to its ability to organize and present data in a structured manner that can be easily analyzed and interpreted by humans and machines alike. This makes it an indispensable tool for decision-making in many fields, including finance, medical research, etc. In the era of deep learning and big data, the need for accurate and reliable data in high quantities is paramount. Training data is essential for machine and deep learning models to perform any desired inference. Gathering and accumulating real-world data is still expensive [Bor+22] and restricted through privacy regulations like GDPR [Eur16]. In recent years, generative modeling approaches have addressed these issues by creating a synthetic copy of the original tabular dataset. This synthetic data version should maintain the important statistical properties and correlations while not disclosing any information from the real original dataset [Goo+20; MLA18]. While generative models in the image domain already show a human-like capability to produce highly realistic images [DN21], tabular data synthesis approaches still offer room for improvements [Chu+22]. One of the reasons for this is the heterogeneous nature of tabular data, which generative models need to reproduce. Tabular data usually consist of a mixture of numerical and categorical distributions with dependencies, which is difficult to capture and reproduce [Bor+22]. So far, GAN-based approaches have shown state-of-the-art capabilities to create synthetic tabular data, which have also been the best model for synthetic image generation. However, diffusion-based models have recently been able to outperform GANs on the task of image synthesis [DN21]. As a result, researchers have tried to use diffusion models to generate synthetic tabular data with promising results [Kot+22; ZC22].

[DN21] argued that GANs have shown superior performance against diffusion because researchers have investigated them more deeply. The authors showed that diffusion models could outperform GANs on image synthesis with a few improvements. A similar observation can be made in the tabular data domain, where the majority of recent publications propose GAN-based solutions. First results [Kot+22] indicate that the superior performance of diffusion models over GANs extends to the tabular data synthesis domain. However, applying diffusion models to tabular data is still a novel approach, and further research is required [Bor+22].

1.2 Goals and Research Questions

This thesis aims to further investigate how diffusion models can be successfully applied to tabular data synthesis. This work will build upon the work of Kotelnikov, Baranchuk, Rubachev, and Babenko [Kot+22] and will explore how the existing approach called TabDDPM can be further improved by adapting already existing tabular processing mechanisms from previous tabular data synthesis solutions. Furthermore, the evaluation and comparison of the diffusion model to various other generative modeling techniques performed by Kotelnikov et al. [Kot+22] will be extended through the usage and analysis of multiple similarity metrics as proposed by Chundawat et al. [Chu+22]. This extended evaluation aims to uncover potential unknown modeling properties of diffusion models that may not have been discovered in previous works.

To summarize, this thesis will investigate the following Research Questions (RQs):

RQ1: What are the effects on the generative capability of the tabular data synthesis model TabDDPM by adding additional tabular data processing mechanisms to the existing generation pipeline?

RQ2: How does TabDDPM (and its variants, produced to answer RQ1) compare to other generative models in an extended similarity evaluation, based on the metrics of Chundawat et al. [Chu+22]?

Answering the above questions contributes to the overall research area by showing possible strengths and weaknesses of diffusion models as a new approach in the tabular data synthesis domain.

1.3 Procedure Overview

In order to answer the above-stated RQs, the following steps have been taken: Prior to any experiments, a literature search has been performed about different possible approaches to processing tabular data, tabular data synthesis, and how to evaluate synthetic data. This research selected a tabular processing mechanism based on predefined criteria (Subsection 4.3.1). Using the existing TabDDPM model [Kot+22], the results of the original authors needed to be replicated and validated in the first step. Afterwards, the software was extended to integrate the selected tabular processing mechanism. The existing evaluation proposed in [Kot+22] was extended by an additional, more elaborate evaluation [Aki23]. Lastly, the different model versions have been trained, and the results have been compared and analyzed.

1.4 Contributions

This thesis provides several contributions to the field of diffusion-based tabular data synthesis. These contributions expand on the understanding of diffusion models' performance and potential improvements in generating synthetic tabular data. The main contributions of this thesis are as follows:

1. **Investigation of tabular processing mechanisms:** This thesis investigates the impact of different tabular processing mechanisms on an existing diffusion model architecture for data synthesis. By comparing a Feature Tokenization (FT) and a Bayesian Gaussian Mixture Model (BGM) processing technique with several baseline models, new insights are gained into how the encoded data format affects the generative capabilities of diffusion models.

2. **Extension of the existing tabular data generation pipeline:** The current TabDDPM [Kot+22] pipeline is extended to allow for easy implementation of different tabular encoding and decoding strategies. This flexibility enables researchers to experiment with various data processing methods and assess their impact on the diffusion model's performance in generating synthetic data.

3. **Comprehensive evaluation:** This thesis introduces a comprehensive evaluation that combines machine learning efficacy, TabSynDex similarity metrics and visualization plots for analysis. This multi-faceted evaluation provides a more holistic view of the generated synthetic tabular data, uncovering strengths and weaknesses from different perspectives.

4. **Identification of diffusion models' superiority:** Through extensive experiments and evaluations, this thesis shows that diffusion models, specifically TabDDPM-BGM, outperform other generative models, such as GAN and VAE in producing synthetic tabular data.

5. **Emphasis on hyperparameter tuning:** This thesis highlights the importance of hyperparameter tuning in diffusion models for tabular data synthesis. It shows that appropriate tuning can significantly enhance the performance of diffusion models across various metrics, making them more versatile and adaptable than other generative models.

1.5 Outline

This thesis starts with essential background information required for a complete understanding of tabular data synthesis in Chapter 2. This is followed by an outline of the most significant related work in Chapter 3, which highlights relevant models from the literature. In Chapter 4, the requirements for the software are listed, and an explanation of the codebase is presented. An expansion of the codebase and a description of the implemented changes follows this. The subsequent chapter, Chapter 5, elucidates how the software was implemented and the architectural changes made. The experiments' results are presented and analyzed in Chapter 6. Afterward, possible limitations of the results are presented and an outline of possible future research has been given. Finally, Chapter 7 summarizes the crucial findings of this thesis.

2 Preliminaries

2.1 Tabular Data

Tabular data is one of the most common forms of structured data [Her+22] used to store, classify and share information [Pil+22]. A table is made up of individual cell entries stored in rows and columns. Rows can be seen as unique data points, and columns as their different features [Bor+22; Yoo+20]. This format is the most common way to maintain massive databases [Esm+22; Yoo+20], and is crucial for applications that store heterogeneous information such as demographics, medical or financial information [Bor+22; Yoo+20]. Tabular data consists of multiple attribute types, such as categorical or continuous data types [Bor+22]. From a statistical perspective, quantitative and qualitative variables are to be differentiated [Lan03]. Quantitative variables are variables that can be measured numerically and express a specific quantity or amount. Qualitative variables cannot be measured numerically but describe qualities or characteristics that do not have any ordering [Lan03]. These variables are divided into categories or groups based on shared attributes. The categories are usually non-numerical and consist of words or labels, such as gender (*male*, *female*) or hair color (*blonde*, *brunette*, *red*, etc.) and may be assigned to numbers.

Quantitative variables can be further subdivided into continuous or discrete variables. Continuous variables can take on an infinite number of values within a specific range. These variables are often measured on a continuous scale, can have decimal or fractional parts, and are stored as a numerical data type [Lan03; LHB22]. Discrete variables are also numbers, but they can only have a specific and limited number of unique values within a certain range. They are usually expressed as whole natural numbers (or integers) and are commonly used for counting or listing items. [Lan03].

Qualitative variables are often called categorical values and can be further classified into binary or nominal types [Lan03]. Binary variables have only two possible outcomes (e.g. booleans with *True* or *False*), while nominal variables consist of at least three distinct values that do not follow a specific order [LHB22; Lan03].

This work adapts the formal definition of a table from [Xu+19]:

> A table T contains N_q qualitative (categorical) columns $\{Q_1, \ldots, Q_{N_q}\}$ and N_p quantitative columns $\{P_1, \ldots, P_{N_p}\}$, where the quantitative columns consist of both continuous and discrete variables. Each column is considered to be a random variable. These random variables follow an unknown joint distribution $\mathbb{P}(Q_{1:N_q}, P_{1:N_p})$. One row $r_j = \{q_{1,j}, \ldots, q_{N_q,j}, p_{1,j}, \ldots, p_{N_p,j}\}$, $j \in \{1, \ldots, n\}$, is one observation from the joint distribution.

The term "table" as described in this definition aligns with the concept of a single "table" or "relation" as defined in the classical relational data model [Cod70] and used in SQL databases [W E09].

For consistency and simplicity, this thesis will further refer to qualitative columns as *categorical* and to quantitative columns as *continuous* or *numerical*. Discrete values are not separately mentioned but should be considered included in *continuous/numerical* columns.

It is also possible that tabular data contains other particular data types like dates or timestamps, which often include information on the specific time a datapoint was recorded [Her+22]. This kind of tabular data can be considered dynamic tabular data, where individual records, i.e. rows, can be dependent on each other, also known as a multivariate time series [Pad+21]. In static tabular data, on the other hand, the individual rows are independent [Pad+21]. Hence, the order of rows and columns does not carry any meaning [Som+21]. However, the individual values in one cell may vary well depending on the values of another cell [LHB22]. An example of such an interdependency could occur in a demographics table, where an individual's legal status may depend on their age, for example, a person under 18 years of age could be considered a minor and has different legal rights and responsibilities than an adult.

The authors of [Bor+22] identify four possible challenges when working with tabular data in a learning context. The first challenge identified is the "low-quality" of the data [Bor+22, p. 4]. Borisov et al. [Bor+22] identified common data quality issues from the literature, which include missing values, data noise, extreme data points, inconsistencies, class imbalances, or high dimensionality of the data after preprocessing.

Secondly the "missing or complex irregular spatial dependencies" of tabular data [Bor+22, p. 4]. Other common data types like images or audio are homogeneous, meaning that they consist of only one feature type [Bor+22]. Since tabular data consists of multiple features made up of a mixture of categorical and numerical values, it is a heterogeneous data format with data points as rows and features as columns [Bor+22]. This format presents particular challenges for neural networks when working with tabular data, as the correlations between features are often weaker due to the absence of spatial or semantic relationships commonly found in image or text data [Bor+22; Yoo+20].

The third challenge is the "dependency on preprocessing" [Bor+22, p. 4]. [Bor+22] highlights the importance of a preprocessing and an explicit feature construction step that

is necessary when working with tabular data in a deep learning context. This preprocessing step is crucial since it not only strongly influences the performance of deep learning models [GRB22], it also introduces new challenges. Depending on the preprocessing strategy (Subsection 2.1.1), it is possible to create a very sparse feature matrix, construct a synthetic ordinal ordering of a nominal variable, or lose some information during the conversion of the data [Bor+22].

The last and fourth challenge concerns the "importance of single features" [Bor+22, p. 4]. In homogeneous data, multiple features need to change in order for the class of the data to be changed. For heterogeneous tabular data, a small change in one feature variable can already alter the class of the row. [Bor+22] illustrates this with the example of an image, where multiple pixels (i.e. features) need to change in a coordinated manner in order to change the content (i.e. class) of the image from a dog to a cat. For tabular data, a single change in a cell can change the prediction of a predictive model [Bor+22]. Consider a model that has to predict whether an individual's income is higher or lower than US$ 50.000 per year [DG17] based on the demographic information of that individual. Switching an individual's "education" value from "Preschool" to "Doctorate" would likely cause the prediction to change from "<=50K" to ">50K".

2.1.1 Tabular Data Preprocessing

Different data types can and should be processed into a meaningful format to be useful for deep learning models in different ways [Fan+20; LHB22]. That is usually the first step before working with the data on any task [Izo+22]. [Gar+16] states, that data preprocessing itself consists of multiple different tasks: data cleaning, data normalization, data transformation, data integration, missing value imputation and noise identification. While each of the preprocessing tasks in itself is important, [FVH12] and [GRB22] showed that a proper transformation of categorical and numerical entries respectively can have a significant influence on a deep learning model's performance. [Xu+19] showed the importance of normalization for synthetic tabular data generation. Since the focus of this work is on tabular data and its synthesis, the following section will highlight the most important tabular data transformation and normalization approaches.

Data Transformation

Borisov et al. [Bor+22, p. 5] introduced a taxonomy for data transformation methods and subdivides the existing approaches into "Single-Dimensional Encodings" and "Multi-Dimensional Encodings". The goal is to transform the different values a column can take and transform them into a different (numeric) representation so that it can be processed by a deep learning model.

Single-Dimensional Encodings: Single-Dimensional encoding techniques encode each cell independently [Bor+22]. The following approaches are common techniques to encode a categorical column entry, usually in a text format, into a numerical format. In ordinal-

(or label-) encoding, a simple mapping from each category to a numeric value occurs. While this introduces a synthetic ordering of potentially unordered categories, One-Hot encoding overcomes this issue by introducing a new vector with the length of all possible values a categorical column can take. All values in this vector are assigned to zero except one entry which represents the category that should be encoded, which is set to one. However, this approach can lead to high-dimensionality feature vectors if the cardinality of the unique categories in categorical columns is large. Binary encoding tries to reduce the dimensionality by setting the vector length to a maximum of $log(c)$ for c unique categorical values in a column [Bor+22]. Each possible value is mapped to a number like in ordinal-encoding, starting at 0, but the number is represented as a binary vector. As an example, the categorical values {*Germany*, *USA*, *France*} would be represented as binary values {(0,1), (1,0), (1,1)}. While representing *France* in a one-hot-encoding creates the vector (0,0,1) of length three, binary encoding encodes the same information into the vector (1,1) of length two, reducing the overall dimensionality of produces encodings. The leave-one-out encoding technique is an approach to encode a categorical column based on the target column in a machine learning scenario [Bor+22]. A categorical entry is replaced by the mean of the target variable of all rows where the same category is present, excluding the target value of the to be encoded value. Lastly, a hash-based approach is worth mentioning, where a deterministic hash function transforms each category into a numerical form [Bor+22].

Numerical data, such as integers or floating point numbers, can often be used directly in deep learning models without undergoing a special encoding process. This is because deep learning algorithms are designed to handle numerical data and can learn patterns and relationships within the data without the need for additional encoding. However, [GRB22] has shown that in some cases, encoding numerical data can improve the performance of deep learning models. Encoding numerical data can be achieved through various methods such as normalization, discretization, or using embeddings.

Discretization techniques transform numerical features to categorical features, hence, quantitative data into qualitative data [Gar+16]. [DKS95] gives an overview of classical discretization techniques, such as equal interval width binning, where the continuous values are divided and assigned to a certain amount of bins. Researchers from NVIDIA [DS22] have recently put forth a new method, introducing a tokenizer designed specifically for handling tabular data with floating-point numbers. This tokenizer converts float numbers into a sequence of token IDs [DS22].

In embedding techniques, values that should be encoded (e.g. words or tabular cell entries) are mapped into a vector representation. This vector of real numbers tries to capture "semantic regularities in vector spaces" [Pil+22, p. 2]. The goal of embeddings is to create a vector space, in which semantically similar values are also numerically similar [Pil+22]. It can be differentiated between static embeddings and contextualized embeddings. While the former embedding technique always provides the identical

numerical representation for an input value, the latter embedding technique changes the vector representation of a value based on its surrounding context [Pil+22]. This is especially important in the Natural Language Processing (NLP) domain, where contextual embeddings have led to state-of-the-art improvements [Pil+22]. Homonyms or polysemic words like *bat* or *second* are words that carry multiple different meanings and their semantic meaning therefore can change with the context they are used in. Hence, the contextualized embedding vector of the word *second* in the context of *Time* (e.g. *"it took me 3 seconds"*) should be different from the one where *second* is used in the context of *Competition* (e.g. *"he achieved the second place*"). Contextualized embeddings have to be learned during some form of (pre-) training [Dev+19; Iid+21; Den+21]. Static embedding techniques, such as Word2Vec [Mik+13] are learned as well. However, a static embedding would always create the same vector representation for *second*, while a contextualized embedding vector would be different, depending on the context the word *second* is used in. Using the embedding layers to get a vector representation without learning is also possible, which can be seen as a "feature tokenization" [ZC22; Gor+21]. However, semantic similarity in a vector space cannot be achieved without any learning, so this embedding technique is more similar to a discretization/tokenization technique.

Multi-Dimensional Encodings: Multi-Dimensional encoding techniques focus on encoding an entire record (or table-row) into another representation [Bor+22]. The VIME framework [Yoo+20] uses a multilayer perceptron as an encoder that encodes a tabular input row into a latent representation, similar to the encoder of famous (variational-) auto-encoders [KW13] architectures. This latent representation is, like in embeddings, a learned representation that is learned through self-supervised learning tasks, feature vector estimation and mask vector estimation [Yoo+20]. [Bor+22] also lists other multi-dimensional encoding techniques, such as the works of [Zhu+21] and [Sun+19]. Both architectures transform tabular data into a two-dimensional structure (e.g. the format of images) to enable the effective use of Convolutional Neural Networks (CNNs).

Data Normalization and Standardization

Normalization and standardization techniques are applied to numerical columns and scale the values so that their distribution is adjusted [Gar+16]. This is especially important in tabular data, where numerical features are likely to have a different scale, e.g. a column *Age* could have a range of 1 to 100 and a column *Capital-gain* could have a range from 0 to 100.000, as it is the adult income dataset [DG17] (Section 5.3). Normalization allows to scale each column a similar range (usually 0 to 1, or -1 to 1) while maintaining the overall distribution of each column [Izo+22]. Standardization on the hand rescales the data so it follows a normal distribution with mean 0 and standard deviation 1 [dev23b]. Standardization and Normalization of values is especially important if the data will be used in a machine learning context, where machine learning models might behave unexpectedly if the features do not look similar to a Gaussian distribution or are not ranged

between 0 and 1 [Ped+11; dev23b]. Through this data rescaling, the sensitivity of models to large values is reduced and the generalizability is increased [Izo+22]. Two popular standardization and normalization approaches are the standard and the min-max scaler, respectively. The standard scaler standardizes a feature value x_i according to the mean and standard distribution of the feature column so that it follows a normal distribution [Gar+16; Izo+22]. It can be defined as $x' = \frac{x_i - mean(x)}{std(x)}$, with x' as the standardized x_i value [Izo+22]. The min-max scaler is a normalization technique that makes use of the maximum and minimum of each feature column and scales the values accordingly. It is defined as $x' = \frac{x_i - min(x)}{max(x) - min(x)}$ [Izo+22]. While the standard scaler is usually used for data that follows a Gaussian normal distribution, the min-max scaler is better suited for arbitrary distributions, however, it is quite affected by outliers [Cho20]. A more complex normalization technique addresses the problem that numeric values in tabular data can follow distributions in a more complex way than just a simple Gaussian normal distribution [Zha+22; Xu+19]. Xu et al. [Xu+19] introduces mode-specific normalization which models complex distributions with multiple simple Gaussian distributions. The authors define mode-specific normalization in the following way [Xu+19, p. 3-4]:

1. For each continuos column C_i, estimate the number of modes m_i by using a Variational Gaussian Mixture model (VGM) [Bis06]. For example, in Figure 2.1 the VGM estimates $m_i = 3$, referred to as η_1, η_2 and η_3. The learned Gaussian mixture is $\mathbb{P}_{C_i}(c_{i,j}) = \sum_{k=1}^{3} \mu_k \mathcal{N}(c_{i,j}; \eta_k, \sigma_k)$ with μ_k and σ_k as the weight and the standard deviation of a mode respectively and normal distribution $\mathcal{N}$. μ_k can also be understood as the proportion of the overall data that comes from the kth mode, essentially measuring the likelihood, a randomly selected datapoint from the overall dataset comes from the kth determined Gaussian distribution.

2. For each $c_{i,j}$ the probability is calculated of $c_{i,j}$ coming from each mode. In Figure 2.1, ρ_1, ρ_2 and ρ_3 denote the probability densities, calculated as $\rho_k = \mu_k \mathcal{N}(c_{i,j}; \eta_k; \sigma_k)$.

3. Given ρ_1, ρ_2 and ρ_3 for $c_{i,j}$ one mode is sampled and used to normalize the value. In the example of Figure 2.1, ρ_3 is most likely and is sampled. The mode selection for $c_{i,j}$ is represented as a One-Hot vector $\beta_{i,j} = [0, 0, 1]$ indicating that the third mode has been sampled. The actual value $c_{i,j}$ is normalized as $\alpha_{i,j} = \frac{c_{i,j} - \eta_3}{4\sigma_3}$, according to the mean and standard deviation of ρ_3.

The normalized representation r of $c_{i,j}$ will be the concatenation of $\alpha_{i,j}$ and $\beta_{i,j}$:

$$r = \alpha_{i,j} \oplus \beta_{i,j}$$

For example, given $\alpha_{i,j} = 0.4$ and $\beta_{i,j} = [0, 0, 1]$, the normalized representation would be

$$r = [0.4, 0, 0, 1]$$

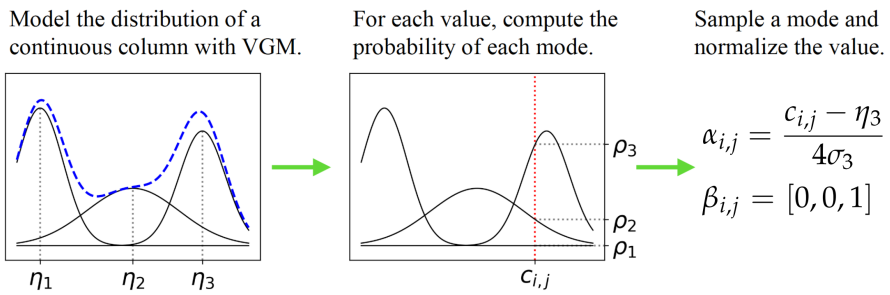


Figure 2.1: An example of mode-specific normalization [Xu+19, Figure 1, p. 4]

Another approach to standardize the range of a column between 0 and 1 is a Quantile-transformation, which is a non-linear transformation [dev23b; Kot+22]. The Quantile-transformation allows the mapping of all values of a random variable into a desired output distribution, such as the uniform or normal distribution, using the quantile function on the inverse cumulative distribution of the random variable. This transformation preserves the rank order of the values but distorts correlations and distances within and across features [dev23b].

2.2 Data Synthesis

2.2.1 Synthetic Data

Synthetic data can be defined as "(...) artificially generated data, that are modeled on real data, with the same structure and properties as the original data (...)" [Kal+20, p. 2] but without containing any actual specific information or entries of the actual real data. While first synthetic data approaches [GR92] focused on imputation techniques to generate synthetic data, the advent of deep learning has led to the topic gaining importance [KWT22; Kal+20]. Acquiring and collecting real data can be costly [Pan22] or may simply not be possible due to the sensitivity of the data (e.g. medical records) [EHR17] or due to regulatory restrictions, such as the GDPR [Eur16]. Synthetic data on the other hand is, compared to real data, cheap to generate [LL21], fulfils regulatory and privacy constraints [Zha+22] and can substitute real data in software development for testing [WHV08] or machine learning training [Pan22]. Hence, synthetic data could be used as an alternative to real data in various use cases. Access to data is still one of the biggest bottlenecks when developing machine learning or deep learning models [Fan+20]. Synthetic data could be used to increase the data quality, by rebalancing skewed datasets [Zha+22] or increasing the dataset size as additional training data or in combination with the real data [LL21; Kim+21]. An example, where working with real data is sometimes not possible, due to privacy or availability reasons could be software development [YDvdS20; WHV08]. In software development, high-quality test data is crucial for development but challenging and time-consuming to generate [WHV08]. Developers' time to create such datasets is

usually scarce and it might be the case that developers do not even have permission to see the data, due to its sensitivity [WHV08]. A synthetic data generation model would possibly allow developers to generate test data with predefined characteristics without it being restricted in any form.

2.2.2 Synthetic Tabular Data Generation

Generative models have recently received a lot of attention, thanks to their impressive results in generating a variety of data types, such as images [HJA20; DN21; Rom+22], videos [Ho+22; Run], text [Rad+18; Ope22], or music [Ago+23; FM22].

While synthetic data generation can be applied to various types of data, generating synthetic tabular data is challenging due to the complexity of the underlying joint distributions between variables [Bor+22]. This works formal definition of the tabular data generation is adapted from [Xu+19, p. 2] and expands the definition of a table from Section 2.1:

> A data synthesizer G is trained on a table T and then used to generate a synthetic table T_{syn}. T is partitioned into a training set T_{train} and test set T_{test}. The synthesizer G is trained on T_{train} and afterwards used to sample independently individual rows to create T_{syn}. Finally, the similarity of T_{syn} to T_{test} is evaluated using a selected similarity evaluation technique.

The synthetic dataset T_{syn} should exhibit a similar relationship with the test dataset partition T_{test} as the one found between the training dataset partition T_{train} and T_{test}. This is crucial for maintaining the underlying structure and properties of the original data. The primary objective for the generative model G is to create T_{syn} in such a way that it appears to originate from the same joint distribution as the complete dataset T. This ensures that the synthetic data maintains the essential characteristics of the original data. Importantly, T_{syn} is not directly compared to T_{train}, because the purpose of generating T_{syn} is not to merely replicate T_{train}, but to create a new dataset that preserves the inherent relationships and properties of the original data.

During the generation of synthetic tabular data, there are two competing objectives: high data utility and low disclosure risk. Data utility is concerned with how well the synthetic data accurately reflects the real data in terms of its underlying patterns and relationships. Low disclosure risk on the other hand is concerned with the preservation of privacy and confidentiality of the information in the real training dataset T_{train} [Lit+21].

Tabular data synthesis approaches can be divided into process- and data-driven methods [Gon+20]. Process-driven models focus on simulating real-world processes and gather data during that simulation to create the synthetic dataset [Gon+20; KWT22]. On the other hand, data-driven approaches aim to generate synthetic data based on already existing real-world datasets [Gon+20; KWT22]. This can be achieved by augmenting the existing dataset, i.e. applying transformations to the existing individual datapoints

to generate new datapoints [KWT22]. Another technique to generate synthetic data is to simply sample from the individual feature distributions of a dataset to create new synthetic data [KWT22]. Lastly, machine learning and deep learning algorithms can be used to learn the underlying joint distributions of the real data to generate new synthetic data [KWT22]. Compared to other data types, the synthesis of tabular data is especially challenging due to its heterogeneity, potential weak correlations among features and the dependency on preprocessing [Bor+22; Yoo+20; GRB22]. To create a realistic synthetic dataset, interdependencies that may exist between variables must be captured and it is possible that the model overfits the training data and generalizes relationships between features that are only present in training but not in the test data [LHB22]. In addition to that, tabular data columns can follow complex distributions, which makes the synthesis challenging. Zhao et al. [Zha+22] categorized the following four distribution types a tabular data column might display, which one should consider when trying to synthesize tabular data:

1. **Single Gaussian variables**: Data whose distribution follows a Gaussian distribution [Zha+22].

2. **Mixed data type variables**: A single column can contain a mix of continuous and categorical values. For instance, a *Loan* column could hold information about the size of an individual's loan. The values in this column are mostly positive float values but can also take the value -1 which indicates that this person did not get approved for a loan. Such a special categorical value in an otherwise purely numerical column needs to be considered when working with this column [Zha+22].

3. **Long tail distributions**: In real-world data, the class distribution is often imbalanced, where a few classes of a column occur much more frequently than most other classes [Zha+23]. When plotting the number of samples per class in a descending manner, the resulting plot shows a large "head", where a few classes have a high number of samples, and a "tail", where the remaining classes are the majority but are each represented only very limited in the dataset [Zha+23, p. 2]. This results in a short "head" but long "tail" distribution, giving the distribution its name. Capturing these long-tail distributions correctly during the synthesis is important, not only because they represent unique or rare scenarios that could be crucial for certain analysis tasks, but also because the minority classes in the tail collectively constitute a significant portion of the entire dataset [ZHW18].

4. **Skewed multi-mode continuous variables**: Complex numerical distributions do not necessarily follow a single Gaussian distribution. It is possible for their distribution to consist of multiple unknown distributions that are potentially skewed as well. Their conjunction as a whole makes up the complex distribution [Zha+22], as it could be seen in the example of Figure 2.1.

A successful tabular data generator should be able to recreate such properties of real data in their synthetic data counterpart.

2.3 Deep Learning Architectures

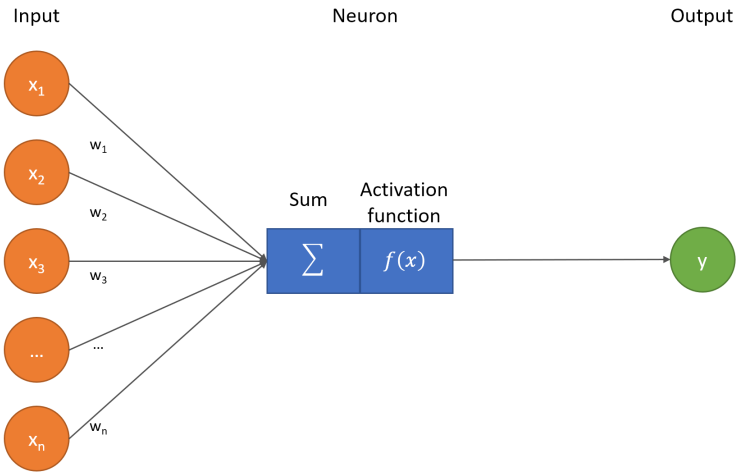
Deep learning has rapidly become a dominant force in artificial intelligence, revolutionizing various fields including computer vision, NLP, and generative modeling. Here, complex neural networks architectures emerged with multiple layers and complex interconnections. These architectures have enabled the creation of highly accurate and efficient models that are capable of processing vast amounts of data from different modalities.

In recent years, this development of new deep-learning architectures has been a key area of research, leading to the creation of such powerful models. This section will start with a high-level introduction into a neural network's most important architectural building blocks. Subsequently, the most important deep learning model architectures for generating data are presented.

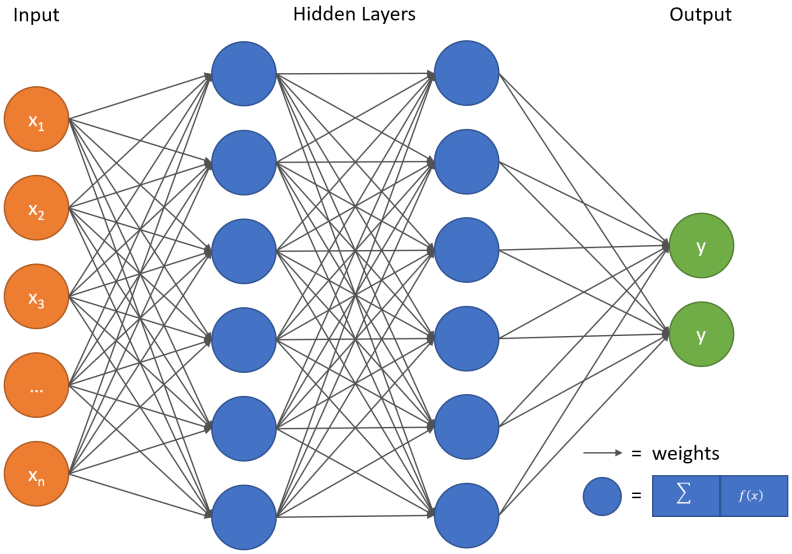
2.3.1 Neural Networks

The most basic neural network is called a perceptron Figure 2.2a [Ros58]. It consists of a single input layer that is connected to an output node. Each input node is connected with the output node via weight edges. For all inputs, the input value is multiplied by each respective weight. The sum of all input-weights' multiplication serves as the input to an activation function, which is different depending on the task at hand. The output is the prediction of the perceptron which is compared to the actual result, the label, and an error is calculated. This error is later used to update the individual nodes' weights via backpropagation [Agg18]. The output of one node can also serve as the input to another node, so multiple layers of nodes are created. This is referred to as a Multilayer Perceptron (MLP) or feed-forward neural network (as presented in Figure 2.2b), because the information flows forward from input through the multiple layers in the middle (also called hidden layers) to the output [Agg18; GBC16]. An MLP tries to approximate some function f^* by defining a mapping $y = f(x;\theta)$ and learning the values of the parameters θ (i.e. the weights) that best approximate f^* [GBC16]. A simple MLP is already able to learn almost any function, which is why they are also considered to be universal function approximators [Agg18; HSW89].

Convolutional networks, or CNNs [Lec+98], are a specialized type of neural network. As their name indicates, they perform convolutions on the data as illustrated in Figure 2.3a. Convolution can be defined as "a dot-product operation between a grid-structured set of weights and similar grid-structured inputs drawn from different spatial localities in the input volume" [Agg18, p. 316]. In other words, a small grid of numbers called *weights* slides over a larger grid of numbers, which is in the context of deep learning usually an image. In each position, the numbers in the small grid are multiplied by the corresponding



(a) Perceptron



(b) Multilayer Perceptron (MLP)

Figure 2.2: Overview Perceptron

numbers they cover in the large grid, and these results are added together. This process is repeated across different areas by sliding the small grid over the large grid repeatedly until the complete large grid was processed. The weights within the small grid, often referred to as the *kernel*, get adjusted during training of the CNN during backpropagation. This fine-tunes these weights based on how far off the network's output is from the expected output, enabling the convolutional network to improve its performance over time [Agg18; GBC16].

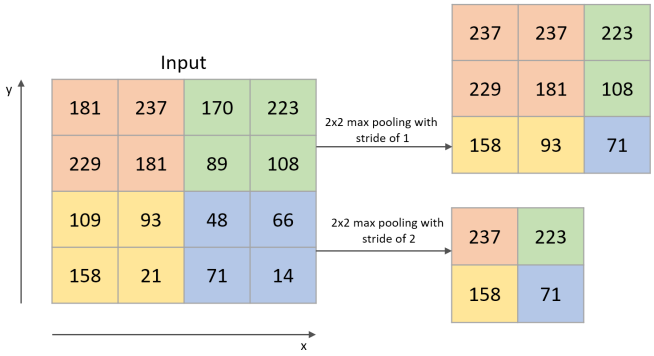
CNNs perform exceptionally well on data with a grid-like topology with strong spatial dependencies such as image or time-series data [Agg18; GBC16]. In a typical convolutional network, a layer starts by performing multiple convolution operations in parallel on the data. The linear activations that are produced by the convolutions are sent through a non-linear activation function (usually a Rectified Linear Activation Unit (ReLU)). Lastly,

a pooling function is applied to the output of the activation function [Agg18]. A pooling function reduces the spatial size of the created feature maps while retaining their essential features [GBC16; Agg18]. Goodfellow, Bengio, and Courville [GBC16, p. 335] also describe it as a type of "summary statistic of the nearby outputs". Pooling allows the features extracted by the CNN to be less sensitive to changes in the position of the input data. This property is known as translation invariance and is one of the reasons why CNNs perform so well with data types, whose format has local spatial dependencies [GBC16; Agg18]. Another way to reduce the dimensionality is to increase the convolution stride. A stride refers to the step size or the number of pixels by which the filter, also known as the convolutional kernel, moves across the input feature map during the convolution operation [Agg18]. The effect of using a stride greater than 1 is that it reduces the spatial dimensions of the output feature map. Specifically, the height and width of the output are determined by the formula $(L_q - F_q)/S_q + 1$, where L_q is the height (or width) of the input, F_q is the size of the filter, and S_q is the stride in layer q [Agg18, p. 324].

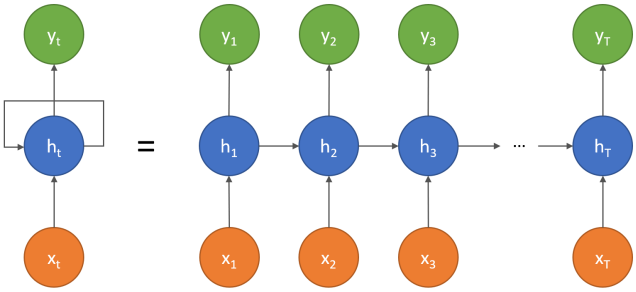
In feed-forward neural networks, information flows forward through the network from input to output node, hence, there are no internal connections that allow the output of the model to be fed back into the network as inputs [Agg18; GBC16]. When such connections are added to the network architecture, the resulting models are referred to as Recurrent Neural Networks (RNNs) [RHW86b]. RNNs are designed to process sequential data, like sentences or time-series data, where the size of the sequence does not have to be fixed. In CNNs, the learned parameters are shared by using the same convolution kernel on each subset of neighbouring input data at each time step (Figure 2.3b). As a result, a succession of output values is produced, each of which is dependent on a few close input values. RNNs share parameters differently since each output value depends on its predecessors' output. This enables parameter sharing via a deep computational graph by allowing the same update algorithm to be applied to all outputs. When processing a sequence, e.g. a sentence, an RNN receives one datapoint, e.g. a word, at each timestep. The network itself has a so-called *hidden state* that is kept throughout all timesteps and changes with each timestep. This, however, means that the output y_3 of the input at $x_{t=3}$ depends on the hidden states h_0 through h_3 and input x_0 through x_3. As a result, the computational graph on which the backpropagation is performed on increases as well [Agg18]. A large computation graph introduces new practical issues, such as an increased computation time and during the training, helpful information might not be propagated from output to the end of the model, due to the vanishing and exploding gradient phenomenon [Agg18]. To address these issues, further improvements to the architecture have been made, such as Long Short-Term Memorys (LSTMs) [HS97] or Gated Recurrent Units (GRUs) [Cho+14].

Over time, additional architectural design choices have been discovered that have led to further improvements and solved other issues that emerged during the development of neural networks. One such design is the ResNet [He+16] which introduced residual connections between layers as depicted in Figure 2.3c [He+16; Agg18]. The connections,

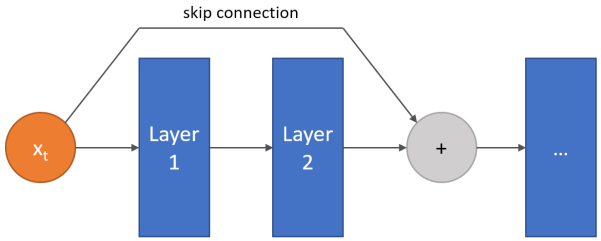
also called skip connections, allow information to be copied over layers and allow the backpropagated gradient information to skip specific layers, if necessary [He+16; Agg18]. This is achieved by adding the output of one layer directly to the output of a later layer, bypassing one or more intermediate layers. This way, even a very small gradient can flow more easily back through the network during backpropagation of the gradient, alleviating the vanishing gradient problem and enabling better learning capabilities [He+16; Agg18]. The decision of which layers to skip is learned by the model itself during the training through the backpropagation algorithm [He+16; Agg18].



(a) Convolution
(adapted from [Ros21])



(b) RNN
(adapted from [Rad17])



(c) Residual Connection
(adapted from [Has19])

Figure 2.3: Overview Neural Network Architectures

Lastly, inspired by the human cognitive capabilities to focus on specific parts of an input, attention mechanisms have been introduced [NZY21; Agg18]. The implementation of attention in neural networks tries to address the problem of information overload[1] and is supposed to enhance the network's ability to focus on relevant portions of the input data [NZY21]. An attention mechanism has been successfully applied to various tasks in different domains, such as computer vision or NLP [NZY21]. The architectural realization of attention is realized differently, depending on the task at hand [Agg18]. The attention mechanism in computer vision involves weighting different regions of the image based on their relevance to the task [Agg18]. This can be achieved through techniques such as spatial attention, channel attention or temporal attention [Guo+22]. In NLP, attention mechanisms try to improve to capture the meaning of a sentence [NZY21]. For instance, this has been achieved through the introduction of a context vector in an RNN encoder-decoder [BCB15] or through self-attention [Vas+17] (see Subsection 2.4.3 for a detailed explanation of self-attention).

2.4 Deep Generative Models

Deep generative models have revolutionized the field of artificial intelligence and machine learning by generating new and original data based on existing patterns and structures. For example, deep generative models have been able to generate new images [HJA20], videos [Ho+22], text [Ope22], or music [Ago+23], based on previously learned data. Generative models try to learn a joint distribution over all the available variables in a dataset [KW19], which is also called the *generative modeling problem* [Goo+20]. Formally, in generative modeling training examples x are drawn from an unknown distribution $p_{data}(x)$ and a generative modeling algorithm is trained to learn a distribution density function $p_{model}(x;\theta)$ that approximates the true probability density function $p_{data}(x)$ as closely as possible [Goo+20, p. 139]. In the generative modeling process, model parameters denoted by θ are essential as they shape the output of the model. These parameters are typically fine-tuned through an optimization process that seeks the most suitable θ values, aiming to enhance the resemblance between p_{model} and p_{data} [Goo+20, p. 139]. Several powerful generative modeling approaches with different architectures have emerged, each with its strengths and weaknesses. This section will explore four of the most popular approaches: Variational Autoencoders (VAE) (Subsection 2.4.1), Generative Adversarial Network (GAN) (Subsection 2.4.2), transformers (Subsection 2.4.3), and diffusion models (Subsection 2.4.4).

[1] Information overload in the context of deep learning typically refers to issues that arise when a model is fed with too much data, that can hinder the model to capture the relevant information. Issues during training may arise if the input data contains too much noise, that is of no relevance or might cause the model to rather learn to predict the noise, than the actual target. With this, the model may overfit the training data and perform poorly on the test data.

2.4.1 Variational Autoencoders

In order to understand VAEs [KW13], an introduction to vanilla autoencoders is required. Autoencoders, firstly introduced in [RHW86a], are a neural network architecture designed to generate a copy of the input [GBC16; BKG20]. This is done by compressing the original input x into a meaningful representation by constricting the number of nodes in the intermediate hidden layer [Agg18; BKG20], also called bottleneck. Figure 2.4a shows that an autoencoder consists of two deep-learning modules, the encoder, and the decoder. The encoder function (or generative model [KW19]), $z = g(x)$ receives an input x and creates a latent representation z (also called "code" [Agg18]) with a dimensionality smaller than x [GBC16; BKG20]. The decoder (or recognition model [KW19]) receives the latent representation z and tries to reconstruct the original input, $r = f(z)$ with the same dimensionality as x [GBC16]. The reconstruction of the input x, designated as r, is also commonly referred to as x'. The objective is to ensure that x' is as similar as possible to the original input x. For the autoencoder to learn and optimize the model's parameters, a loss L is calculated by applying a reconstruction loss function, which measures how close the reconstruction is to its original counterpart [BKG20; MMS22]. The reconstruction loss depends on the use case, for instance, it could be the Mean-squared error between x and r, $L = \sum_{i=1}^{N}(x_i - r_i)^2$ for N features in x, [Agg18; GBC16]. Since z is restricted in dimensionality, a simple copying from input to output, i.e. $f(g(x)) = x$, should not be possible, and the encoder should learn to create an approximation of x with its most essential features [Agg18]. Therefore, autoencoders are a dimensionality reduction technique. The latent representation of a complete linear autoencoder without any non-linear operations achieves the same representation as a Principal Component Analysis (PCA) [Pla18; BKG20]. Further variations of autoencoders have emerged. Deep autoencoders expand the neural network from one hidden layer to multiple layers with non-linear activation functions, increasing the representation power [Agg18]. Other variants include denoising autoencoders, convolutional autoencoders, or sparse autoencoders [MMS22; GBC16]. However, vanilla autoencoders suffer from a lack of regularity in the latent space [Raz+22]. This means that it is possible, that the encoder may produce a latent space that is not well structured. In other words, similar inputs are not necessarily mapped to similar points in the latent space. The consequence of this irregular latent space is, that points from the latent space may, once decoded, result in a meaningless output [Roc19]. Hence, the autoencoder struggles with interpolation for datapoints that have not been present in the input sequence, since the latent space lacks a meaningful structure [Raz+22].

VAEs increase the reconstruction robustness by letting the decoder sample from continuous distributions [KW13]. By forcing the latent space into a continuous distribution, a regularised latent space is obtained [Roc19]. This is achieved by introducing the Kullback-Leibler (KL) divergence (see Equation 2.2) to ensure that the latent space representation follows a standard Gaussian distribution (zero mean and unit variance) [Raz+22; Agg18]. Instead of the encoder mapping the input into a fixed vector z, it is mapped into a distri-

bution by generating two vectors representing the mean μ and standard deviation σ of the distribution (see Figure 2.4b) [Raz+22; Agg18]. Constraining the hidden representation z to a Gaussian distribution allows the decoder to generate new data by simply feeding samples from the standard normal distribution [Agg18]. However, if every object is generated from the same distribution, distinguishing between different objects or reconstructing them from a given input would be impossible. Thus, the distribution of the encoded information z, conditioned on a specific input x, would differ from the standard normal distribution [Agg18]. Hence, the constraint that the hidden representation z follows a standard normal distribution would not be achieved across the encoded distribution from a single object but rather for the entire dataset [Agg18]. The encoder is represented as $q(z|x)$, creating the conditional distribution from which the latent vector is sampled $z \sim q(z|x)$ [KW13]. However, this stochastic sampling process makes computations non-differentiable. In other words, the backpropagation of the gradient is not possible and our neural networks weights cannot be updated [Agg18]. To address this issue, a "reparameterization trick (Equation 2.1) was introduced [KW13, p. 4]. Since z is normally distributed ($z \sim N(\mu, \sigma^2)$) it can be rewritten as $z = \mu + \sigma\epsilon$ with ϵ used as an auxiliary noise variable $\epsilon \sim N(0, 1)$ sampled from the standard normal distribution [KW13].

$$N(\mu, \sigma^2) = \mu + \sigma\epsilon \tag{2.1}$$

This transformation moves the stochastic sampling process to the fixed ϵ variable. It allows μ and σ to be differentiable variables, which enables the gradient to flow and the neural network to learn [KW13]. In order for the model to learn that z should be close to a normal distribution, a regularization loss is introduced using the KL-divergence.
The KL-divergence is defined as:

$$KL(p \parallel q) = \sum_{i}^{N} p_i \cdot log(\frac{p_i}{q_i}) \tag{2.2}$$

with distributions p and q:

$$L_{regularization} = KL(q(z|x) \parallel p(z)) \tag{2.3}$$

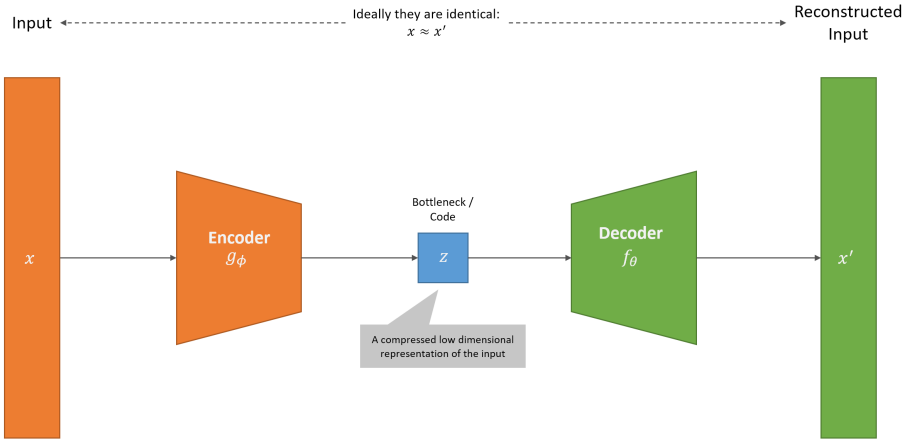
The KL-divergence measures how diverged two probability distributions are from one another. With $p(z)$ as a prior defined standard Gaussian distribution, minimizing this function means optimizing the encoders ($q(z|x)$) parameters of μ and σ towards resembling the distributions it is compared to.
Additionally, a reconstruction loss is computed based on the negative log-likelihood that measures the error between the input and output:

$$L_{reconstruction} = -\mathbb{E}_{q(z|x)}[logp(x|z)] \tag{2.4}$$

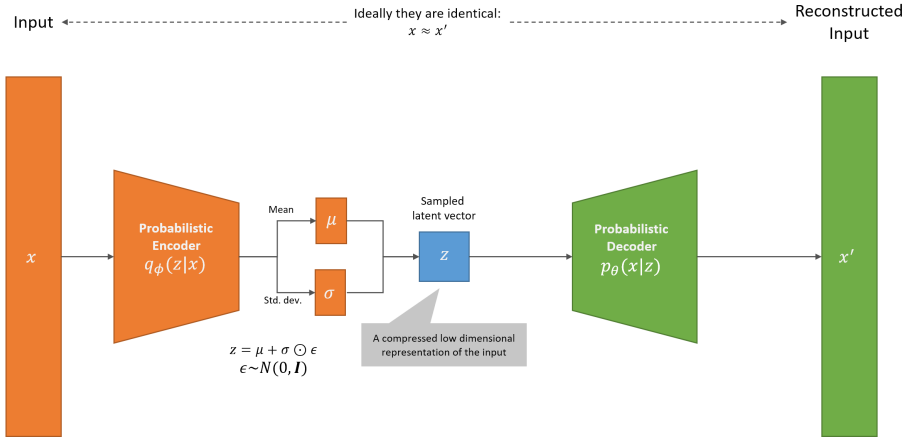
with $E_{q(z|x)}$ denoting the expectation with respect to the encoder distribution.

Hence, the final loss function that is minimized during the training of the VAE is the combination of both loss functions[2]:

$$L_{vae} = \mathbb{E}_{q(z|x)}[logp(x|z)] - KL(q(z|x) \parallel p(z)) \tag{2.5}$$



(a) Autoencoder



(b) Variational Autoencoder

Figure 2.4: Architecture of Autoencoder and Variational Autoencoder (adapted from [Wen18])

[2]The loss components in the original work [KW13] are formulated as a maximization objective. In practice, a loss that can be minimized during training is required. Hence, we convert the maximization objective to a minimization objective by multiplying it with -1.

2.4.2 Generative Adversarial Networks

GANs [Goo+14] have been introduced in 2014 as a new architecture to address the generative modeling problem. The architecture of GANs is based on game theory where two models compete in a zero-sum min-max game [Goo+14; Zha+22]. A zero-sum min-max game, in the GANs, refers to the adversarial interaction between two models - the generator and the discriminator - where one model's gain (minimizing its loss) corresponds to the other model's loss (maximizing its loss), such that the total sum of their gains and losses is always zero. While the generator is tasked to generate realistic-looking data points, its adversary, the discriminator is tasked to discriminate whether a given datapoint is a real datapoint or a fake datapoint produced by the generator. The generator function $G(z;\theta_G)$ with learnable parameters θ_G receives as input a vector z and is tasked to transform z into a realistic-looking sample x. An input vector z is drawn from a prior distribution $p(z)$, usually a Gaussian or uniform distribution, and is just an unstructured noise vector. The fake data sample $x = G(z)$ "is intended to be statistically indistinguishable from the training data" p_{data} [Goo+20, p. 141]. The discriminator function $D(x;\theta_D)$ with learnable parameters θ_D receives as input samples x that could either stem from p_{data} or from p_{model} generated by the generator. Like a binary classifier, it tries to predict whether a given sample x is real or fake [Goo+14]. The discriminators' output classification is finally used in the backpropagation algorithm to update not only the discriminators parameters θ_D, but also the generators parameters θ_G. So while the generator tries to produce real-looking fake data samples, the discriminator tries to tell real from fake data samples apart. This type of training can be seen as a mentor (discriminator) providing feedback to a student (generator) on the quality of his work [Zha+22]. Formally, the objective function J_D for the discriminator that will be maximized if real x are correctly classified to 1 and fake x are correctly classified to 0 can be defined as [Agg18]:

$$Maximize_D J_D = \sum_{x \in p_{data}} log[D(x)] + \sum_{x \in p_{model}} log[1 - D(x)] \tag{2.6}$$

with the first part of Equation 2.6 handling the real data samples and the second part the synthetic data samples. Hence, the discriminator tries to maximize correctly predicting real and fake samples as real and fake, respectively.

The generator, on the other hand, tries to fool the discriminator. Therefore, the generator tries to minimize the likelihood that the generated fake samples are flagged as fake. Its objective function J_G is defined as [Agg18]:

$$Minimize_G J_G = \sum_{x \in p_{model}} log[1 - D(x)] = \sum_{x \in p_z} log[1 - D(G(z))] \tag{2.7}$$

Ultimately, the generator and the discriminator are playing a two-player minimax game with the following value function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(G(Z)))] \tag{2.8}$$

Equation 2.8 is used to formulate the loss functions that are used in training. However, [Goo+14] notes that in practice, Equation 2.8 "may not provide sufficient gradient [for the generator] to learn well" [Goo+14, p. 3]. This seems to be caused by the fact that the discriminator is able to reject the early samples from the generator with high confidence, which results in poor learning capabilities [Goo+14]. The proposed solution for this is to change the objective function for the generator from the minimization function Equation 2.7 to a maximization of $logD(G(z))$. Thus, instead of minimizing the discriminator being correct, the new objective is to maximize the discriminator being wrong [Goo+14]. Goodfellow et al. [Goo+14, p. 3] observes, that this change "provides much stronger gradients early in learning" and therefore allows for an overall better model learning. This heuristic means that the problem is technically no longer a minimax optimization problem, but is usually favored as it enhances the training process [Agg18, p. 442]. The two loss functions L_G and L_D that are minimized during training of the generator and discriminator, respectively, are defined as:

$$\begin{aligned} L_D &= logD(x) + log(1-D(G(z))) \\ L_G &= -log(D(G(z))) \end{aligned} \tag{2.9}$$

In practice, the discriminator is trained for k steps for each generator training step [Agg18]. This process is repeated iteratively until a Nash-equilibrium is reached, at which point the discriminator is unable to tell real and fake samples apart ($p_{model} \approx p_{data}$) [Goo+14; Agg18].

2.4.3 Transformers

In 2017 Vaswani et al. [Vas+17] proposed the *transformer* architecture for sequence modeling tasks. While in this domain RNN based approaches have been established state-of-the-art techniques, the newly proposed *transformer* architecture can be trained in a parallel manner, overcoming the central issue of RNN-based models, their high computational demand. *Transformers* have been successfully adapted in multiple domains, such as NLP [Gil+20], computer vision [Kha+22], audio processing [Gon+22] or tabular data [Hua+20] and show properties of a universal computation engine [Lu+21; Lin+22]. This is achieved through the combination of several mechanisms, namely, self-attention, multi-head attention, embeddings, and positional encoding [Vas+17].

The transformer model follows an encoder-decoder structure (see Figure 2.5). The encoder maps an input sequence to an abstract continuous representation that encapsulates all the learned information from the input. The decoder, in turn, utilizes this representation

to generate outputs while also incorporating the previously produced output. The output of the decoder is a probability score for each possible token. These probabilities are then used to determine the next token, as the token with the highest probability is the next token [Vas+17].

The first step in the transformer architecture is the transformation of the raw input through an embedding layer (see Subsection 2.1.1). This embedding allows the transformation of input tokens into vectors of dimension d_{model} [Vas+17]. While the encoder produces embeddings just based on the input sequence, the decoder operates in an autoregressive manner, generating embeddings not only from the encoded output it receives from the encoder, but also considering its own previously produced outputs in the sequence generation process [Vas+17]. Embeddings of encoder and decoder are combined with so-called positional encodings [Vas+17; Lin+22]. Positional encodings have the same dimensionality as the embeddings and carry information about the positions of the tokens in the overall sequence. [Vas+17] makes use of a *sin* and *cos* function to compute this vector. The sum of the positional encoding vector and the embeddings forms the new input to the encoder or decoder [Vas+17].

The encoder and decoder consist of multiple blocks, named encoder-blocks and decoder-blocks, respectively. Each encoder block is based on a multi-head self-attention module, followed by a position-wise feed-forward network. Residual connections are employed around the two sublayers, followed by a layer normalization, to facilitate building a deeper network [Vas+17; Lin+22].

A decoder block is similar to the encoder block; however, an additional multi-head attention module is added at the beginning in front of the other two-sub layers. Additionally, the self-attention modules in the decoder are adapted in such a way that positions are not able to attend subsequent positions (referred to as "masking" [Vas+17, p. 3]). This ensures that predictions for a position can only depend on known outputs [Vas+17; Lin+22].

Self-attention allows the model to associate individual words in the input with other words in the input. This attention mechanism is realized using a scaled Dot-Product on query, key, and value vectors (Q, K, and V respectively) [Vas+17]. All three vectors are created by sending the input through different linear layers. The dot product multiplication of the query and key matrixes creates a score matrix that indicates how much focus an individual word should have on the other words, i.e. how much the network should attend to it [Vas+17]. The created attention matrix is scaled (according to the key dimensionality d_k) and multiplied with the value vector to generate the output probabilities. Multi-head attention refers to the fact that input is processed simultaneously by different attention heads, i.e. with multiple Q, K, and V matrices as described above [Vas+17]. The Outputs of the multiple self-attention heads are ultimately concatenated back together [Vas+17].

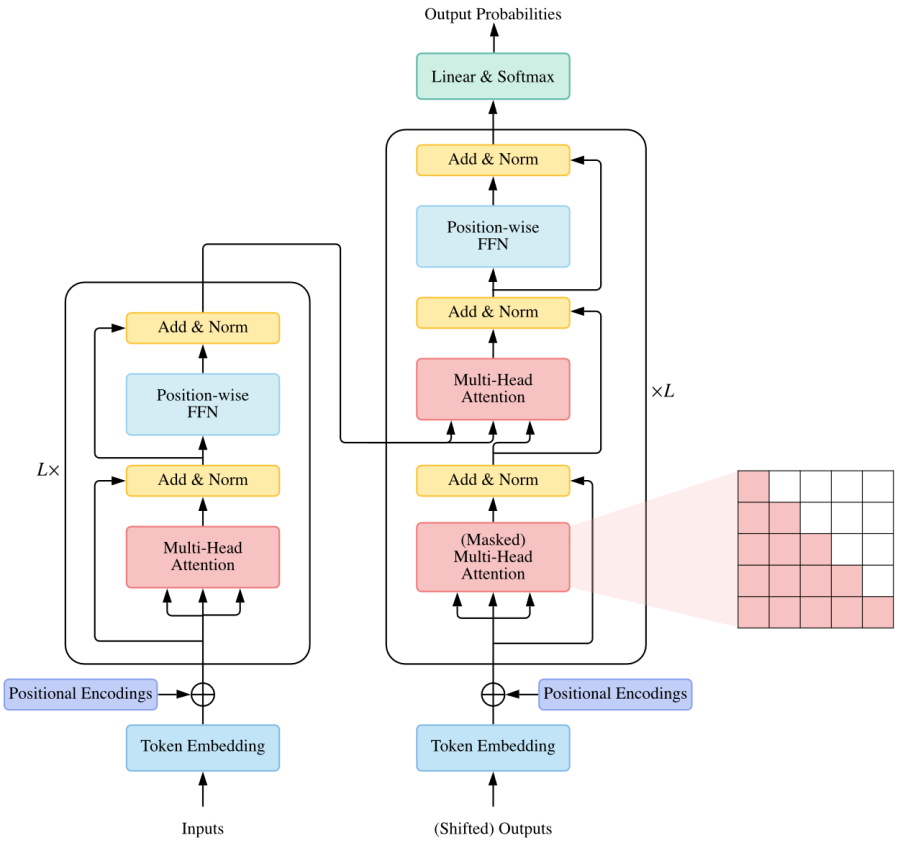


Figure 2.5: Transformer Architecture [Lin+22, Figure 1, p. 3]

The attention mechanism can be defined as [Vas+17, p.4]:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.10}$$

To summarize, the transformer model employs an encoder-decoder structure for sequence modeling tasks. The encoder receives an input sequence and transforms it into an abstract continuous representation via an embedding layer, which includes positional encodings. This encoded input, containing the attention information from the input, is passed to the decoder. The decoder operates in an autoregressive manner, incorporating both the encoded input and a sequence of previously generated outputs in its process. The output of the decoder is a probability vector, with each element representing the probability of a specific possible token. The token corresponding to the highest probability value is then chosen as the next output token. Ultimately, the decoder is subsequently used to generate output tokens until it produces an ending token. Both the encoder and decoder utilize self-attention mechanisms, allowing for context-dependent concentration on different parts of the input sequence.

2.4.4 Diffusion Probabilistic Models

The Denoising Diffusion Probabilistic Models (DDPMs) (from now on referred to as "diffusion model"), firstly introduced by Sohl-Dickstein et al. [Soh+15], is a generative modeling approach inspired by non-equilibrium statistical physics [Soh+15]. Ho, Jain, and Abbeel [HJA20] in their work, how diffusion models can successfully generate high-quality synthetic images. This section will provide an explanatory introduction to the diffusion model concept, drawing heavily on the foundational work of [HJA20]. Although several other researchers have made notable advancements to diffusion probabilistic models, the work presented in [Soh+15; HJA20] serves as the cornerstone upon which all subsequent improvements have been built. Improvements and alternations made to [HJA20] will be discussed in detail in Section 3.3.

On a high level, the whole diffusion concept can be summarized as follows:

> "The essential idea, (...) is to systematically and slowly destroy structure in a data distribution through an iterative forward diffusion process. We then learn a reverse diffusion process that restores structure in data, yielding a highly flexible and tractable generative model of the data." [Soh+15, p. 1]

Formally, [HJA20] defines diffusion models as a class of latent variable models of the form $p_\theta(x_0) := \int p_\theta(x_{0:T})dx_{1:T}$, with $x_1, ..., x_T$ as latents of the same dimensionality as samples from the data distribution $x_0 \sim q(x_0)$. Diffusion models consist of a forward process and a reverse process. During the forward process, an initial data point x_0 is obtained from the data distribution $q(x_0)$. In the forward process (also called diffusion process), a fixed Markov chain gradually adds noise to the data. T defines the number of noising steps and $x_1, ..., x_T$ denotes the latent variables, i.e. the noised versions of the original data, with x_T being the fully noised version of the data [HJA20]. The amount of added (Gaussian) noise in each timestep varies depending on the current timestep, which is controlled by a variance schedule $\beta_1, ..., \beta_T \in (0, 1)$ [HJA20]. The forward process is defined as:

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}), \quad q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{2.11}$$

For the variance schedule it holds that if $\beta_1 < \beta_2 < ... < \beta_T$ for $t \to \infty$, the final latent x_t is isotropic Gaussian noise, i.e. $x_t \sim \mathcal{N}(0, \mathbf{I})$, with identity matrix $\mathbf{I}$ [Zbi22]. β increases linearly in the original DDPM approach [HJA20] but was later improved to follow a cosine schedule [ND21], improving the overall quality of the model.

Equation 2.11 [HJA20] shows how iteratively one noising step can be applied to get from x_{t-1} to x_t with $q(x_t|x_{t-1})$. Through the use of a reparameterization (see Equation 2.1) and defining $\alpha := 1 - \beta_t$ and $\overline{\alpha_t} := \prod_{s=1}^{t} a_s$, the forward diffusion process from an unnoised input x_0 to a noised version at t can be expressed as follows:

$$q(x_t|x_0) \quad := \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{2.12}$$

Equation 2.12 [HJA20] allows computing the forward noising from x_0 to any timestep t within one computation step instead of iteratively applying noise over and over again.

During the reverse process, the goal is to learn the reverse distribution from noised data x_t to slightly less noised data x_{t-1}, $q(x_{t-1}|x_t)$. With bayes-theorem we can see that $q(x_{t-1}|x_t) = \frac{p(x_{t-1},x_t)}{p(x_t)}$, which means $q(x_{t-1}|x_t)$ depends on the marginal distribution of x_t, denoted as $p(x_t)$. To calculate the marginal probability distribution $p(x_t)$ integration over all possible realizations of x_{t-1} would be necessary, which would be computationally intractable [Cap22]. As a result, $q(x_{t-1}|x_t)$ is approximated by a neural network with learnable parameters θ that learns the joint distribution $p_\theta(x_{0:T})$. $p_\theta(x_{0:T})$ is a Markov chain, defined in Equation 2.13, with the transitions defined in Equation 2.14 [Cap22; HJA20].

$$p_\theta(x_{0:T}) := p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t), \tag{2.13}$$

$$p_\theta(x_{t-1}|x_t) := \mathcal{N}(x_{t-1}; \mu_\theta(x_t,t), \Sigma_\theta(x_t,t)) \tag{2.14}$$

While the mean $\mu_\theta(x_t,t)$ is learned by the neural network, the variance $\Sigma_\theta(x_t,t)$ stayed fixed in the initial work by [HJA20] but was later in Nichol and Dhariwal [ND21] work a learnable parameter as well [Zbi22].

Like in VAEs (Subsection 2.4.1), the goal is to maximize the log-likelihood $log(p_\theta(x_0))$. $p_\theta(x_0)$ depends on all timesteps before x_0 and is, therefore, not tractable in practice [Zbi22]. As a solution, the variational lower bound on the log-likelihood can be computed as illustrated in as the objective function Equation 2.15:

$$log(p_\theta(x_0)) \geq log(p_\theta(x_0)) - KL(q(x_{1:T}|x_0) \parallel p_\theta(x_{1:T}|x_0)) \tag{2.15}$$

The right-hand side of the inequation is increased by decreasing the KL-divergence (Equation 2.2), as the KL-divergence is a non-negative measure of dissimilarity between two probability distributions. In this case, for the KL-divergence to be small, the approximated distribution $p_\theta(x_{1:T}|x_0)$ needs to be similar to the true posterior distribution $q(x_{t-1}|x_t)$. As the inequation has to hold, maximizing the right-hand side means that the left-hand side will automatically be maximized as well. This implies that the objective function provides a lower bound on the log-likelihood and any improvement in minimizing the KL-divergence will ultimately lead to a maximization in this lower bound, which in turn will maximize the initial objective, the log-likelihood.

For the deep learning model to learn, a loss function that will be minimized during training needs to be defined. The maximization problem can easily be transformed into a

minimization problem since maximizing the log-likelihood is equivalent to minimizing the negative log-likelihood[3]. This also changes the signs since the whole inequation is multiplied by -1:

$$-log(p_\theta(x_0)) \leq -log(p_\theta(x_0)) + KL(q(x_{1:T}|x_0) \parallel p_\theta(x_{1:T}|x_0)) \tag{2.16}$$

Rewriting $KL(q(x_{1:T}|x_0) \parallel p_\theta(x_{1:T}|x_0))$ to $log(\frac{q(x_{1:T}|x_0)}{p_\theta(x_{1:T}|x_0)})$ and some further reformulations (see Equation A.3) using bayes-rule results in Equation 2.17 leads to the following inequation [HJA20]:

$$-log(p_\theta(x_0)) \leq log(\frac{q(x_{1:T})|x_0)}{p_\theta(x_{0:T})}) \tag{2.17}$$

Equation 2.17 is the variational lower bound that is minimized during training to optimize $-log(p_\theta(x_0))$. With additional reformulations (see Equation A.4), the loss function can be derived as [HJA20]:

$$\mathbb{E}\left[\underbrace{D_{KL}(q(x_T|x_0) \parallel p(x_T))}_{L_T} + \sum_{t>1} \underbrace{D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t))}_{L_{t-1}} \underbrace{-logp_\theta(x_0|x_1)}_{L_0}\right] \tag{2.18}$$

L_T consists of $q(x_T|x_0)$ and is a noising forward process that has no learnable parameters and $p(x_T)$ is Gaussian distributed noise. If the forward process correctly destroys the data, $q(x_T|x_0)$ will converge to a standard Gaussian distribution as well and one can be confident that L_T will be close to 0. Thus, L_T will be neglected.

To L_0 can also be referred to as a "reconstruction term" [Luo22, p. 10]. Ho, Jain, and Abbeel [HJA20] initially use an independent discrete decoder to determine L_0, derived from $\mathcal{N}(x_0; \mu_0(x_1, 1)\sigma_1^2, \mathbf{I})$. However, ultimately the authors decide to remove the term in favor of a simpler loss term, which leads to a simpler implementation and better quality [HJA20].

Inside L_{t-1}, one compares the target denoising step $q(x_{t-1}|x_t, x_0)$ that is also conditioned on the input x_0 with the approximated denoising step $p_\theta(x_{t-1}|x_t)$. Both terms have a closed-form solution[4] thanks to the conditioning on x_0 [HJA20]:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \beta\mathbf{I}) \tag{2.19}$$

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu_t}(x_t, x_0), \tilde{\beta_t}\mathbf{I}) \tag{2.20}$$

The $\tilde{\mu_t}$ and $\tilde{\beta_t}$ of Equation 2.20 are defined as:

$$\tilde{\mu_t}(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\alpha_{\bar{t}}}x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t \quad \text{and } \tilde{\beta_t} = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{2.21}$$

[3]Also transforming the lower-bound from the maximization to an upper-bound for the minimization
[4]Meaning they can be computed within a finite amount of standard computations [BC13].

To further simplify $\tilde{\mu}_t$, the formula to create a noised input x_t in a single step from the unnoised input x_0 is used. x_t can be written in a closed-form solution as:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon \tag{2.22}$$

which in turn can be rewritten in terms of x_0 as:

$$x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}} (x_t - \sqrt{1 - \bar{\alpha}_t} \epsilon) \tag{2.23}$$

Given that β was fixed by Ho, Jain, and Abbeel [HJA20], x_0 as in Equation 2.23, the mean $\tilde{\mu}_t$ (Equation 2.21) can be further simplified to:

$$\tilde{\mu}_t(x_t, x_0) = \frac{1}{\sqrt{\bar{\alpha}_t}} (x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon) \tag{2.24}$$

which is just scaled random noise that is subtracted from x_t, the fully noised data [HJA20].

Hence, the KL-divergence in L_{t-1} in Equation 2.18 ultimately measures the difference between the actual noise that was added ($\tilde{\mu}_t$ in Equation 2.24) and the predicted noise from the neural network (μ_θ in Equation 2.19). Ho, Jain, and Abbeel [HJA20] proposes the Mean-Squared Error (MSE) between $\tilde{\mu}_t$ and μ_θ as a loss function:

$$L_{t-1} = \mathbb{E}_q \left[\frac{1}{2\sigma_t^2} \left\| \tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t) \right\|^2 \right] + C \tag{2.25}$$

The constant C will be removed during training, as it does not depend on θ [HJA20]. By applying the same reparameterization that was done in VAEs (see Equation 2.1) to μ_θ, it can be reformulated in the following way:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} (x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t)) \tag{2.26}$$

With the reparameterization of Equation 2.26, the parametrization of what the model should predict in the reverse process (from x_t to x_{t-1}) is moved. Instead of predicting the mean and variance of the reverse distribution ($p_\theta(x_{t-1}|x_t)$), i.e. directly predicting the less noised data, the model predicts the noise ϵ that makes the difference between x_{t-1} and x_t. In other words, instead of predicting the new data directly, the model learns to predict what noise it needs to remove from the data [Cap22]. Ultimately, during sampling x_{t-1} can be calculated by:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha} t}} \epsilon_\theta(x_t, t) \right) + \sigma_t z, \quad \text{with } \beta_t := 1 - \alpha_t \tag{2.27}$$

$z \sim \mathcal{N}(0, \mathbf{I})$ is a random component that is added during the reverse process (except for $t = 1$) to ensure the generation of diverse samples.

Substituting $\tilde{\mu_t}$ as in Equation 2.24 and μ_θ as in Equation 2.26 with some further simplification leads to:

$$\begin{aligned} L_{t-1} &= \mathbb{E}_{x_0,\epsilon}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}||\epsilon-\epsilon_\theta(x_t,t)||^2\right] \\ &= \mathbb{E}_{x_0,\epsilon}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}||\epsilon-\epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}x_0+\sqrt{1-\bar{\alpha}_t}\epsilon}_{x_t},t)||^2\right] \end{aligned} \tag{2.28}$$

Ultimately, Ho, Jain, and Abbeel [HJA20, p. 5] discovered that simplifying the loss term even further improves not only the sample quality but is also easier to implement. Equation 2.29 shows the final simplified loss function L_{simple}:

$$L_{simple}(\theta) = \mathbb{E}_{t,x_0,\epsilon}\left[||\epsilon-\epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0+\sqrt{1-\bar{\alpha}_t}\epsilon,t)||^2\right] \tag{2.29}$$

Ho, Jain, and Abbeel [HJA20] state that this simplified loss version is a weighted variational bound, which down-weights the loss for small t's, which corresponds to data point with only little added noise. They further argue that this is actually beneficial for the model since learning small noise is not difficult. Therefore the model can focus more on the large t's, where the denoising is more difficult [HJA20].

Algorithm 1 shows the full training algorithm, which will be realized by firstly randomly sampling a datapoint $x_0 \sim q(x_0)$ from the dataset, a timestep $t \sim Uniform(\{1,...,T\})$ and some random Gaussian noise $\epsilon \sim \mathcal{N}(0,\mathbf{I})$. Afterward, the gradient step will then be performed according to Equation 2.29 using gradient descent.

Sampling, on the other hand, follows Algorithm 2, starting with sampling Gaussian noise x_T. The Gaussian noise is then denoised in an iterative manner for T timesteps according to Equation 2.27 until a new, noise-free datapoint x_0 is generated.

In summary, [HJA20] introduced DDPM based on the work of [Soh+15]. Their diffusion model is based on a Markov chain and consists of the forward noising and the backward denoising part. During the forward process, a timestep t is sampled and the raw input data x_0 is noised accordingly using Equation 2.12. A function approximator (usually a neural network) with learnable parameters θ tries to remove the noise by predicting the noise that was added during the forward process. This difference between the actual noise ϵ and the predicted noise ϵ_θ is used as a loss function (Equation 2.29) to perform gradient descent. Once the training is finished, and the model converges, new synthetic data can be generated during sampling. In the sampling process, Gaussian noise can be sampled, and the model progressively denoises the sample for T timesteps until a new, denoised datapoint x_0, is generated.

Algorithm 1 Training [HJA20, p. 4]

repeat
  $x_0 \sim q(x_0)$
  $t \sim Uniform(\{1, ..., T\})$
  $\epsilon \sim \mathcal{N}(0, \mathbf{I})$
  Take gradient descent step on
    $\nabla_\theta ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, t)||^2$
until converged

Algorithm 2 Sampling algorithm [HJA20, p. 4]

$x_T \sim \mathcal{N}(0, \mathbf{I})$
for $t = T, \ldots, 1$ **do**
  $z \sim \mathcal{N}(0, \mathbf{I})$ if $t > 0$ else $z = 0$
  $x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}t}} \epsilon_\theta(x_t, t) \right) + \sigma_t z$
end for
return x_0

2.4.5 Multinomial Diffusion

The DDPM apprach uses a Gaussian diffusion process, designed to operate in continuous space [Kot+22]. However, many different data types are of categorical nature, where the Gaussian noising process proposed in Subsection 2.4.4 is not applicable. As a response, Hoogeboom et al. [Hoo+21] introduced multinomial diffusion, "a diffusion model directly defined on categorical variables" [Hoo+21, p, 3].

[Hoo+21] defines the multinomial diffusion process in the following way:

$$q(x_t|x_{t-1}) = \mathcal{C}(x_t|(1 - \beta_t)x_{t-1} + \beta_t/K) \tag{2.30}$$

The term β_t denotes the likelihood of swapping the current categorical value with a different one. Essentially, it is deciding how much random variation, or "noise", is introduced. This noise is based on a categorical uniform distribution, which means that it considers all available categories equally when picking a random value [Hoo+21]. Furthermore, $\mathcal{C}$ denotes a categorical distribution and x_t is represented in a One-Hot encoded categorical vector format with K different classes, $x_t \in \{0, 1\}^K$ [Hoo+21].

The noising step from x_0 to any noised version x_t can be expressed as:

$$q(x_t|x_0) = \mathcal{C}(x_t|\overline{\alpha}_t x_0 + (1 - \overline{\alpha}_t)/K) \tag{2.31}$$

with $\alpha_t = 1 - \beta_t$ and $\overline{\alpha}_t = \prod_{\tau=1}^{t} \alpha_\tau$. Uniform noise is added over the K classes in each timestep. If $(1 - \beta_t)$ is large, the model is likely to retain the previous category from x_{t-1}. Likewise, if $(1 - \beta_t)$ is small, the model will rely more on sampling a random category

from a uniform distribution over all categorical values for the variable.

Hoogeboom et al. [Hoo+21] derives from Equation 2.30 and Equation 2.31 the categorical posterior:

$$\begin{aligned} q(x_{t-1}|x_t, x_0) = \mathcal{C}(x_{t-1}|\theta_{post}(x_t, x_0)), \text{ where } \theta_{post}(x_t, x_0) = \tilde{\theta} / \sum_{k=1}^{K} \tilde{\theta}_k \\ \text{and } \tilde{\theta} = [\alpha_t x_t + (1-\alpha_t)/K] \odot [\overline{\alpha}_{t-1} x_0 + (1-\overline{\alpha}_{t-1})/K] \end{aligned} \tag{2.32}$$

They further argue, that predicting the added noise, as in [HJA20], is difficult, and instead the authors predict a probability vector for $\hat{x_0}$ from x_t [Hoo+21]. The reverse distribution $p_\theta(x_{t-1}|x_t)$ is realized in the form of $q(x_{t-1}|x_t, \hat{x_0})$ where $\hat{x_0} = \mu(x_t, t)$ is predicted by the model and tries to approximate x_0 [Hoo+21]. Hence:

$$p(x_0|x_1) = \mathcal{C}(x_0|\hat{x}_0) \text{ and } p(x_{t-1}|x_t) = \mathcal{C}(x_{t-1}|\theta_{post}(x_t, \hat{x}_0)) \text{ where } \hat{x}_0 = \mu(x_t, t) \tag{2.33}$$

In other words, determining the categorical value assigned to x_{t-1} (or x_0 in the last step) is based on the predicted probabilities $\hat{x}_0$ produced by the model.

The loss function in terms of KL-divergence (Equation 2.18) can be computed with Equation 2.32 and Equation 2.33:

$$\begin{aligned} L_{t-1} = \mathrm{KL}\Big(q(x_{t-1}|x_t, x_0)\big|\big|p(x_{t-1}|x_t)\Big) = \mathrm{KL}\Big(\mathcal{C}(\theta_{post}(x_t, x_0))\big|\big|\mathcal{C}(\theta_{post}(x_t, \hat{x}_0))\Big) \text{ and} \\ L_0 = -log p(x_0|x_1) = \sum_k x_{0,k} log \hat{x}_{0,k} \end{aligned} \tag{2.34}$$

Like in [HJA20], L_t has no learnable parameters and will be omitted.

2.5 Evaluation of Synthetic Tabular Data

A proper evaluation of the generated synthetic dataset is required to measure the effectiveness of the synthetic tabular data generation model. This evaluation is no trivial task due to the complexity and challenges of tabular data discussed in Section 2.1. Other data modalities, like images, are usually much easier to evaluate, for example, through human inspection [Chu+22]. This is not the case for tabular data, where various factors need to be considered during evaluation. Goncalves et al. [Gon+20, p. 6] identified two major dimensions that need to be considered during the evaluation, "data utility" and "information disclosure". The former dimensionality focuses on the quality of the synthetic data regarding the usefulness and statistical similarity. A synthetic dataset should have the same statistical and practical properties as its real counterpart from which it was generated, so it is helpful in practice and could be used as a substitute. The latter focuses on the amount of information carried over from the original dataset since the synthetic dataset

should not just be a copy of the real dataset. Hence, it is realized by privacy measures that capture how much information of the real dataset is disclosed in the synthetic dataset [Gon+20]. Little et al. [Lit+21, p. 2] highlights that these two evaluation aspects, "high data utility" and "low disclosure risk", are two "competing interests" that need to be balanced. This section will present different ways to evaluate synthetic tabular data, each focusing on a different evaluation aspect. [El 20] identifies seven ways to evaluate synthetic data's utility aspect. However, in the literature and in practice, only a few approaches are used and will be presented in greater detail. Firstly, statistical similarity (data utility) will be presented, followed by machine learning efficacy (data utility) and privacy evaluation methods (information disclosure). Lastly, additional evaluation methods are shortly listed, and a proposed "similarity score" as a unified metric is presented.

2.5.1 Statistical Evaluation

Various different metrics can be used to measure the statistical similarity between a synthetic dataset and the corresponding real dataset. As a start, synthetic data should have similar basic statistical measures. For example, [dCar+17] compares different statistical properties of the original and synthetic data. The authors include computing the mean, standard deviation, proportion of unique values, and the number of distinct values for each attribute.

[Gon+20] proposes multiple more complex statistical measures that go beyond the comparison of single values, like the mean or standard deviation. The KL-divergence or its extension the Jensen-Shannon distance [Zha+22] can be calculated between a pair of probability distributions from the real and synthetic dataset [Gon+20; Li+22]. While this measures the similarity for one variable, or column, of the data only, correlations between multiple pairs of variables can also be measured using the Pairwise Correlation Difference (PCD) [Gon+20]. This metric evaluates the extent to which the synthetic dataset maintains the same inter-column relationships as those present in the original dataset. Another measurement is the support-coverage score [Gon+20]. The support-coverage metric deals with the mathematical support of the variables. More specifically, it assesses the similarity or identity between the range of potential values in the synthetic dataset and those in the original dataset.

2.5.2 Machine Learning Efficacy

The machine learning efficacy (also referred to as "machine learning utility" [Zha+21a] or "cross classification" [Gon+20]) is an approach that measures the utility of synthetic data. It has been used in a variety of works [Zha+22; Xu+19; Zha+21a; Bou+21; Ge+21]. This metric investigates whether "it is possible to replace the real with generated data to solve problems using Machine Leaning models" [Bou+21, p. 7]. A set of classification models (for example decision trees, regression models, neural networks, and more) are evaluated in a classification or regression task.

Goncalves et al. [Gon+20, p. 7] highlights two distinct ways to perform the machine learning efficacy:

1. "CrCl-RS": train a model on real data and test it on hold-out data (test set) from both real and synthetic datasets.

2. "CrCl-SR": train a model on synthetic data and test it on hold-out data from both real and synthetic datasets.

The authors argue, that the former is useful to evaluate if statistical properties are similar between real and synthetic datasets and that the latter allows confirming whether "scientific conclusions drawn from (...) models trained on synthetic datasets can safely be applied to real datasets" [Gon+20, p. 7]. The performance of the classification models itself is based on commonly used metrics for classification/regression tasks, such as the f1 score or Root Mean Squared Error (RMSE) score [Bou+21; Chu+22]. It is important to mention that there is no universally accepted way to perform this machine learning efficacy, yet, as it is implemented differently throughout the literature. For example, [Zha+22] makes use of five models and calculates five different metrics, [Ge+21] on the other hand, uses nine different models but calculates only three different metric scores. Moreover, [Kun+21] uses four models with four metrics, and [Kim+21] uses three models on only one metric. In addition to the various models and metric scores used, each model can have different hyperparameters. For example, [Kun+21] uses the default hyperparameters[5] for all models and [Kim+21] chooses the hyperparameters that show the best performance in a cross-validation hyperparameter tuning. Lastly, the different synthetic data generation models might use different datasets to compute their machine learning efficacy scores. Even though most of the generation approaches have adopted the *CrCl-SR* [Gon+20] method to calculate the machine learning efficacy score, the aforementioned variabilities in terms of metrics- and model-choice make it hard to compare the performance across the different generative models using the machine learning efficacy.

2.5.3 Privacy Evaluation

Goncalves et al. [Gon+20, p. 7f.] discussed two types of privacy aspects that should be evaluated, "membership disclosure" and "attribute disclosure". While the former is concerned with whether an individual from the original dataset can be identified in the synthetic dataset, the latter is concerned with the possibility of inferring actual values for an attribute of the original dataset, given the synthetic dataset. To measure the *membership disclosure*, [Gon+20] proposes calculating the distance of synthetic datapoints to real datapoints. If this distance is smaller than a specific value, the synthetic datapoint is considered to be present in the original dataset. The distance measure used by [Gon+20] was the *Hamming-distance*, but this could be replaced with a similarity measure of choice,

[5] From the scikit library [Ped+11].

depending on the data. *Attribute disclosure* could be measured by testing if a potential attacker could infer unknown values of attributes of an individual, given a subset of known attributes and their values by using a k-nearest neighbors algorithm [Gon+20]. These privacy measures are especially prominent and essential in the context of medical data, which usually contain sensitive information [Gon+20; Cho+17].

To ensure privacy is granted, research towards including Differential Privacy (DP) [Dwo08] in the model has been made, e.g. in [Xie+18; JYS18; TFR22]. DP is a concept in the field of data privacy that aims to protect the privacy of individuals while still allowing useful information to be extracted from data [Dwo08]. It provides a mathematical framework for measuring the privacy risk associated with releasing data and designing algorithms that minimize this risk [Dwo08; Zha+22]. DP can be achieved by injecting noise to the gradients during the learning process [Xie+18].

2.5.4 Additional Evaluation Methods

There are several other ways to evaluate synthetic data that do not fall directly into the categories above but are worth mentioning:

Expert Evaluation

El Emam [El 20] mentioned that the quality of synthetic data can be determined by a domain expert, who is familiar with how data in the given domain should look. The expert could be asked to differentiate a set of synthetic data samples from a set of real data samples, and the accuracy could be measured. However, this metric is not only highly dependent on the individual expert it is also very time-consuming and hardly viable in practice. A possible way to automate this approach is to use a classification model. El Emam [El 20] proposes to train such a classification model to distinguish between real and synthetic data. If such a model is not able to reliably distinguish between real and synthetic data, the synthetic data resemblance to the real data is high El Emam [El 20, p. 57]. This technique was employed in the work of Alzantot, Chakraborty, and Srivastava [ACS17], who deployed an LSTM-based model as a discriminator, whose task was to discriminate between real and synthetic time-series records.

Comparative Assessment

Another measure proposed by [El 20] is to compute aggregations and statistics of the synthetic data and compare it to publicly available known statistics and aggregations that are about the same topic. If the synthetic is good, it would "tell the same story as the public data" [El 20, p. 58]. To determine the stability of the generative model, El Emam [El 20] proposes that the model could be used to generate not a single synthetic dataset but multiple. Evaluation (e.g. using a statistical measure such as in Subsection 2.5.1) is then performed on all synthetic datasets, and averages are calculated. Comparing the

individual synthetic datasets statistical measures with the average over multiple datasets could highlight if certain synthetic datasets are biased in any form. Measures like the variance could also highlight if the generation process is overall stable (small variance), meaning the generation model tends to produce synthetic datasets with similar statistical properties, or unstable (high variance), meaning the generation model tends to produce synthetic datasets with different statistical properties. Hence, this synthesis of multiple datasets should highlight if the generative model has learned any form of bias or instability, which is not desirable [El 20].

Resemblance Techniques

Lastly, [Her+22] proposes a set of resemblance techniques to measure the similarity of the synthetic data to the real data. Most of [Her+22] proposed methods have already been mentioned and fall into one of the categories above, such as "consultation with [...] experts" or computing "correlation coefficients" [Her+22]. Another viable way to quickly get an indication of how well the similarity is is through visualization techniques. Hernandez et al. [Her+22] proposes dimensionality reduction techniques like PCA, histograms of attributes, and correlation matrices can be computed and visualized. With this visualization, a human can quickly determine, if the visualizations produced by the synthetic data resemble those of the real data. For example, this type of visual evaluation was performed in the works Li et al. [Li+22], Le Minh and Le [LL21], and McKeever, Walia, and Tierney [MWT20].

2.5.5 Similarity Score

Given the previous sections, one can see that various evaluation techniques exist. This variety is problematic for comparing the approaches to generate synthetic data, as there is no universally accepted metric or benchmark to compare the techniques [Her+22; Chu+22]. Brenninkmeijer, Hille, and Vries [BHV19] proposes a similarity score to address this issue by providing a single score that aggregates different metrics used throughout the literature. It consists of the average of five different similarity metrics that cover different aspects of the data. These are:

- S_{basic}: compares the mean and standard deviation of the real and synthetic dataset using $Spearman's\ \rho$ [Lan03] correlation per column.
- S_{corr}: calculates correlations/associations between the columns within the real and synthetic dataset using a correlation/association matrix. Different statistical correlation measures are used to measure the correlation between columns of different data types [BHV19, p. 37]. Pearson's correlation for two continuous columns, correlation ratio between continuous and categorical, and Theil's U for two categorical [BHV19; Lan03]. Ultimately, the matrices are compared using Pearson correlation $\rho_{pearson}$.

- S_{mirr}: the mirror column association score is similar to S_{corr}, but instead of comparing the correlations occurring between columns within the dataset, the metric compares whether the same columns in the real, and synthetic dataset correlate. This is done for each column and the average score is calculated.
- S_{pca}: measures whether the dimensionality reduction of the real and synthetic dataset through PCA is similar. Five principal components are computed for each dataset, and the average error in percentage between both PCAs is calculated using the mean absolute percentage error.
- S_{est}: is a machine learning efficacy similarity score. Four different models are chosen, depending on whether the dataset is a classification or regression dataset, and performance metric averages are computed. The root mean squared error and F1 score for regression or classification tasks respectively is chosen as performance metric.

Chundawat et al. [Chu+22] propose TabSynDex, a single similarity score based upon combining multiple scores covering different aspects of data similarity like in the works of [BHV19]. In their work, they point out weaknesses of the similarity score proposed by Brenninkmeijer, Hille, and Vries [BHV19]. For example, they criticized the overreliance on the correlation coefficient as a primary similarity metric, which is not affected by the scale of the variables that are compared. Chundawat et al. [Chu+22] argue that this is problematic since even when the statistics of real and synthetic data closely align but fail to follow the same pattern, they do not correlate. Consequently, the perceived similarity between the datasets is underestimated. The authors further argue that due to the reliance on correlation, even when comparing two segments of the same dataset, their measured similarity may seem low, which is contradictory since they originate from the same distribution. As a consequence, they propose the following similarity metrics:

Basic statistical measure: Computes basic statistics of the data for each column, namely mean, median and standard deviation, and calculates the relative error between the real and fake columns:

$$e = \frac{1}{N}\left|\frac{R_i - F_i}{R_i}\right| \tag{2.35}$$

with R_i and F_i denoting the statistical measure of column i for the real and synthetic data, respectively. The score for each metric is calculated as follows:

$$s_m = 1 - \frac{e_1^m + e_2^m + ... + e_n^m}{n}, \text{ with } m \in \{mean, median, std\}$$

where n is the number of columns. The final score will be calculated as

$$S_{basic} = \frac{s_{mean} + s_{median} + s_{std\_dev}}{3} \tag{2.36}$$

Log-transformer Correlation: Compares the correlations between columns within the datasets to compute two matrices with n^2 values for n columns. Correlations are calculated depending on the data type, like in [BHV19]. The matrix entries are also log-transformed to account for the issue of very small values in the correlation matrix of real data, which can result in higher relative error values when compared to the correlation matrix for generated data, even if both correlations are nearly zero. The log transformation helps to rescale the values and provide a more accurate comparison between the correlation matrices of real and generated data. Equation 2.37 defines the correlation score, where $sln(x) = sign(x)ln(|x|)$ and $r_{i,j}, s_{i,j}$ indicating the matrix entries of the real and synthetic data. $sign(x)$ is defined as

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \end{cases}$$

which results in the final score:

$$S_{corrl} = 1 - \frac{1}{n^2 - n} \sum_{i=1}^{n} \sum_{j=1_{j \neq i}}^{n} \left| \frac{sln(r_{i,j}) - sln(s_{i,j})}{sln(r_{i,j})} \right| \tag{2.37}$$

Propensity Mean Squared Error: Is used to "robustly discriminate between real and synthetic data" [Chu+22, p. 3]. It is based upon the work of Snoke et al. [Sno+18], who propose the use of the pMSE in the context of synthetic data generation. The pMSE is computed by combining a set of samples from the real and synthetic dataset and adding a label Y to each entry, indicating whether the given sample stems from the real or the synthetic dataset. A logistic regression model is trained to predict probabilities $\hat{p}_i$ for each sample i for it being a synthetic data sample. With $N = N_{real} + N_{syn}$, the pMSE is calculated as:

$$pMSE = 1 - \frac{1}{N} \sum_{i=1}^{N} (\hat{p}_i - c)^2$$

with $c = N_{syn}/(N_{real} + N_{syn})$ indicating the proportion of synthetic samples.

For a similarity score, Snoke et al. [Sno+18] ultimately proposes the pMSE-ratio as a measure. For this, the so-called null distribution of pMSE is required [Sno+18, p. 668]. This null distribution is essentially derived under the assumption of perfect synthesis, where the synthetic data are generated from the same model that produced the original data. This creates a baseline for comparing the quality of synthetic data, serving as a reference point to evaluate the effectiveness of different synthetic data generation methods [Sno+18]. Essentially, it computes the expected value of pMSE when the real and synthetic data are indistinguishable.

It is calculated as [Sno+18; Chu+22]:

$$\mathbb{E}(pMSE_0) = \frac{(k-1)(1-c)^2 c}{N}$$

with the logistic regression model having k parameters (including bias) [Chu+22]. [Chu+22] improves the original metric as introduced in [Sno+18] further through the introduction of a value α that can be chosen task-specific and helps standardization of the score to $[0, 1]$ (the authors' default is set to $\alpha = 1.2$) The final score is calculated as

$$S_{pMSE} = \alpha^{-|1-\frac{pMSE}{\mathbb{E}(pMSE_0)}|} \tag{2.38}$$

that calculates the ration of pMSE and $pMSE_0$. Hence, the closer the ratio is to 1, the more indistinguishable the synthetic data is from the real data [Chu+22].

Regularized Support Coverage: Measures how well a synthetic dataset covers the variables in a real dataset. For a single variable, the score is calculated as the average of the number of samples in each category in the fake data divided by the number of samples in the corresponding category in the real data, multiplied by a scaling factor. It penalizes the score more if rare categories are not well-represented.

Mathematically, the score for a single variable or column c is calculated as follows:

$$s_{cr} = \frac{1}{n_{cat}} \sum_{i=1}^{n_{cat}} \frac{n_i^s}{n_i^r} \cdot scaling\ factor$$

Where n_{cat} is the number of categories for a variable, n_i^s is the number of samples in that category in the synthetic data, n_i^r is the corresponding number of samples in the real data, and the scaling factor is N_{real}/N_{syn} [Chu+22]. The authors limit the S_{cr} score to a predefined β value (the authors' default is set to $\beta = 2$), to account for the potential for s_{cr} to become very sensitive to n_i^s, if n_i^r is very small [Chu+22].

The final S_{cr} score is calculated as the average of the score for each column [Chu+22]:

$$S_{cr} = \frac{1}{n} \sum_{i=1}^{n} s_{cr_i} \tag{2.39}$$

Machine Learning Efficacy: Is a metric to indicate how valuable the synthetic data is in a machine learning scenario (see Subsection 2.5.2). [Chu+22] adopts the proposed metrics and models by [BHV19] that differentiates the chosen set of models and metric based on if the dataset contains a classification or regression target. The models include *Random Forest-*, *Lasso-*, *Ridge-* and *ElasticNet-Regression* for regression tasks and *Logistic Regression-*, *Random Forest-*, *Decision Tree-* and an *MLP-classifier* for classification tasks [BHV19; Chu+22]. For each model, two versions are trained, one trained on synthetic and one trained on real training data. Their performance is tested on both the real and synthetic targets using the F1-score or RMSE-score for

classification and regression tasks respectively. For each model, the relative error (Equation 2.35) is calculated between the metric results of the model that was trained on real data and the model that was trained on synthetic data. The final score is the average performance error for all four models:

$$S_{ml} = 1 - \frac{e_{m1} + e_{m2} + e_{m3} + e_{m4}}{4} \tag{2.40}$$

where e_{mi} is the error for the i^{th} model [Chu+22].

The final unified metric, named TabSynDex, is the average of all scores mentioned above and is described in Equation 2.41 [Chu+22]:

$$TabSynDex = \frac{S_{basic} + S_{corrl} + S_{pMSE} + S_{cr} + S_{ml}}{5} \tag{2.41}$$

3 Related Work

In recent years, substantial research has been conducted in the realm of tabular data modeling. This chapter aims to comprehensively review the most significant methodologies employed in tabular data synthesis. The initial section emphasizes advancements in GANs, highlighting the progress that has resulted in state-of-the-art performance. Subsequently, influential alternative strategies for tabular data synthesis that do not depend on GANs are explored.

Following this, the discussion turns to diffusion models, a topic that has garnered significant attention in the image generation domain. The most prominent works and their respective enhancements in this field are examined in detail. Finally, the chapter delves into exploring diffusion models and their utilization in the context of tabular data synthesis.

3.1 Generative Adversarial Networks Models

The GAN-architecture is a common way to address the problem of tabular data synthesis in the literature [Bor+22]. Over the last years, several improvements have been made to account for their shortcomings, such as the introduction of the Wasserstein [Fro+15] distance [ACB17], conditioning [MO14], or gradient penalty [Gul+17]. GANs have already shown stunning results across several data modalities [MWT20], and their success extends into the domain of tabular data generation.

The authors of medGAN [Cho+17] showed how GANs can be used in the medical domain to generate synthetic patient records. Their medGAN is able to work with discrete data and includes an autoencoder in their architecture. In the medGAN architecture, the autoencoder translates the generator's continuous outputs into discrete synthetic patient records.

TableGAN proposed by [Par+18] takes a different approach towards preprocessing the tabular data compared to other research. They use a simplistic min-max scaling for continuous columns and label-encode categorical values. The authors' most significant change is converting the tabular 1-dimensional data into a 2-dimensional matrix form. The authors give an example how such a conversion could look like: A record of 24 values will be converted into a 5x5 square matrix, where zeros fill the remaining entries [Par+18, p-4]. This allows them to use CNNs inside their model.

There are also models that specifically focus on privacy, such as PATE-GAN [JYS18]. PATE refers to "Private Aggregation of Teacher Ensembles" [Pap+17] and is a framework

that provides strong privacy guarantees for sensitive training data [JYS18]. PATE-GAN integrates this framework into the GAN architecture and demonstrated that it is able to produce not only high-quality synthetic data but also able to hold strict privacy guarantees.

The TGAN [XV18] model is able to generate discrete and continuous data simultaneously. TGAN deals with continuous variables through a mode-specific normalization with Gaussian Mixture Models (GMMs) [XV18, p. 3]. Their architecture includes LSTM cells with attention that generate the data column-wise instead of record-wise, as done in other works [XV18].

The authors of TGAN follow up on their own work and introduce a new architecture named CTGAN [Xu+19]. They first identified the biggest challenges in tabular data generation, namely "mixed data types", "Non-Gaussian distributions", "Multinomial distributions", "learning form sparse One-Hot encoded vectors" and "highly imbalanced categorical columns" [Xu+19, p. 3]. To address some of these issues, they improve their preprocessing and their architecture. Firstly, instead of using a Gaussian Mixture Model (GMM), they use a Variational Gaussian Mixture model (VGM) [Xu+19]. Secondly, they introduce the possibility of conditioning the GAN to produce certain column values by employing a conditional vector during training [Xu+19]. Lastly, they changed their model architecture by removing the LSTM cells for better computation time, added batch normalization, and adopted a GAN architecture that uses Wasserstein-distance and gradient penalty [Gul+17].

Additional research towards improving the CTGAN has been made by Zhao, Kunar, Chen, and Birke that introduced CTAB-GAN [Zha+21b] and their successor CTAB-GAN+ [Zha+22]. The authors criticize that other works do not account for mixed data types [Zha+22]. As a result, they introduce a mixed-data type encoding (see Subsection 5.1.2) that allows encoding a column that contains both categorical and numerical data. This is achieved in conjunction with the mode-specific normalization, which was also used by Xu and Veeramachaneni [XV18] and Xu et al. [Xu+19]. Furthermore, they specifically address the problem of imbalanced training datasets and the treatment of long tails in data (compare Subsection 2.2.2). To counter imbalanced training datasets, the authors perform a training-by-sampling, meaning they adjust the probability of each data class being selected as a training sample using a logarithmic function. This gives rare classes a higher chance of being selected during the training process, thereby rebalancing the data distribution and helping the model better understand underrepresented categories. In order to treat long tails, they propose a preprocessing technique that uses logarithmic transformation for variables with long-tailed distributions. This helps handle outliers and extreme data points in these distributions effectively. Additionally, CTAB-GAN introduces an auxiliary (pre-trained) classifier to provide "additional supervision to improve its [CTAB-GANs] utility for ML applications" [Zha+21b, p. 2]. The auxiliary classifier is used to enhance the realism and semantic integrity of the synthetic data generated by the GAN model. It does this by taking the synthesized record and trying to predict the target

class of the synthetic record. The predicted target class and the synthesized target class created by the generator are then compared. The discrepancy between the prediction of the classifier and the synthesized target class is transformed into a classification loss and also used to update the weights of the generator. The authors' newest version of CTAB-GAN, called CTAB-GAN+ [Zha+22], achieves state-of-the-art performance on synthetic data generation. Changes compared to the first CTAB-GAN version include new feature encoding, adopting Wasserstein distance and gradient penalty, the auxiliary classifier can be exchanged for an auxiliary regression model, and using differential privacy stochastic gradient decent [Aba+16] for discriminator training (adopted from [JYS18]) [Zha+22].

In summary, it is clear that many different GAN approaches exist for tabular data synthesis, each focusing on improving certain aspects that make dealing with tabular data challenging. Since this section is meant to give an overview of the topic, it is necessary to mention that there exist numerous other noteworthy studies that merit further exploration. However, they are not included in this thesis due to limitations in length. Nevertheless, these works provide valuable insights and contributions to the field under investigation, and their exclusion should not be interpreted as a lack of significance or relevance. In particular, the works of Fan et al. [Fan+20], Hernandez et al. [Her+22], and Bourou et al. [Bou+21] provide a valuable overview of the research area.

3.2 Other Models

Several other approaches towards generating synthetic tabular data do not rely on GANs. Famous classical approaches include Synthetic Minority Oversampling Technique (SMOTE) [Cha+02], an over-sampling technique to tackle imbalanced datasets. SMOTE works by sampling new synthetic data entries from the minority class. However, entries are not just duplicated. A minority class instance and its 'k' closest minority class neighbors are randomly selected. A synthetic instance is then created at a randomly selected point along the line connecting the chosen instance and one of its randomly selected neighbors in the feature space. Other classical approaches include Bayesian methods such as [Zha+17] or models that rely on a hidden Markov model as in [DC19]. Another class of models takes a probabilistic approach, such as flow-based models [KAD21] or VAE based approaches [KW13]. In [Xu+19], the authors do not only introduce CTGAN, they also present TVAE, a tabular variational autoencoder. Recently, transformer models, which have been heavily used in the NLP domain, have also been applied to tabular data. The authors of [Hua+20] introduce TabTransformer, which learns contextual embeddings and can be applied to tabular data for (semi-) supervised learning scenarios. In terms of synthetic data generation, Padhi et al. [Pad+21] are able to generate synthetic tabular time-series data using their TabGPT model and a tokenization of the tabular entries. Another approach for synthetic data generation was proposed by Ge et al. [Ge+21] with their Kamino model. Kamino ensures the preservation of the data's original structure and

correlations while also maintaining differential privacy. Kamino uses the input database's schema to create a synthetic dataset that maintains the usefulness of the original data for further tasks such as model training and answering questions, while also ensuring privacy [Ge+21].

3.3 Diffusion Models

3.3.1 Diffusion Probabilistic Models

Since the release of DDPM [HJA20], several researchers have worked on diffusion models and were able to improve the model's performance in terms of quality and efficiency. In the work of Nichol and Dhariwal [ND21], the authors build upon the work of Ho, Jain, and Abbeel [HJA20] and propose changes to improve the overall model's generative capability. Initially, the variance in diffusion models was fixed according to the noise schedule with β_t and $\tilde{\beta}_t$ as the upper and lower bound on the variance, the two opposite extremes. Rather than keeping the variance constant, Nichol and Dhariwal [ND21] developed a method where the model learns the variance. This is done by using v as the model's output vector, which is then converted into different variances.

$$\Sigma_\theta(x_t, t) = exp(v log\beta_t + (1 - v) log\hat{\beta}_t) \tag{3.1}$$

Equation 3.1 defines the procedure for transforming the model's output vector, v, into a learned variance, illustrating an interpolation between the upper and lower bounds of the variance, β_t and $\hat{\beta}_t$ [ND21]. The loss L_{simple} does not depend on the variance, which results in [ND21] proposing a hybrid loss:

$$L_{hybrid} = L_{simple} + \lambda L_{VLB} \tag{3.2}$$

with scaling factor λ and variational lower bound loss L_{VLB}.

Secondly, the authors propose a different noise schedule for the noising forward process. The authors argue that the previous linear noising process destroys the information in the image too fast, especially towards the end [ND21]. As a consequence, they introduce a cosine schedule. Figure 3.1 shows latent samples from a linear noising schedule (top) and from the proposed cosine schedule (bottom) [ND21, Figure 3, p. 4], where one can see how the cosine schedule noises the image more slowly [ND21].

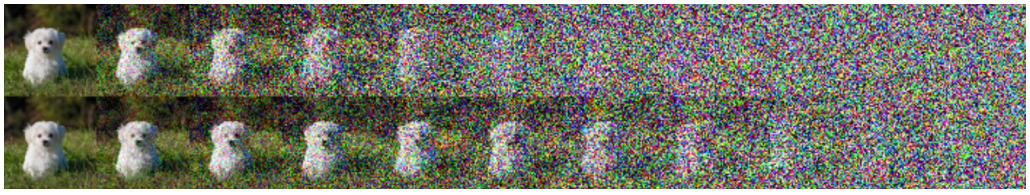


Figure 3.1: Latent samples from a linear noising schedule (top) and from the proposed cosine schedule (bottom) [ND21, Figure 3, p. 4]

A significant milestone for diffusion models was done by [DN21], where the authors were able to show that diffusion models are able to outperform GAN models on image synthesis, which have been considered state-of-the-art at that time [DN21]. The authors state:

> "We hypothesize that the gap between diffusion models and GANs stems from at least two factors: first, that the model architectures used by recent GAN literature have been heavily explored and refined; second, that GANs are able to trade off diversity for fidelity, producing high-quality samples but not covering the whole distribution. We aim to bring these benefits to diffusion models, first by improving model architecture and then by devising a scheme for trading off diversity for fidelity. With these improvements, we achieve a new state-of-the-art, surpassing GANs on several different metrics and datasets." [DN21, p. 2]

Thus, they argue that GAN models received more attention, and, therefore, much more improvements in their architecture as well as their training have been discovered, leading to their superior performance. Furthermore, similar improvements on diffusion models would show that they are able to outperform GAN models. These improvements include: First, several architectural improvements [DN21, section 3] and an introduction of a classifier guidance mechanism for conditioning [DN21, section 4]. Architectural changes include the adding of residual blocks, additional attention mechanisms at different layers, adaptive group normalization, and adding class embeddings into residual blocks [DN21]. To further improve the conditioning mechanism, i.e. controlling what class the model will generate, an additional classifier is introduced [DN21]. This classifier is trained on noisy images at different timesteps and is tasked to predict the class of the input image. During sampling, the gradients produced by this classifier are used by the diffusion model to guide it towards creating an image of that class[1] [DN21]. With these improvements, the authors outperformed GAN models on unconditional and class-conditional image generation. Additionally, their model can be controlled in terms of diversity or fidelity through scaling of the classifier guidance gradients.

Since classifier guidance, as presented above, requires training of an additional classifier and thus, increasing the time for computation, [HS22] introduced a mechanism to achieve a guidance mechanism without an additional classifier. Their guidance mechanism is based on the idea of jointly training a conditional diffusion and an unconditional diffusion model and interpolating between their gradients to control the amount of guidance for the output [HS22].

All the above models apply the diffusion process on the pixel space, resulting in large tensors that require more computation time and require large Graphics Processing Units

[1] The class of an image depends on the specific dataset. A typical example would be a dataset of animals, where the class of the image would be the name of the animal, e.g. a photo of a dog would be of class "dog".

(GPUs) for computation [Rom+22]. [Rom+22] addresses this issue by moving the diffusion process from the pixel to a latent space. In the first step, the encoder part of an autoencoder (see Subsection 2.4.1) is trained to learn a latent representation with smaller dimensionality of the input image [Rom+22]. Given the encoded latent representation, a decoder is tasked to reconstruct the original input. Figure 3.2 shows the architecture of the overall latent diffusion model [Rom+22, Figure 3, p.4]. The raw input x is encoded into a latent vector z through the pre-trained encoder $\mathcal{E}$ [Rom+22]. Inside this latent space, the diffusion with forward and reverse noising process is performed [Rom+22]. Additional conditioning information can be added by transforming the conditioning input into the latent space as well (τ_θ) [Rom+22]. The denoising model is changed compared to previous approaches through the addition of cross-attention blocks to further enhance the generation quality [Rom+22]. After the denoising is completed, the pre-trained decoder $\mathcal{D}$ moves the latent diffusion model output back into the pixel space, generating an image $\tilde{x}$ [Rom+22].

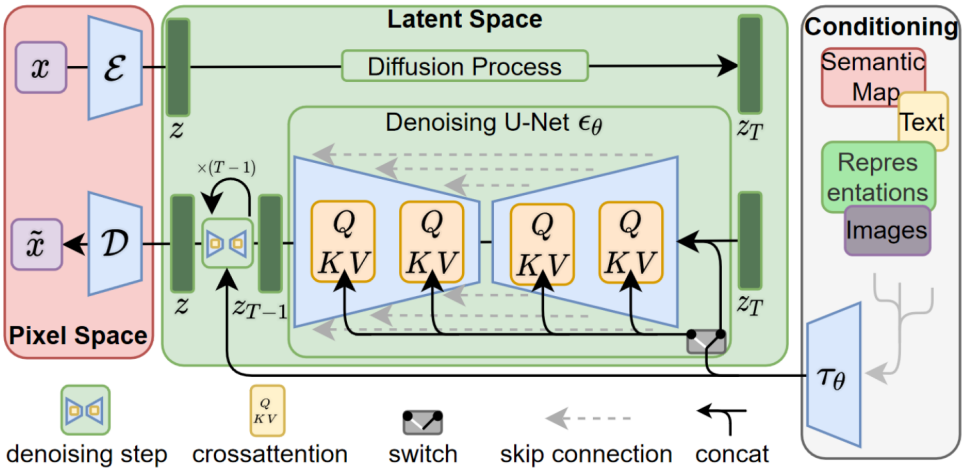


Figure 3.2: Latent diffusion model architecture [Rom+22, Figure 3, p.4]

Thanks to the authors of [Rom+22] and Huggingface [Fac23c], the latent diffusion model was made publicly available and a pipeline-framework was developed, allowing developers to build upon the existing work easily, test the models locally, and explore and implement new features [Fac23b]. This step has led to various community-build diffusion models, ranging from an image-to-image inpainting[2] diffusion model to a text-to-image diffusion model with multilingual support [Fac23a].

3.3.2 Diffusion Probabilistic Models for Tabular Data

At the time of writing, there has been limited work on applying diffusion models to tabular data. Nevertheless, this section will cover two research papers that apply diffusion to tabular data.

[2]Image-to-image inpainting is a process in computer vision where missing or corrupted parts of an image are filled in with plausible content, usually based on the surrounding context of the image.

Diffusion on tabular data imputation

Tashiro et al. [Tas+21] proposed a conditional score-based diffusion model (CSDI) for the time series imputation task, i.e. the task of estimating and filling in missing or absent data points within a time series dataset based on the existing data. Their CSDI model shows a significant performance increase in generating time-series data compared to other imputation techniques. [ZC22] showed how the CSDI model can be used in the context of missing value imputation in tabular data that is not a time series (CSDI_T). The authors address the problem of simultaneously handling numerical and categorical variables in their work. Their diffusion model does not reconstruct the complete original data point (e.g. an entire tabular data row). Instead, the diffusion model receives a splitted version of the input, one part that is observable ("conditional part") x^{co} and one part that is unobservable ("target part") x^{ta}. The goal of the model is, given the unnoised observed part and the noised version of the unobserved part, to return the denoised unobserved part of the next timestep, i.e. $p_\theta(x^{ta}_{t-1}|x^{ta}_t, x^{co}_0)$ [ZC22]. Categorical entries are converted in three different ways to handle them in conjunction with numerical values. Categorical values are either One-Hot encoded, analog-bits encoded (also called binary encoded), or embedded through a feature tokenization approach proposed by [Gor+21]. In the embedding encoding approach, numerical values are encoded as well. However, during training, neither the categorical nor the numerical embeddings are learned and remain static and fixed [ZC23] The authors observed that diffusion achieves competitive results across several datasets for the given evaluation metrics for the task of missing value imputation[3] [ZC22]. The different encoding techniques perform comparably well, with the embedding technique outperforming the other two techniques by a slight margin [ZC22].

Diffusion on tabular data synthesis

The task of synthesizing an entire tabular dataset through diffusion has been only explored by Kotelnikov et al. [Kot+22]. The authors introduce their diffusion model, called TabDDPM, for tabular data synthesis, outperforming the existing state-of-the-art approaches using alternative architectures like GANs or VAEs [Kot+22]. Since the TabDDPM approach will be the base for this thesis, it will be explained in detail in this section.

Like Zheng and Charoenphakdee [ZC22], Kotelnikov et al. [Kot+22] identifies the need for processing categorical variables in some form so that they can be processed by a diffusion model. However, TabDDPM takes a different approach than the CSDI model, using two different diffusion processes. The classical Gaussian diffusion process [HJA20] for numerical columns and multinomial diffusion [Hoo+21] (Subsection 2.4.5) for categorical/binary columns [ZC22]. Firstly, the feature columns are preprocessed. Numerical features are transformed using the Gaussian quantile transformation (Subsection 2.1.1),

[3] The authors used the RMSE as well as the error rate as evaluation metric [ZC23, p. 2].

and categorical features are One-Hot encoded (Subsection 2.1.1) [Kot+22]. The objective function is defined as:

$$L_t^{TabDDPM} = L_t^{simple} + \frac{\sum_{i \leq C} L_t^i}{C} \tag{3.3}$$

where L_t^{simple} is equivalent to Equation 2.29, the simplified loss function from Ho, Jain, and Abbeel [HJA20] and L_t^i being the KL-divergence for each multinomial diffusion term (L_{t-1} in Equation 2.18) divided by the number of categorical features C [Kot+22].

The neural network used to predict the noise in the reverse process is a simple MLP based upon the works of [Gor+21]. The architecture consists of several identical MLP-blocks comprising a Linear layer, followed by an activation layer (ReLU), and a final dropout layer. As in [ND21; DN21], the authors combine the input x_{in} with a sinusoidal temporal embedding[4] and a class label embedding. Firstly, the input is sent through a linear layer and combined with the temporal and class label embeddings. The output of that process is sent through a specified number of identical MLP-blocks specified above before passing a last linear layer [Kot+22].

TabDDPM is compared against several baseline models including TVAE [Xu+19], CTAB-GAN [Zha+21b], CTABGAN+ [Zha+22], and a classical interpolation-based approach SMOTE [Cha+02]. Fifteen datasets have been used for generating synthetic data, of which six have a regression task type, seven are binary classification tasks, and two are multiclass classification tasks [Kot+22]. Eight of the datasets have numerical and categorical features, and seven only consist of numerical features [Kot+22]. The overall number of features in the datasets varies heavily. For a complete list of all datasets, please refer to "Table 2" in [Kot+22, p. 5].

The primary evaluation metric chosen by the authors is the machine learning efficacy (see Subsection 2.5.2). Kotelnikov et al. [Kot+22] realizes this in three different ways, firstly, by using a set of machine and deep learning models[5] and averaging their performance, secondly, using a CatBoost model [Pro+18](a state-of-the-art model on tabular tasks) [Kot+22], and thirdly, an MLP architecture proposed by [Gor+21]. In the latter case, the models are previously tuned on the real dataset so that the best hyperparameters can be found for each dataset, which will be used during the evaluation [Kot+22]. To account for other random factors that could influence the results, the authors compute the machine learning efficacy metric scores by generating five different synthetic datasets and train the evaluation models for ten random initializations. The average over all these 50 variations is reported ([Kot+22, Table 3, 4, p. 8]) including the standard deviation across the metric results [Kot+22].

[4] An encoding of the current timestemp in the diffusion process, computed using the mathematical sine function.

[5] Decision Tree, Random Forest, Logistic Regression, MLP.

The authors summarize three major findings [Kot+22]:

- TabDDPM outperforms TVAE and CTABGAN+ on most datasets.
- The classical approachSMOTEshows surprisingly competitive performance.
- The authors contend that the second evaluation protocol, which employs a state-of-the-art model such as CatBoost, is more suitable for calculating the machine learning efficacy, despite the prevalent use of the first protocol in prior works. The firstly mentioned set of models[5] shows overall lower metric scores compared to the CatBoost model. Thus, the authors argue the set of other models'[5] performance values are of lower relevance for practitioners, who, in real-world scenarios, are rather interested in using state-of-the-art models rather than suboptimal models. Additionally, for the set of models, the models trained on synthetic data outperformed the models trained on actual real data, indicating that the synthetic data is "more valuable than the real" data, which should not be the case [Kot+22, p. 8]. This behavior cannot be observed if tuned MLP or CatBoost models are used for evaluation [Kot+22].

Additionally, the authors provide a small comparison based on distribution plots and correlation matrices. In their comparison, they show that the produced feature distribution plots by TabDDPM for a selected number of columns of different datasets look more similar to the real distributions, compared to plots produced by CTABGAN+ or TVAE [Kot+22]. The authors argue that this observation is consistent for numerical, categorical, and mixed data type columns [Kot+22]. The pairwise correlation matrices difference plots (performed similarly to in [BHV19]) show a smaller difference between the synthetic and real data for the TabDDPM model compared to CTABGAN+ and TVAE. Since the TabDDPM model show more realistic correlation difference plots for multiple datasets, the authors assert that this demonstrates that their TabDDPM model exhibits greater flexibility than other alternatives and generates synthetic data of higher quality [Kot+22].

Lastly, [Kot+22] performs a privacy evaluation against theSMOTEapproach. The median Distance to Closest Record (DCR) is, which calculates the minimum Euclidean distance to real data points for each synthetic sample [Zha+21c]. Small DCRs point towards mere copying of data samples (overfitting), while high values indicate actual new synthetic data samples [Kot+22]. The results presented by the authors show a higher DCR score for TabDDPM for all datasets compared toSMOTEwhile having a similar or higher machine learning efficacy score [Kot+22].

To summarize, the TabDDPM approach is a novel way to generate synthetic data of high utility using Gaussian and multinomial diffusion. TabDDPM outperforms other state-of-the-art baseline models in terms of machine learning efficacy on multiple datasets. Moreover, distributions and correlations produced from synthetic data by TabDDPM look more similar to those of the real data than synthetic data produced by other models.

[5] Decision Tree, Random Forest, Logistic Regression, MLP.

4 Conceptual Design

4.1 Requirements

In this chapter, the requirements for the synthetic data generation system will be first established. The goal is to gather a systematic collection of requirements on which the success of the developed system can be measured. Firstly, non-functional requirements will be explained before continuing with the functional requirements.

Non-Functional Requirements

Non-functional requirements (or qualitative requirements) describe requirements and constraints on a system that governs how the functions specified in the functional requirements are to be performed [BK21]. They focus on the question of how a system's functional requirements should be realized [BK21]. The ISO-Norm 25010 [ISO] defines the base characteristics relevant for a software system's quality [Hao+17]. The key elements of software architectures include functional suitability, reliability, performance efficiency, compatibility, usability, security, maintainability, and portability [Hao+17]. Each of these is of different importance for each software system. Additionally, [VB19] highlights that additional quality requirements might be necessary in the context of machine learning tasks, for which a standard has yet to be defined. These could include (but are not limited to) quantitative targets (i.e. measurable results), explainability, freedom from discrimination, legal and regulatory requirements, and data requirements.

This thesis focuses on a research prototype rather than a fully developed software system. Therefore, only a subset of the aforementioned requirements are prioritized during development. The following requirements R1 - R5 are the non-functional requirements for the software developed in this thesis:

R1 - Functional Suitability: Functional Suitability refers to "the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions" [BCK13, p. 219]. In the context of this thesis, the system first needs to be able to generate synthetic data and needs to be able to show how different tabular processing techniques influence the performance of the overall synthesis approach. This performance should be measured across different similarity measures.

R2 - Maintainability: Maintainability focuses on the "degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers" [BCK13, p. 220] One of the core questions of this thesis is to compare different tabular processing techniques in diffusion models. For this, a variety of different approaches are intended to be compared. Hence, it is required that the code and software architecture is designed in a maintainable way, such that it can be easily extended with additional processing techniques, not only by the developer, but also for future researchers who might build upon the work of this thesis.

R3 - Performance efficiency: In the context of deep learning, the program's performance is always essential since training, hyperparameter search, and inference usually demand a lot of computation time by nature. Consequently, the developed software should reduce unnecessary computations as much as possible.

R4 - Portability/Reproducibility: Portability is usually referring to the extent of "effectiveness and efficiency with which a system, product, or component can be transferred from one hardware, software, or other operational or usage environment to another" [BCK13, p. 220]. Since this thesis should encourage other researchers to reproduce and extend the developed codebase, the software should be able to run on other machines running a different operating system given a specified development environment, including packages used and their version numbers. In general, it is required that all results should be fully reproducible, given the parameters and the configuration of the experimental setups

R5 - Quantitative Targets: It is required that the different model versions produced in the thesis are compared using metrics that are commonly used in the domain of tabular data synthesis. This should enhance the comparability to other approaches.

Functional Requirements

Functional requirements describe what the software system should do and which practical aspects must be implemented. Please note that the software of [Kot+22] will be extended, which has consequences for certain functional requirements.

FR1 - Data Import and Export: The software shall support importing tabular data in the same format as specified in [Kot+22]. Kotelnikov et al. [Kot+22] makes use of datasets, that are already separated into numerical, categorical, and target columns, each saved in a NumPy [Har+20] array. Additionally, the datasets that will be imported have to be separated into training, validation, and testing sets. For the NumPy arrays to be imported, they need to be of file type ".npy". Synthetic data produced by models shall be saved in the same data segmentation format as the imported data.

FR2 - Tabular Processing: The software shall support tabular data processing techniques that are able to encode and decode tabular data. An *IdentityProcessor* that has no effect is required to be implemented to test models without any tabular processing mechanism. Additional data transformation techniques proposed by [Kot+22], including normalization or categorical encoding, shall still be supported as intended by the authors. Adding new tabular processing mechanisms shall not affect other existing tabular processing mechanisms. This requirement implies several sub-requirements:

FR2.1 - Tabular Processing Fit: If necessary, a Tabular processing mechanism should be able to be fitted according to the data. However, it is required that the tabular processing mechanism only receives training and validation data and does not have access to the test data.

FR2.2 - Tabular Processing Transform: Each Tabular processing mechanism needs to transform the raw data into a specified format.

FR2.3 - Tabular Processing Inverse Transform: Each Tabular processing mechanism needs to be able to inversely transform data back into its original form.

FR3 - Diffusion Process Configuration: Several experiments with different model configurations will be tested. The software shall allow users to configure the diffusion process by specifying parameters such as the number of iterations, batch size, sampling size, and others.

FR4 - Training, Tuning, Sampling, Evaluation: The software shall support the training of various model versions, including the TabDDPM diffusion model, variations of TabDDPM and other non-diffusion baseline models. The software shall allow finding of suitable hyperparameters in a specified hyperparameter search space. During training, relevant metrics and loss values shall be saved. The software shall generate synthetic samples from the trained diffusion model during the sampling process, where the user specifies the number of synthetic samples to be generated. The software shall handle the evaluation of synthetic data in multiple ways. Firstly, the existing evaluation framework of [Kot+22] shall remain. Additionally, a similarity evaluation shall be implemented according to [Chu+22]. Lastly, the software should also produce visualizations of certain properties of the synthetic data to allow a visual comparison with the same properties of the real data, for example, to allow for a comparison of distributions of columns.

FR5 - Modularity: The individual software components (tabular data processing, training, sampling, evaluation) shall be implemented in such a way, that they are easily replaceable by alternatives (for tabular data processing) or shall be executed individually (training, sampling, evaluation). For example, sampling synthetic data shall be

separated from the training so that not every sampling process requires training a model from the start.

4.2 Existing Code Base

The software [Aki23] used in this experiment was programmed in the programming language python [VD95] and is based upon the work of [Kot+22]. It is changed and modified according to the needs of the experiments. This section will first explain the architecture of the existing code base before introducing the proposed adaptations in the next section.

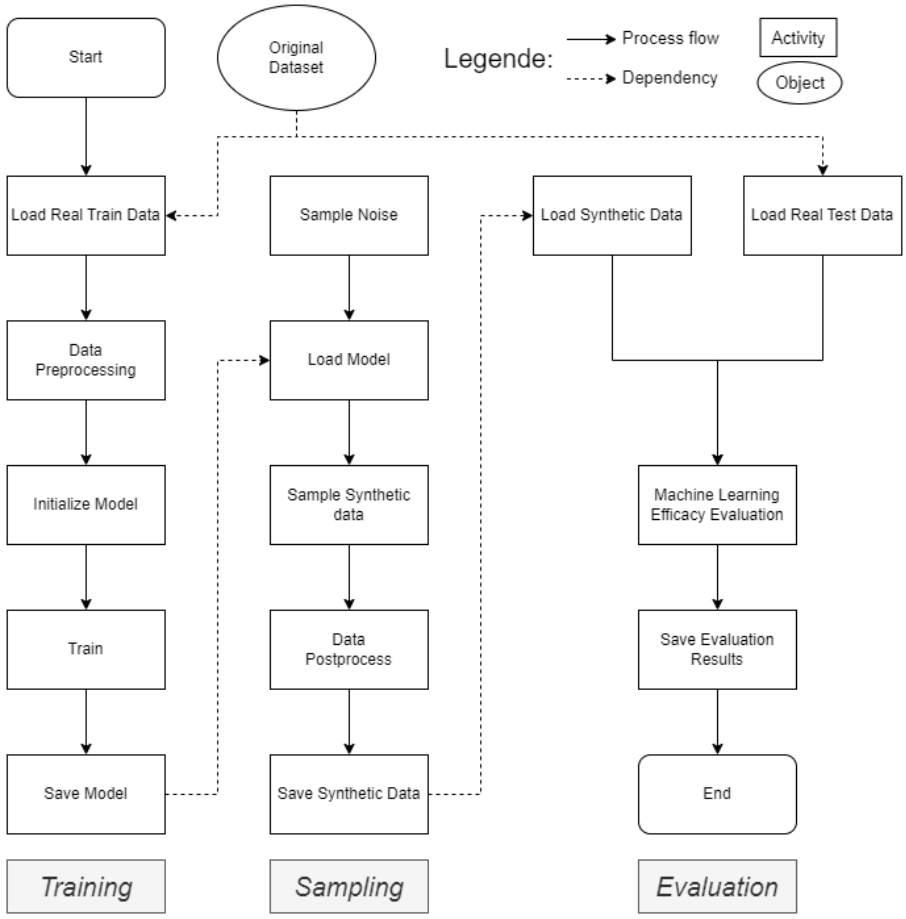


Figure 4.1: Overview synthetic data generation process in [Aki23]

4.2.1 Original Implementation

Akim [Aki23] provide a software implementation for their proposed TabDDPM. This implementation consists of multiple scripts and implements several of the functional requirements listed in Section 4.1, including FR1, FR4, FR5, and parts of FR4. This section will explain what kind of software was already provided by [Kot+22] and explain important architectural design choices.

Software Procedure

An overview of how the overall process in this synthetic data generation approach is done can be seen in Figure 4.1. The whole software process can be roughly summarized into three parts, the training, the sampling, and the evaluation. In the training, real training data is taken from the original dataset. This training data is preprocessed, according to a predefined preprocessing strategy that may vary from dataset to dataset (compare Subsection 2.1.1). After preprocessing, the diffusion model is initialized and trained to learn the underlying joint distribution of the training data. Once the training is finished, the sampling of new synthetic data can be executed. The previously trained model is loaded and synthetic data will be produced by denoising previously sampled noise. The synthetic data will be in the same format as the training data after it has been preprocessed. Hence, the synthetic data will be postprocessed by inverting the preprocessing, such that the synthetic data will be in the same format as the raw training data. As a last step, the synthetic data will be evaluated. For this, the held-back real test set will be loaded. With the real test set and the synthetic dataset sharing the same data format, a machine learning efficacy evaluation (compare Subsection 2.5.2) is performed to compare the two datasets. Ultimately, the results of this evaluation are saved and can be interpreted by the user.

Scripts

The implementation [Aki23] already provides the necessary scripts that allow for an easy training, sampling, evaluation, and hyperparameter tuning for their proposed TabDDPM model. Additional non-diffusion baseline models the TabDDPM is compared against are supported as well. The functionality of the most important scripts can be summarized in the following way:

train.py: The script "train.py" is responsible for the training of the diffusion model. It accepts configuration parameters that guide the training process. Initially, it loads the designated dataset and carries out preprocessing as per the given configuration. Subsequently, it initializes the necessary class instances and starts the training loop. Once the training loop is finished, the script saves the trained model along with its loss history.

sample.py: The script "sample.py" carries out the task of sampling from a pre-trained diffusion model. It takes in specific configuration parameters that are used during the sampling procedure. This script starts by loading a pre-trained model and generating samples from it. These new samples are then modified in accordance with the inverse of the preprocessing defined in the configuration. Once the transformation is complete, the resulting samples are saved.

eval_[catboost|mlp|simple].py: The scripts "eval_catboost.py", "eval_mlp.py", and "eval_simple.py" evaluate a synthetic dataset based on machine learning efficacy. In

this process, a machine learning model, chosen by the user, is trained on either real or synthetic data, as specified in the configuration. The model's performance is then assessed on the real test set by computing a set of metric scores.

pipeline_*.py[1]: The scripts, denoted as "pipeline_*.py", define the full processing pipeline, which includes the processes of training, sampling, and evaluation. They ensure that each function within the pipeline correctly receives its necessary attributes from the configuration. These scripts also provide the flexibility to execute specific portions of the pipeline individually.

tune_*.py[2]: The "tune_*.py" scripts are generally responsible for the hyperparameter tuning process. It starts by defining a hyperparameter search space (see [Kot+22, Table 1, p. 4] and [Kot+22, Table 7-11, p. 13 f.]). Next, an objective function is defined that is maximized for 50 trials through the Optuna framework [Aki+19]. Within the objective function, a set of hyperparameters is selected from the search space. These selected hyperparameters are used to invoke *pipeline.py*, which initiates the model training Afterward, for five different random initializations, *pipeline.py* is called to sample and evaluate the model, creating, and assessing five different synthetic dataset versions. In each training, sampling, and evaluation, the training and validation set are redistributed and shuffled to implement some form of cross-validation [Koh95]. The average evaluation score (machine learning efficacy based) of the five synthetic dataset versions is returned as the objective that Optuna tries to maximize. After optimization, the best hyperparameter configuration is saved. If the user has set an - - *eval_seeds* flag, a final evaluation of the best-found model will be started by calling *eval_seeds.py*.

eval_seeds.py: The "eval_seeds.py" script's purpose is to perform an extensive evaluation, given a trained model. The script starts by loading the pre-trained sampling model
([TabDDPM | SMOTE | CTABGAN | CTABGAN+ | TVAE]). Given the sampling model, *n_datasets* synthetic datasets will be produced depending on the specified evaluation parameters by calling the sampling script in the respective *pipeline_*.py* script. For each produced dataset, evaluation is performed using a predefined evaluation model ([Catboost | MLP]) for a predefined number of random initializations. Ultimately, the average metrics are calculated over all performed runs, reported, and saved.

In summary, to get a fully optimized diffusion model with an extensive evaluation, the user just has to run the *tune_ddpm.py* script with the - - *eval_seeds* flag. A detailed activity diagram for each script can be found in the Appendix A.2.

[1] Each implemented baseline model (SMOTE, TVAE, CTABGAN, CTABGAN+) has its own implementation as pipeline_*modelName*.py.
[2] Each implemented baseline model has its own implementation as tune_*modelName*.py.

Configuration

In order to start one of the above scripts, a configuration file has to be specified, detailing the most important parameters for training, sampling, and evaluation. For example, Listing 4.1 shows how such a configuration file could look like and which parameters need to be specified. The file is stored as a TOML file (short for "Tom's Obvious, Minimal Language") [Pre21], which contains sections, indicated by squared brackets. Within these sections, key-value pairs define configuration options.

Listing 4.1: Example Configuration File

```
parent_dir = "exp/adult/check"
real_data_path = "data/adult/"
num_numerical_features = 6
model_type = "mlp"
seed = 0
device = "cuda:0"

[model_params]
num_classes = 2
is_y_cond = true

[model_params.rtdl_params]
d_layers = [
    256,
    256,
]
dropout = 0.0

[diffusion_params]
num_timesteps = 1000
gaussian_loss_type = "mse"
scheduler = "cosine"

[train.main]
steps = 1000
lr = 0.001
weight_decay = 1e-05
batch_size = 4096

[train.T]
seed = 0
normalization = "quantile"
num_nan_policy = "__none__"
cat_nan_policy = "__none__"
cat_min_frequency = "__none__"
```

```
36    cat_encoding = "__none__"
37    y_policy = "default"
38
39    [sample]
40    num_samples = 216000
41    batch_size = 10000
42    seed = 0
43
44    [eval.type]
45    eval_model = "catboost"
46    eval_type = "synthetic"
47
48    [eval.T]
49    seed = 0
50    normalization = "__none__"
51    num_nan_policy = "__none__"
52    cat_nan_policy = "__none__"
53    cat_min_frequency = "__none__"
54    cat_encoding = "__none__"
55    y_policy = "default"
```

The key *model_type* refers to the model that should approximate the noise in the reverse noising process. Most parameters are self-explanatory, controlling one specific part of the pipeline. For example, all *model_params* specify different relevant aspects for instantiating the model. Inside the *model_params* section, the *num_classes* key indicates the number of possible target classes the given dataset has, and *is_y_cond* indicates if the dataset is a classification (=*True*) or regression dataset (=*False*). The *model_params.rtdl_params*[3] specify the mlp architecture, by controlling the number of layers and the amount of dropout. As the name suggests, the keys inside the *diffusion_params* section control the diffusion process. Noteworthy are the *train.T* and *eval.T* sections, whose keys define how preprocessing is done at the beginning of training and evaluation, respectively. The *eval.type* section keys are useful to further specify the kind of evaluation that will be executed. The machine learning efficacy model is determined by the *eval_model* key and is, in this example, set to the CatBoost model. To control what kind of data will be evaluated, the *eval_type* key can be set to either "real" or "synthetic". If the value is set to "real", the real training data is loaded and compared to the real test data, as if it was synthetic data. This setting allows the creation of baseline metric scores against which synthetic values are compared. As this is usually done only once per dataset, *eval_type* is usually set to "synthetic" to calculate metric scores for the newly created synthetic data. This configuration structure allows to efficiently perform a variety of experiments with different key configurations by simply changing the configuration file and inputting it into the next experimental run.

[3] rtdl is an abbreviation for the python library [Gor+21] from which the MLP model architecture was used.

Dataset Segmentation

The authors provide access to 12 different datasets, which have already been cleaned and formatted by a program provided by Gorishniy [Gor23]. In order to use and process the data, the software of [Kot+22] is built towards processing the data that was segmented as specified by Gorishniy [Gor23]. For the purpose of clarity in this thesis, any future reference to "data segmentation" will refer to the separation of a dataset into multiple parts, detailed as follows:

Each dataset contains data separated into multiple subcomponents, saved as individual files. To create these subcomponents, the columns of the dataset are manually classified into categorical, numerical, ("X") and target ("y") columns. Secondly, the dataset records are distributed into a training, validation, and testing set. Therefore, each dataset is separated into nine parts, conforming to the subsequent naming convention: X_[cat | num]_[train | val | test].npy and for the target y_[train | val | test].npy

Additionally, a *info.json* file is provided, containing the necessary information for handling the data. In Listing 4.2, one can see how such an info-file contains the information on how the splits mentioned above have been done and what kind of task this dataset usually has (it is differentiated between binary classification (binclass), multiclass classification (multiclass), and regression (regression)).

Listing 4.2: Example Data-info File

```
{
"name": "Adult",
"id": "adult--default",
"task_type": "binclass",
"n_num_features": 6,
"n_cat_features": 8,
"test_size": 16281,
"train_size": 26048,
"val_size": 6513
}
```

Inside the code of TabDDPM [Aki23], a custom dataset class provided by the author handles any dataset that follows the above naming convention and provides a matching information file.

4.3 Proposed Concept

Changes to the current architecture have been made to answer the research questions specified in Section 1.2. Figure 4.2 shows that the overall synthesis process does not deviate much from Figure 4.1. Changes are highlighted in green and affect each step in the diffusion process: the training, sampling, and evaluation step. For a detailed overview of how each individual scripts changed, please refer to Appendix A.2.

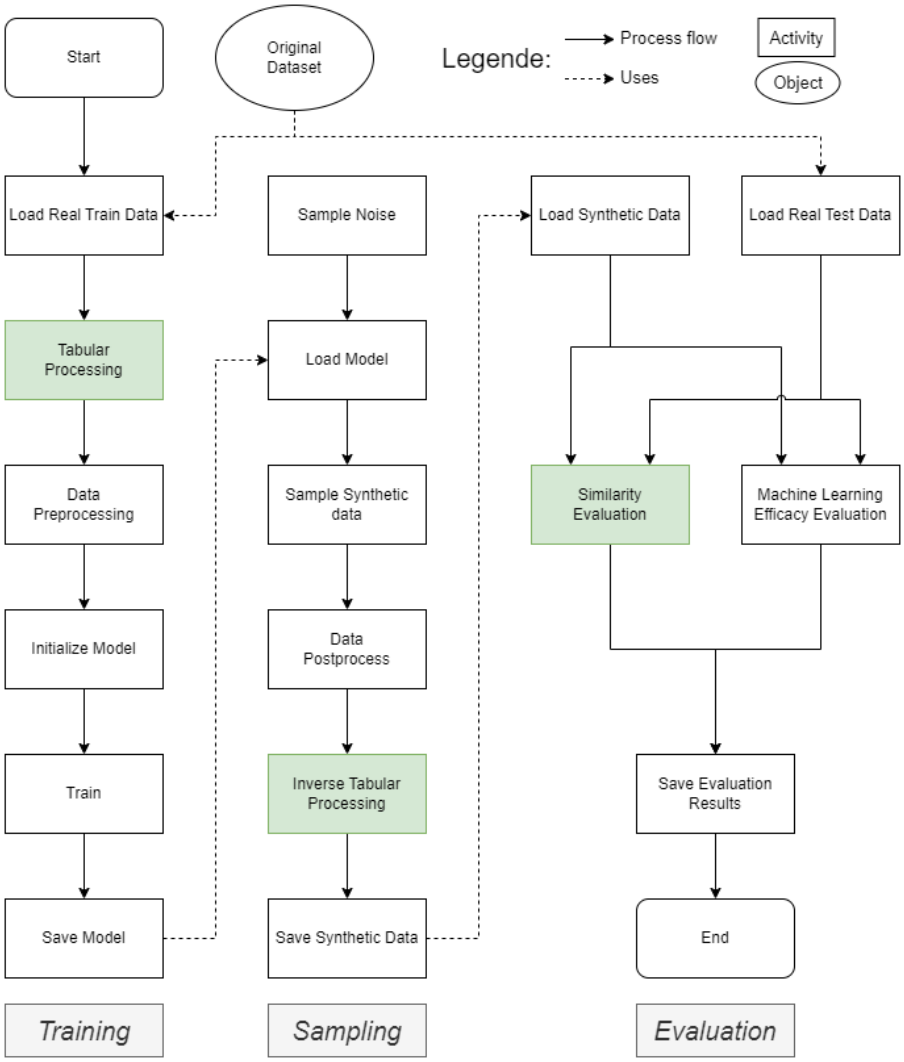


Figure 4.2: Overview of proposed synthetic data generation process changes (highlighted in green)

C1-Tabular Processing: To test the effect of different tabular processing techniques, a "Tabular processing" step is required before training the diffusion model. Within this processing, the raw data is transformed according to the defined tabular processing strategy. The tabular processing data output format needs to be the same as before the transformation, described in Subsection 4.2.1, to ensure that the remaining pipeline remains functional and keeps changes to the original code minimal.

C2-Inverse Tabular Processing: Similar to the data preprocessing, which requires an inverse data postprocessing, to turn the data back into its original format and scale, the same is necessary for the tabular processing. Since the model is trained on data that has been transformed according to the tabular processing mechanism, the diffusion model will produce data in the same format. Consequently, the synthetic data must be transformed back by an inverse function of the tabular processing mechanism before it is evaluated.

C3-Similarity Evaluation: Lastly, an extended evaluation of the synthetic data's similarity to the real data should be performed. For this, the original machine learning efficacy evaluation remained unchanged. However, the evaluation pipeline is extended by a similarity evaluation that does not only calculate a unified score, TabSynDex [Chu+22], but also reports other metric results that are of interest. Hyperparameters of the generation models can also be tuned towards the TabSynDex score. In addition to the numeric metric results, several visualizations comparing real and synthetic data will be generated.

4.3.1 Tabular Processing Criteria

The main source for finding the processing mechanisms should be academic literature, where said mechanism has been successfully applied to a related problem. The selection of which tabular processing mechanisms are realized heavily depends on the requirements defined in Section 4.1.

For a tabular processing mechanism to be used, the following conditions must be met:

1. **Reversibility:** the tabular processing mechanism must not only transform the data into a different format, but it also must be able to revert the transformation such that the data can be transformed back into its original form.

2. **Compatibility:** the data must be able to be separated into numerical and categorical parts after the transformation since the remaining code expects data in this format, as specified in Subsection 4.2.1.

3. **Complexity:** the time it takes to transform the data should be within a reasonable timeframe. They shall not add too much computation time to the already extensive model-tuning process.

4. **Availability:** due to the limited time frame of this thesis, the code for the mechanisms should be publicly available (or could be implemented within a reasonable timeframe).

4.3.2 Evaluation

The evaluation framework shall contain multiple evaluation aspects. Firstly, the existing machine learning efficacy using a CatBoost model proposed by [Kot+22] shall be executed and remain untouched. In addition, the TabSynDex [Chu+22] similarity will be performed in each evaluation. Machine learning efficacy that is based upon an MLP or using a set of classification models will not be performed for the following reasons:

1. [Kot+22] argues convincingly that machine learning efficacy not based on CatBoost is less informative compared to the CatBoost counterpart.

2. The TabSynDex metric contains a machine learning efficacy metric based upon a set of models.

Consequently, machine learning efficacy will be evaluated twice in each evaluation, once based on CatBoost and once inside the TabSynDex metric using a set of standard machine learning models.

Lastly, for a visual evaluation, several plots indicating the similarity of the synthetic data to the real data shall be generated using the implementation of Brenninkmeijer, Hille, and Vries [BHV19]. If necessary, modifications to the plots may be required.

5 Methodology

This chapter delves into the methodology employed to address the research questions and objectives outlined in the previous chapters. The chapter is organized into three main sections. Section 5.1 discusses the architecture of our proposed solution, elaborating on the tabular processor and its implementations, namely the BGMProcessor and FTProcessor. In addition, this section also highlights the modifications made to the existing software to incorporate the proposed solutions.

Section 5.2 focuses on the experimental setup, outlining the experiments conducted to evaluate the performance and effectiveness of the proposed models. It provides detailed information on the execution environment and the hyperparameter search space used in the study, ensuring the reproducibility of the results.

Finally, Section 5.3 presents the dataset utilized in the experiments, including its source and characteristics. This information is crucial to understanding the context in which our proposed methodology was tested and evaluated.

By the end of this chapter, the reader will gain a comprehensive understanding of the methodology implemented in this thesis, which will lead to the presentation and analysis of the results in the subsequent chapters.

5.1 Architecture

The software's architecture was developed with the requirements in Section 4.1 in mind. The following section will explain how the individual additions to the existing code are designed and why the design was chosen.

5.1.1 Tabular Processor

To allow for flexible use and the possibility of adding additional processing mechanisms (R5 - Maintainability), the *strategy design* pattern (Figure 5.1) was chosen [Gam+94]. The pattern is beneficial if a specific behavior is required, but needs to be realized in different ways. The *Strategy* class defines what kind of methods need to be implemented and what behavior the class should have. The *Strategy* classes children define a *ConcreteStrategy* and realize the desired behavior in different ways. Different *ConcreteStrategies* can be used interchangeably independent of the *Context* they are used in since all implement the same functionalities. The *Context* is has a reference to the *ConcreteStrategies* and ultimately decides when an instance of type *Strategy* is executed. As the *ConcreteStrategies*

implement the same interface, the *Context* can execute any function a *ConcreteStrategies* has implemented, without needing to know the inner workings of that function. Hence, whenever the *Context* needs to execute a certain functionality, it delegates the execution to the *ConcreteStrategy* object it needs, which may vary on the situation. One of the biggest advantages is, that this way new *ConcreteStrategies* can be added easily without affecting any of the other *ConcreteStrategy* or the *Context* [Gam+94].

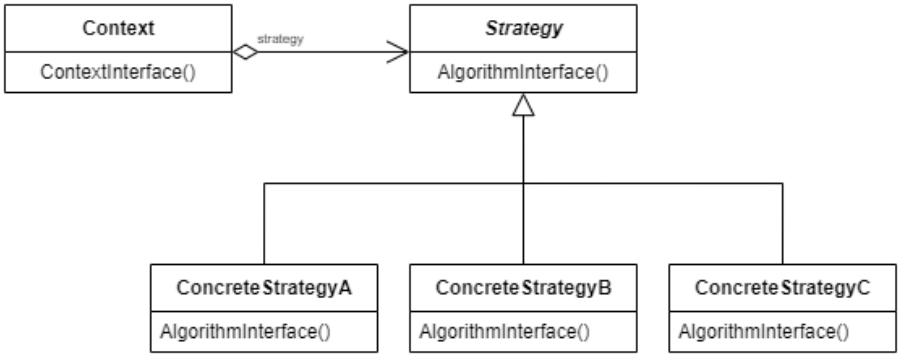


Figure 5.1: Strategy design pattern [Gam+94, p. 316]

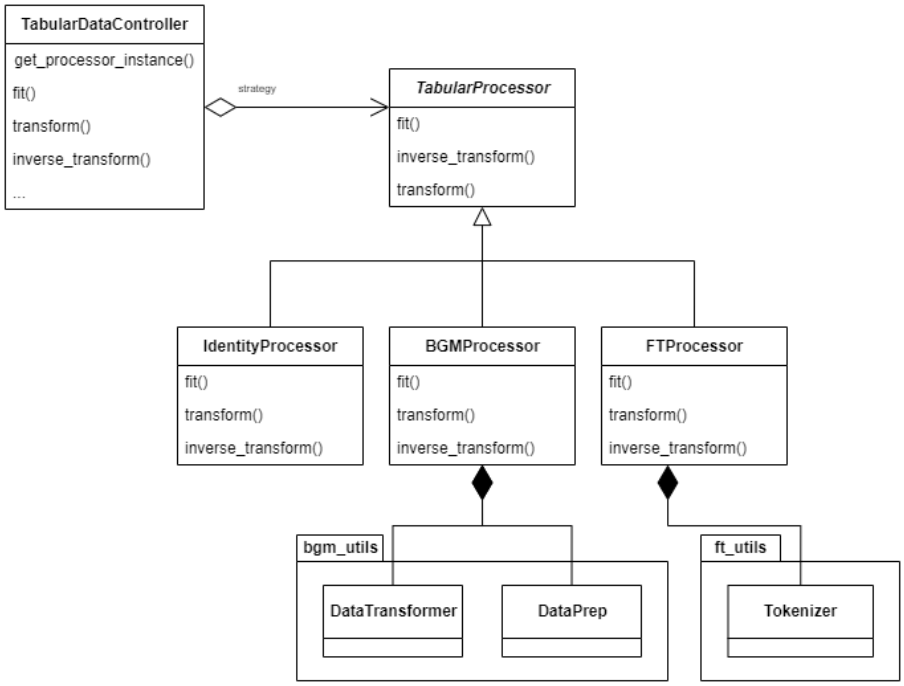


Figure 5.2: Tabular Processor Design

Figure 5.2 shows how the *strategy design* pattern is realized. The overall *Strategy* class is defined in the *TabularProcessor* class in the form of an abstract class with three core methods each tabular processing mechanism needs to implement. These functions are the *fit*, *transform*, and *inverse transform* functions (FR2.1 - FR2.3).

Each tabular processing strategy is different and realizes the transformation of the data in a different way by implementing the abstract functions of the parent strategy.

Therefore, new tabular processing mechanisms can be easily added by just implementing the methods of the abstract class (R2, FR2). Furthermore, the tabular processing mechanism can be exchanged easily since they share the same functions. This is handled in the *TabularDataController* class, which is equivalent to the *Context* of the *strategy design* pattern. The *TabularDataController* handles all relevant aspects to use the tabular processing mechanisms, including but not limited to the instantiation, fitting, saving, and loading of *TabularProcessor* instances. The data, whose columns are segmented into categorical, numerical, and target as numpy arrays [Har+20], needs to be provided to instantiate a tabular processor. Target refers to the column of the dataset that a model tries to predict or estimate in a classification or regression scenario. The first step in using the tabular processor is fitting it to the data. Additional information required for the fitting process can be provided through meta-data in the form of a dictionary, which may change for different realizations. Note that this fit function might not be necessary, depending on the processing mechanism, but it is required to be implemented anyways. Only after fitting the transform function can be called transforming the data format and returning the transformed data in the same data segmentation structure, i.e. separated into categorical, numerical and target arrays. The inverse transformation works similarly, reversing transformed data back to its original format. For clarity, please note the difference between the terms "data format" and "data segmentation" in this thesis. The term "data segmentation" refers to the separation of columns into categorical, numerical, and target segments. On the other hand, the term "data format" refers to the modification of the data values achieved through the encoding and decoding processes of the *TabularProcessor*, specifically via their "transform" and "inverse transform" functions.

5.1.2 Tabular Processor Implementations

Three different tabular processor versions have been implemented. The *IdentityProcessor* does not do anything to the data. It is used to do experiments without any tabular processing mechanism and allows reproducing the original results of [Kot+22]. In this thesis, only two approaches have been implemented, the BGM and FT tabular processors. Other potential tabular processing mechanisms have been considered but usually lack one of the criteria specified in Subsection 4.3.1. The biggest challenge usually is the reversibility, since several encoding strategies only allow an encoding of the data but not a decoding of already encoded data. The reason for this is that usually, decoding of encoded data is not necessary for most other deep learning tasks, where the encoded data is directly fed into the model. Consequently, a decoding mechanism would need to be developed if one is not present, which is depending on the encoding technique, not a straightforward task and would likely require a special decoding model that is specifically trained for the encoding technique at hand. However, this would exceed the scope of this thesis, as it would be very complex and time-consuming. Therefore, it remains an open task for future researchers.

BGMProcessor

The BGMProcessor[1] is the same processing mechanism that has been introduced by the CTABGAN+ model [Zha+22]. This processing mechanism was chosen since it fulfills all criteria mentioned in Subsection 4.3.1. Moreover, it is part of a state-of-the-art tabular data synthesis GAN-based model, which has already proven to be successful.

Firstly, a mixed-type encoding is introduced, specifically designed to encode single columns containing a mixed data type of numerical and categorical data types. For the continuous values in the column, the authors adapt the mode-specific normalization technique from [Xu+19] using a BGM implementation [dev23a], which is a VGM (see Subsection 2.1.1 for details). For the categorical values, the One-Hot vector β, which usually just indicates how many modes for normalization have been used, is extended by the number of possible categorical values in the column. If a categorical value should be encoded, the α value is set to 0, and the One-Hot encoding indicating the specific category is set to 1. Additionally, the authors allow missing values, by modeling them as another categorical entry.

Furthermore, the authors introduce a "general transform" mechanism [Zha+22, p. 7] that is supposed to be used for simple distributions and counters the problem of exploding dimensionality caused by a One-Hot encoding [Zha+22]. This general transformation is mapping the values into a range of [-1, 1], which is achieved through a shifted and scaled min-max normalization (see min-max scaler in Subsection 2.1.1). Mathematically, the encoded datapoint x_i^t can be computed using the original datapoint x_i in column x through the following formula:

$$x_i^t = 2 * \frac{x_i - \min(x)}{\max(x) - \min(x)} - 1 \tag{5.1}$$

Which can easily be reversed in the following way:

$$X_i = (\max(x) - \min(x)) * \frac{x_i^t + 1}{2} + \min(x) \tag{5.2}$$

Categorical values are label encoded before applying the normalization [Zha+22]. The authors observe that this simplistic general transformation for continuous values only works well for simple distributions (e.g. single-mode Gaussian) and not for complex distributions, for which the mode-specific normalization is preferred [Zha+22]. For categorical values, the general transformation should only be applied if the number of categories a column can take is very high and would lead to very sparse vectors that make computations complex [Zha+22]. Lastly, the authors log-transform columns that suffer from very long tails (see Subsection 2.2.2) because BGMs seem to have difficulties encoding values towards the tail [Zha+22]. The log-transformation of each value τ in the column will be

[1] The name was chosen since a Bayesian Gaussian Mixture Model (BGM) is the main component used in the processor.

compressed to τ^c, given a lower bound l using:

$$\tau^c = \begin{cases} log(\tau) & \text{if } l > 0 \\ log(\tau - l + \epsilon) & \text{if } l \leq 0, \text{ where } \epsilon > 0 \end{cases} \tag{5.3}$$

This encoding strategy is realized in the BGMProcessor class, which uses the DataPrep and DataTransformer class, as developed by [Zha+22]. These classes require additional information about the dataset, provided by the user, including what columns are categorical, categorical columns that have a high cardinality/dimensionality[2], numerical, mixed (including what values are categorical), as well as what columns require a log transformation and which a general transformation. This information is provided to the *BGMProcessor* by extending the info.json (Listing 4.2) with an additional entry, *dataset_config*, see Listing 5.1 for an example.

Listing 5.1: Example extended data info file from the adult dataset (Section 5.3)

```
{
"name": "Adult",
[...]
"val_size": 6513,
"dataset_config": {
    "cat_columns": [ // categorical columns
        "workclass",
        (...),
        "income"
        ],
    "non_cat_columns": [], // high dim-categorical
    "log_columns": [], // log-transformation
    "general_columns": ["age"], //  general-transformation
    "mixed_columns": { // mixed-data-types
        "capital-loss": [0.0], // the "0.0" value is categorical
        "capital-gain": [0.0]
        },
    "int_columns": [ // numerical columns
        "age",
        (...),
        "hours-per-week"
    ],
    "problem_type": "binclass", // [binclass|multiclass|regression]
    "target_column": "income"
    }
}
```

[2]Inside the software of the authors, they refer to it as "non_categorical_columns", which is a bit misleading. "Non_categorical_columns" are categorical columns with high dimensionality that are transformed into numerical columns and handled as if they were continuous.

FTProcessor

The *FTProcessor*, short for *Feature Tokenization Processor*, is based upon the work of Zheng and Charoenphakdee [ZC22] and Gorishniy et al. [Gor+21]. Similar to the BGM variant, it fulfills all criteria stated in Subsection 4.3.1. However, the FT mechanism has only been used in a data imputation task so far, where only individual cells must be synthesized. It is unclear whether its functionality can be extended to multiple-cell synthesis.

The *FTProcessor* transforms a tabular data input x with k features (or columns) into a static embedding representation $T \in \mathbb{R}^{k \times d}$ with d as the embedding dimensionality. Categorical columns are embedded through an embedding layer, which is basically a look-up table [PyT23] of fixed size. Each categorical input tensor of dimensions $[N]$ (i.e. a table with N columns) will be transformed in an embedding of size $[N, H]$ with H as the embedding dimensionality [Gor+21]. Numerical columns will be processed by a simple multiplication of linear layers weights W [Gor+21]. Lastly, a bias term b is added to both categorical and numerical columns. Figure 5.3 illustrates how a tabular data row with three numerical and two categorical features could be transformed into a new tensor [Gor+21, Figure 2a, p.4].

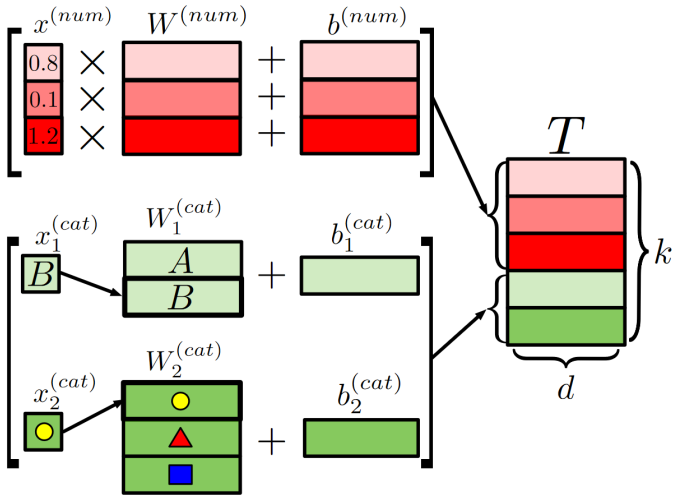


Figure 5.3: Feature Tokenization [Gor+21, Figure 2a, p.4]

To revert the synthetic data that the diffusion model produces to the original data format, the numerical and categorical columns need to be handled differently. The corresponding embedding weights divide the numerical output element-wise, and the average value is computed as the final output [ZC22]. This transforms output having the same dimensionality as the embedding (d) back into a single value [ZC22]. For categorical output, the closest categorical embedding is determined by computing the Euclidean distance between the produced output and every categorical embedding [ZC22].

In the original work of [Gor+21], the feature tokenizer is directly applied in front of a transformer model, allowing the gradients to flow through the embedding and linear layers during training. This enables the model to learn a meaningful embedding. In

the adaptation of [ZC23] and in this thesis, the weights of the layers are frozen. Hence, learning is not possible, and the embedding is static. The reason for this in this thesis have been because of the overall software architecture. The data transformation and the actual model training are fully separated, resulting in gradients that do not flow back to the feature tokenizer to update any weights of the embeddings. To achieve the update of weights, the tokenizer would need to be fully integrated into the diffusion model architecture, which in turn could have negatively affected the overall maintainability, compatibility, and performance (see R2-R4, Section 4.1) of the software.

5.1.3 Changes to the existing software

The changes made to the existing software (Section 4.2) are various and include multiple major changes as well as a lot of small changes. Below, the major changes to the individual scripts are listed. An additional visualization of the process of the scripts can be found in Appendix A.2

train.py: Before the dataset is preprocessed and transferred into a dataset class, the *TabularDataController* handles the tabular processing mechanisms. For this, a *TabularDataController* is instantiated at the beginning and either loads a previously fit *TabularProcesser* instance or fits and saves the *TabularProcesser* on the training data. It receives context information on which tabular processing mechanisms and which properties the dataset has through the configuration file and handles the instantiation, fitting, and transformation of the raw data. Despite the addition of the *TabularDataController*, the actual training remained untouched. Hence, introducing a *TabularProcesser* only changes the data format (and maintains the segmented data structure) the diffusion model will receive and does not change the actual training loop.

sample.py: A TabularDataController is instantiated at the beginning and loads a previously fit *TabularProcesser* instance. If there is no previous *TabularProcesser* instance to be loaded, it will call the fit function again. After the sampling is finished, the *TabularProcesser*'s inverse transform function is called to bring the synthetic data into the original data format.

eval_similarity.py: This script does not replace the previous evaluation script. Instead, it is called after the original evaluation script (see *pipeline.py*). First, three *TabularDataController* instances are created, one for each train-, validation- and test data split. With the *TabularDataController* instances, Pandas [Wes10] data frames are created, which are required for the TabSynDex similarity score. After the TabSynDex score is calculated, visualizations are created, if set by the user.

pipeline_*.py: The basic pipeline hardly changed. The only major change is that the similarity score evaluation is performed after the default evaluation using machine learning efficacy. In this evaluation, the TabSynDex metric is calculated as well as (if set by the user) visualization are created.

tune_*.py: Inside the tuning script, the objective function changed. If set by the user, the objective function will be based on the TabSynDex similarity score instead of the machine learning efficacy. To reduce processing time, visualizations inside the similarity evaluation part will, by default, only be created after the hyperparameters have been optimized and the eval_seeds script will be executed.

eval_seeds.py: The basic eval_seeds script did not change a lot. Again, the TabSynDex metric will be called after the default machine learning efficacy.

In order to easily change the *TabularProcesser* type that should be executed in an experiment, the experiment configuration displayed as Listing 4.1 was extended. The additional configuration section *tabular_processor* has been added, containing a *type* key. The *type* key can be either set to "identity", "ft" or "bgm" to indicate, which *TabularProcesser* type should be used.

5.2 Experimental Setup

5.2.1 Experiments

To systematically analyze the performance of the different tabular processing mechanisms, the following baseline experiments have been performed, creating a set of baseline models:

Baseline-Real: Comparison of the real training data with the real test data. The similarity evaluation is expected to be very high since the data is from the same joint distribution. The computed machine learning efficacy score can be seen as an objective value that should be reached. The closer the synthetic data based machine learning efficacy results are to this target, the better the synthetic data.

Baseline-TabDDPM: Reproduction of the TabDDPM results of the authors [Kot+22] but with the extend evaluation explained in Subsection 4.3.2 (Machine learning efficacy + TabSynDex + Visualizations).

Baseline-TVAE: Like Baseline-TabDDPM but for TVAE.

Baseline-CTABGAN: Like Baseline-TabDDPM but for CTABGAN.

Baseline-CTABGAN+: Like Baseline-TabDDPM but for CTABGAN+.

Baseline-SMOTE: Like Baseline-TabDDPM but for SMOTE.

For each implemented Tabular Processing mechanism, the following three experiments should be performed:

Experiment 1: Tabular Processing combined with TabDDPM with additional similarity evaluation. Hyperparameters tuned like in the original experiment (tuned after CatBoost-machine learning efficacy score).

Experiment 2: Like Experiment 1, but hyperparameters are optimized towards the TabSynDex similarity score instead of the machine learning efficacy score.

Experiment 3: Like Experiment 2, but the preprocessing strategy, that normalizes numerical data after the tabular processing transformation is replaced from a quantile transform function to a min-max transformation or is completely removed.

Depending on the results of the first experiments, subsequent experiments may only be performed on a subset of promising Tabular processing mechanisms. To not get confused with the different model versions, the models are named after the following principal:

$$\text{Modelname[-TabularProcessor]}^{tuning}_{preprocessing}$$

The Modelname is followed by an optional tabular processing mechanism. The tuning strategy ("ml" for "machine learning efficacy score" or "s" for "TabSynDex similarity score") is indicated via a superscript (Experiment 2). The preprocessing strategy ("q" for "quantile transform" or "m" for "min-max", or "n" for "none"/"no transformation") is indicated via a subscript (Experiment 3).

The min-max strategy was chosen as a commonly used linear transformation counterpart to the non-linear quantile transformation. Since some tabular processing strategies already normalize or standardize the data, additional quantile or min-max transformations may not be required, which is why experiment 3 will also investigate the effect of applying no transformation after the tabular processor.

5.2.2 Execution Environment

The code was developed in python version 3.9.7 and made use of several libraries. Most important libraries include (for a full list, please see the *environment.yml* file in the source code):

- azure-core, azureml-core: For running the code in the Microsoft Azure cloud [Mic23].
- catboost (1.0.3): contains the CatBoost model for the machine learning efficacy.
- pytorch (1.10.1): neural network framework.
- table_evaluator (1.4.2): used to produce visualizations.
- numpy (1.21.4): allows efficient array computation.

- optuna (2.10.1): used for hyperparameter tuning.
- pandas (1.5.2): allows fast computations with dataframes of tabular data.
- scikit-learn (1.0.2): includes several utility functions for machine learning, e.g. metrics such as accuracy or F1-score.
- TabSynDex [Chu22]: TabSynDex metric source code.

The training was performed using Microsoft Azure Machine Learning Studio. Each experiment was executed on a STANDARD_NC6 compute cluster running on a Linux distribution, consisting of six virtual Central Processing Units (CPUs) (Intel Xeon E5-2690v3) with 56 GB Memory, 340 GB (SSD) Storage and the computing of one-half GPU (Tesla K80) with 12 GB GPU memory [vMSF22]. All experiments could also be performed locally on a Windows machine using a CPU. Local experiments with a GPU should be possible but have not been tested.

5.2.3 Hyperparameters

The hyperparameter search space for the various models was not changed and is equal to the experiments in [Kot+22]. The search space for the different models is listed in Tables 5.4, 5.1, 5.2 and 5.3.

Parameter	Distribution
Learning Rate	LogUniform[1e-5,3e-3]
Batch Size	Cat{256,4096}
Diffusion timesteps	Cat{100,1000}
Training iterations	Cat{5000,10000,20000}
# MLP layers	Int{2,4,6,8}
MLP layer width	Int{128,256,512,1024}
Proportion of samples	Float{0.25, 0.5, 1, 2}
Train size	#entries in training dataset
# Samples	Proportion of samples * Train size
Dropout	0.0
Scheduler	cosine
Gaussian diffusion loss	mse
Number of tuning trials	100

Table 5.1: TabDDPM model hyperparameter tuning search space [Kot+22]

Parameter	Distribution
# classif. layers	UniformInt[1,6]
Classif. layer size	Int{62, 128, 256, 512}
Training iterations	Cat{5000, 20000, 30000}
Batch size	Cat{456,4096}
Embedding dim.	Int{16,32,64,128,256,512,1024}
Loss factor	LogUniform[0.01, 10]
Proportion of samples	Float{0.25, 0.5, 1, 2, 4, 8}
Number of tuning trials	100

Table 5.2: TVAE model hyperparameter tuning search space [Kot+22]

Parameter	Distribution
# classif. layers	UniformInt[1,4]
Classif. layer size	Int{62, 128, 256}
Training iterations	Cat{1000, 5000, 7500}
Batch size	Int{512,1024,2048}
random dim.	Int{16,32,64,128}
# Channels	Int{16, 32, 64}
Proportion of samples	Float{0.25, 0.5, 1, 2, 4, 8}
Number of tuning trials	30

Table 5.3: CTABGAN/CTABGAN+ model hyperparameter tuning search space. Training iterations and the number of tuning trails were reduced compared to the original [Kot+22], to reduce computation time.

Parameter	Distribution
Max depth	UniformInt[3, 10]
Learning rate	LogUniform[1e-5, 1]
Bagging temperature	Uniform[0,1]
L2 leaf reg	LogUniform[1,10]
Leaf estimation iteration	UniformInt[1,10]
Number of tuning trials	100

Table 5.4: CatBoost evaluation model hyperparameter tuning search space (proposed by [Gor+21])

5.3 Dataset

The dataset that was used for the experiments was the *Adult* dataset, also known as "Census Income" from the UCI Machine Learning repository [DG17]. The dataset was constructed through an extraction from a 1994 census database [Koh96]. Overall, the dataset has 15 columns, of which nine are categorical and six are numerical. In total, the dataset consists of 48842 rows.

The numerical columns are: *age*, *fnlwgt*[3], *education-num*, *capital-gain*, *capital-loss* and *hours-per-week*.

The categorical columns are: *workclass*, *education*, *marital-status*, *occupation*, *relationship*, *race*, *sex*, *native-country*, and *income* (target column).

The dataset was created for binary classification, where the model should predict the column value of the income column (either *>50K* or *<=50K*). Table 5.5 shows five example entries of the adult dataset:

age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship
39.0	State-gov	77516.0	Bachelors	13.0	Never-married	Adm-clerical	Not-in-family
50.0	Self-emp-not-inc	83311.0	Bachelors	13.0	Married-civ-spouse	Exec-managerial	Husband
38.0	Private	215646.0	HS-grad	9.0	Divorced	Handlers-cleaners	Not-in-family
53.0	Private	234721.0	11th	7.0	Married-civ-spouse	Handlers-cleaners	Husband
28.0	Private	338409.0	Bachelors	13.0	Married-civ-spouse	Prof-specialty	Wife

(a) First eight columns of Adult income dataset

race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
White	Male	2174.0	0.0	40.0	United-States	$\leq$50K
White	Male	0.0	0.0	13.0	United-States	$\leq$50K
White	Male	0.0	0.0	40.0	United-States	$\leq$50K
Black	Male	0.0	0.0	40.0	United-States	$\leq$50K
Black	Female	0.0	0.0	40.0	Cuba	$\leq$50K

(b) Second seven columns of Adult income dataset, including the dataset target column "income"

Table 5.5: Adult income dataset with five exemplary entries

[3]Short for "final weight", a sampling weight.

6 Results

In this chapter the results from the previous in Chapter 5 introduced experiments are presented. Firstly, Section 6.1 deals with a reproduction and a verification of the results achieved by Kotelnikov et al. [Kot+22]. Subsequently, Section 6.2 will present the numerical metric results of the experiments that have been outlined in Chapter 5. These include the metric results from the machine learning efficacy scores, as well as the TabSynDex similarity score metrics. Section 6.3 will present selected visual results from the executed experiments, highlighting important characteristics and observations made from the synthetic data that has been generated. All of this provides the basis for Section 6.4, which analyses and interprets the numerical and visual results from the previous sections. This section will also revisit and answer the research questions of this thesis from Section 1.2 according to the gathered data. Lastly, this chapter will finish with Section 6.5, critically assessing the constraints and limitations of the study, taking into account the data use, experimental scope, and model interpretability.

6.1 Reproduction and Verification of Results

Given that this thesis draws heavily upon the research conducted by [Kot+22], it was deemed necessary to first reproduce the original experiments and subsequently verify the authors' findings, prior to conducting any new experiments. Fortunately, the publicly available code [Aki23] facilitated the replication of the experiments with relative ease.

The reproduction is limited to the adult dataset from Section 5.3 and to the machine learning efficacy computed with a tuned CatBoost [Pro+18] model.

Firstly, a CatBoost model was tuned on the adult dataset using the provided tuning script (tune_evaluation_model.py). Next, for each sampling algorithm, a model was trained according to the configuration files provided by the authors. Each configuration file contains the parameter settings the authors identified during hyperparameter tuning. Hence, with the models' respective pipeline.py script, the best-found model from hyperparameter tuning could be trained and saved. Finally, the trained CatBoost and trained sampling model were used in the evaluation script (eval_seeds.py), which calculates and reports the results.

Table 6.1 shows the computed F1-scores achieved by the CatBoost model when trained on different synthetic datasets generated by different sampling models. The scores that could be reproduced are almost exactly the same as those reported by the original authors. All differences are within the standard deviation reported by the authors, except for

Model	Reproduction	Original	Difference
Real	0.815	0.815	0
TVAE^{ml}	0.780	0.781	-0.001
CTABGAN^{ml}	0.775	0.783	-0.008
CTABGAN+^{ml}	0.775	0.772	+0.003
SMOTE^{ml}	0.791	0.791	0
TabDDPM$^{ml}_{q}$	0.794	0.795	-0.001

Table 6.1: Comparison of the CatBoost F1-score on synthetic datasets created by different sampling models. F1-Scores of the reproduction experiments are compared against the results reported by the original authors [Kot+22, Table 4, p. 8].

the CTABGAN model. It is unclear why the CTABGAN-model score deviates in the reproduction experiment from the original score. However, It is important to note that minor modifications to the code or different python library versions could have caused this alternation, which were required to train the model in the cloud environment, specified in Section 5.2.

Therefore, the results reported by the authors could be reproduced and verified overall.

6.2 Metric Results

Section 6.2 delves into the metric results of the conducted research.

The chapter starts with baseline experiments in Subsection 6.2.1, which provides a reference point for the subsequent modifications. Subsection 6.2.2 presents the first experimental changes with the incorporation of tabular processing. In Subsection 6.2.3, the focus shifts to the effects of changing the hyperparameters strategy. Finally, Subsection 6.2.4 details the third set of experiments, exploring to what extent a change in the normalization process affected the metric results. The chapter's objective is to highlight and compare the results of these experimental variations.

6.2.1 Baseline Experiments

After verifying that the models are able to reproduce the machine-learning efficacy scores as reported, their performance is additionally evaluated using the similarity evaluation as proposed in Subsection 4.3.2. Table 6.2 shows the complete machine learning efficacy score results for the different sampling techniques:

In addition to the machine learning efficacy scores, the similarity scores of the TabSynDex metric (compare Subsection 2.5.5) are computed (Table 6.3).

These baseline experiments show that the Diffusion based synthesis approach outperforms other models not only in terms of machine learning efficacy but also in terms of other metrics. However, Table 6.3 shows that the CTABGAN models outperform TabDDPM in the Support coverage metric. Additionally, it is worth mentioning that even

Model	Accuracy	F1	ROC-AUC
Real	0.874	0.815	0.928
TVAE^{ml}	0.845	0.780	0.900
CTABGAN^{ml}	0.850	0.775	0.900
CTABGAN+^{ml}	0.855	0.775	0.907
SMOTE^{ml}	0.858	0.791	0.910
TabDDPM$_q^{ml}$	**0.860**	**0.794**	**0.913**

Table 6.2: Machine learning efficacy (CatBoost) baseline results. The best results are highlighted in bold (excluding the real data).

Model	Similarity Score	Basic	Correlation	ML	Support	pMSE
Real	0.960	0.992	0.943	0.998	0.984	0.882
TVAE^{ml}	0.658	0.854	0.814	0.962	0.657	0.000
CTABGAN^{ml}	0.741	0.940	0.832	0.984	**0.947**	0.000
CTABGAN+^{ml}	0.750	0.969	0.882	0.990	0.892	0.019
SMOTE^{ml}	0.723	0.953	0.865	**0.992**	0.804	0.000
TabDDPM$_q^{ml}$	**0.759**	**0.973**	**0.919**	**0.992**	0.874	**0.035**

Table 6.3: Similarity baseline results using the TabSynDex metrics. The *Similarity Score* is the average of the other five scores. The best results are highlighted in bold.

though TabDDPM achieves the highest pMSE score, it is still extremely low and almost 0, which is the same for all other models. The authors of [Chu+22] confirm this observation in their experiments. During their experiments, the tested sampling techniques (various GAN-based approaches) also struggle to produce any synthetic data which achieves a pMSE score that is noticeably higher than 0. Table 6.3 indicates that this observation also holds for a diffusion-based approach, whose hyperparameters were tuned towards a machine learning efficacy score using a CatBoost model.

6.2.2 Experiment 1: Adding Tabular Processing

The first set of experiments evaluates the different tabular processing mechanisms described in Subsection 5.1.2. For this, the tabular processing mechanisms have been added to the pipeline of TabDDPM, as described in the concept in Figure 4.2. Consequently, the diffusion model's tuning (tune_ddpm.py, see Subsection 4.2.1) with the additional tabular processing was required. The models' hyperparameters have again been tuned towards the machine learning efficacy of a CatBoost model.

The results of the machine learning efficacy and TabSynDex metric results can be found in Table 6.4 and Table 6.5, respectively.

Both evaluations show that the additional BGM tabular processing increases the ML-efficacy scores. All metrics in Table 6.4 are highest for the TabDDPM-BGM model and the ML efficacy score of TabSynDex (which makes use of different machine learning

Model	Accuracy	F1	ROC-AUC
Real	0.874	0.815	0.928
TabDDPM$_q^{ml}$	0.860	0.794	0.913
TabDDPM-BGM$_q^{ml}$	**0.863**	**0.798**	**0.916**
TabDDPM-FT$_q^{ml}$	0.785	0.552	0.821

Table 6.4: CatBoost machine learning efficacy scores for different tabular processing techniques.

Model	Similarity Score	Basic	Correlation	ML	Support	pMSE
Real	0.960	0.992	0.943	0.998	0.984	0.882
TabDDPM$_q^{ml}$	**0.759**	**0.973**	**0.919**	0.992	0.874	**0.035**
TabDDPM-BGM$_q^{ml}$	0.742	0.964	0.918	**0.996**	0.831	0.000
TabDDPM-FT$_q^{ml}$	0.595	0.495	0.648	0.869	**0.963**	0.000

Table 6.5: TabSynDex evaluation metric scores for different tabular processing techniques.

models) is highest for TabDDPM-BGM as well, although only by a slight margin. Table 6.5 indicates that this increase seems to come at the cost of reduced performance in the other metrics, which are highest for the Basic-, Correlation- and pMSE-Score for the plain TabDDPM version. The performance of TabDDPM-FT lags notably behind its counterparts in terms of Correlation and Basic metric scores. Interestingly, TabDDPM-FT performance is significantly better in the Support score than the other versions and approximately 13%-points worse in terms of ML efficacy computed by the TabSynDex metric. More details can be seen in the Table 6.4, which shows that especially the F1 score from TabDDPM-FT is much worse than the F1 score of the other models. Lastly, neither the BGM nor the FT tabular processing enable the diffusion model to produce synthetic data that is able to increase the pMSE score.

6.2.3 Experiment 2: Similarity Hyperparameter optimization

The second set of experiments is very similar to the first experiment. Instead of tuning the models' hyperparameters after the machine learning efficacy, as proposed by the original authors, the models' hyperparameters are tuned after the TabSynDex similarity score.

The machine learning efficacy and TabSynDex metric results can be found in Table 6.6 and Table 6.7, respectively.

The results show that TabDDPM-BGM outperforms all other models in terms of the CatBoost machine learning efficacy and is on par with TabDDPM in the TabSynDex ML-efficacy score. TabDDPM-BGM also has the highest overall Similarity Score and achieves the highest Basic and pMSE scores. However, The simpler TabDDPM outperforms its BGM counterpart regarding Correlation and Support score. Overall, TabDDPM achieves comparable performance to TabDDPM-BGM on all metrics, with the biggest difference in

Model	Accuracy	F1	ROC-AUC
Real	0.874	0.815	0.928
TabDDPM^s_q	0.856	0.782	0.908
TabDDPM-BGM^s_q	**0.859**	**0.792**	**0.911**
TabDDPM-FT^s_q	0.767	0.450	0.712
CTABGAN^s	0.850	0.776	0.900
CTABGAN+^{s}	0.851	0.768	0.902
TVAE^s	0.845	0.780	0.900

Table 6.6: CatBoost machine learning efficacy scores for different tabular processing techniques, whose hyperparameters have been tuned towards the TabSynDex similarity score.

Model	Similarity Score	Basic	Correlation	ML	Support	pMSE
Real	0.960	0.992	0.943	0.998	0.984	0.882
TabDDPM^s_q	0.852	0.976	**0.921**	**0.991**	0.952	0.420
TabDDPM-BGM^s_q	**0.857**	**0.982**	0.858	**0.991**	0.920	**0.532**
TabDDPM-FT^s_q	0.589	0.513	0.620	0.819	**0.992**	0.000
CTABGAN^s	0.740	0.938	0.833	0.984	0.947	0.000
CTABGAN+^{s}	0.784	0.970	0.888	0.987	0.908	0.167
TVAE^s	0.658	0.856	0.815	0.962	0.656	0.000

Table 6.7: TabSynDex evaluation metric scores for different tabular processing techniques, whose hyperparameters have been tuned towards the TabSynDex similarity score.

the pMSE score of -0.11%-points. TabDDPM-FT performance is significantly worse than all other TabDDPM variants in all metrics except the support score. On the one hand, it seems to be the case that hyperparameter tuning after the similarity score does have a big influence on the TabDDPM and TabDDPM-BGM models' pMSE score. On the other hand, this hyperparameter tuning did not affect the pMSE score of the other tested models, the CTABGAN, TVAE, or the TabDDPM-FT, all having zero pMSE scores. The CTABGAN+^{s} model is the only non-diffusion model that achieves a non-zero pMSE score of 0.167%, which is still noticeably smaller than the score of TabDDPM^s_q and TabDDPM-BGM^s_q.

6.2.4 Experiment 3: Exchanging the Normalization

So far, all TabDDPM models received data normalized according to the quantile transform function (applied after tabular processing), as proposed in [Kot+22]. A third experiment will investigate how the models perform when their quantile transform function is replaced by a simpler min-max scaler, as explained in Subsection 2.1.1. Based upon the performance of the model variants so far, only the plain TabDDPM and TabDDPM-BGM, tuned after the similarity score will be investigated since the performance of TabDDPM-FT was noticeably worse.

Model	Accuracy	F1	ROC-AUC
Real	0.874	0.815	0.928
TabDDPM^s_m	0.856	0.778	**0.910**
TabDDPM-BGM^s_m	**0.857**	**0.787**	0.909
TabDDPM-BGM^s_n	0.855	0.784	0.907

Table 6.8: CatBoost Machine learning efficacy scores for different tabular processing techniques whose hyperparameters have been tuned towards the TabSynDex similarity score and the data was transformed according to a min-max-scaler.

Model	Similarity Score	Basic	Correlation	ML	Support	pMSE
Real	0.960	0.992	0.943	0.998	0.984	0.882
TabDDPM^s_m	**0.869**	0.938	**0.930**	0.990	**0.928**	**0.558**
TabDDPM-BGM^s_m	0.856	**0.981**	0.913	**0.992**	0.915	0.476
TabDDPM-BGM^s_n	0.833	0.975	0.916	**0.992**	0.915	0.369

Table 6.9: TabSynDex evaluation metric scores for different tabular processing techniques whose hyperparameters have been tuned towards the TabSynDex similarity score and the data was transformed according to a min-max-scaler.

Given the results in Table 6.8 and Table 6.9, it seems to be the case that TabDDPM-BGM outperforms TabDDPM in terms of the ML and the Basic score, but only by a slight margin. TabDDPM, on the other hand, achieves a higher overall similarity score, mainly due to the superior performance in the Correlation, Support, and pMSE-score. Additionally, not having any transformation after the tabular processing (TabDDPM-BGM^s_n) results in an overall slightly worse similarity score compared to applying a min-max transformation.

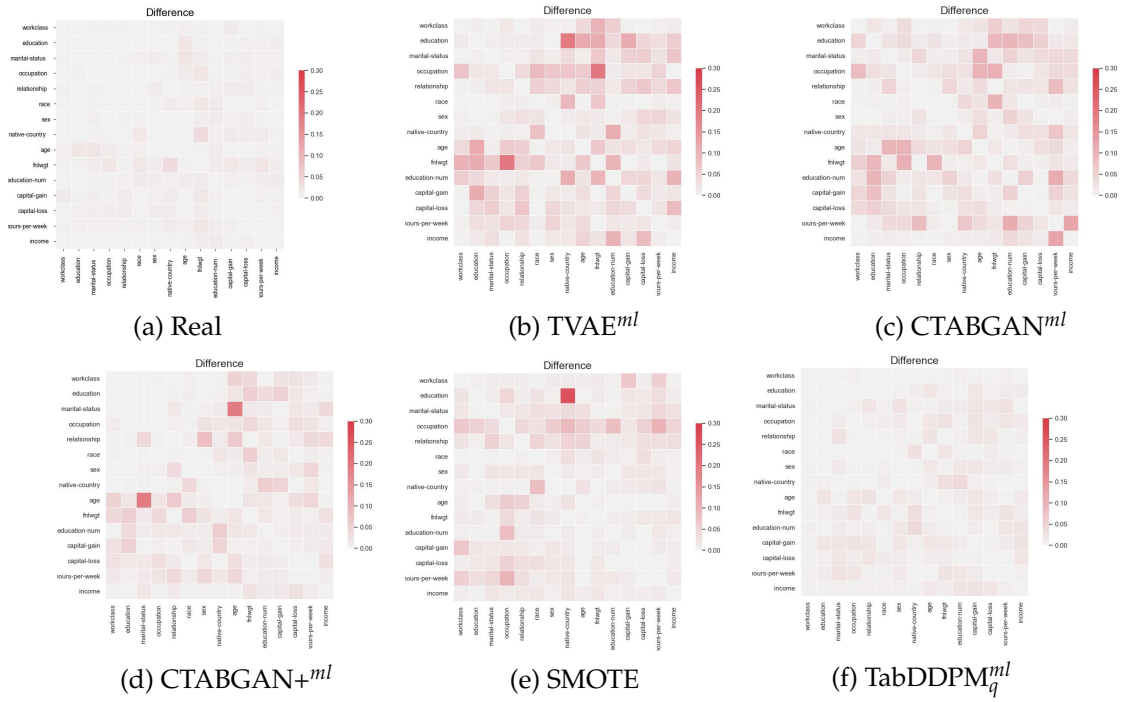
6.3 Visual Results

Several plots have been produced according to the methodology of [BHV19]. This section will cover which plots have been produced and how the plots for different model versions differ.

6.3.1 Correlation Difference Matrix

Correlation matrices have been computed to get an intuition on how well correlations between two variables within the dataset are reproduced. The correlation matrix for the synthetic dataset has been subtracted from the correlation matrix of the real dataset in order to produce a correlation difference matrix. The smaller the difference, the more similar the correlations within the synthetic dataset are to the correlations within the real dataset.

Looking at the correlations matrix differences in Figure 6.1 for the baseline models that are not diffusion-based, one can see that the CTABGAN+^{ml} andSMOTEapproaches have the smallest correlation matrix difference. However, the diffusion model TabDDPM^{ml} produces the matrix with the smallest overall differences.

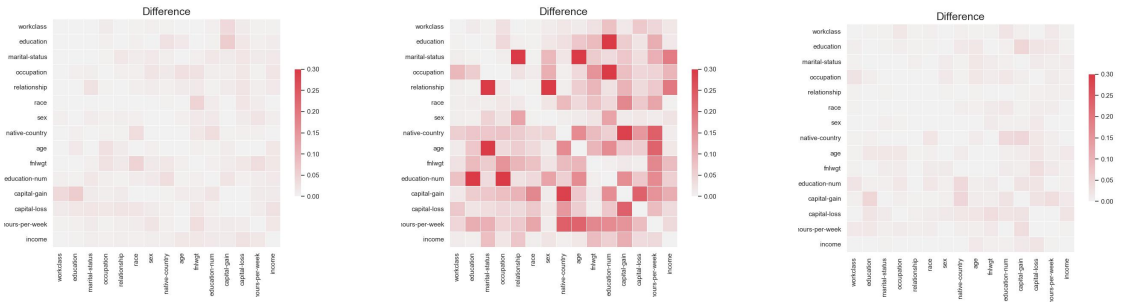


(a) Real (b) TVAE^{ml} (c) CTABGAN^{ml}

(d) CTABGAN+^{ml} (e) SMOTE (f) TabDDPM$^{ml}_{q}$

Figure 6.1: Correlation Matrix difference for Baseline models.

Figure 6.2 shows that while the TabDDPM-BGM^{ml} are able to produce small correlation differences, the TabDDPM-FT^{ml} is not able to reproduce the correlation matrix of the real dataset. The overall best correlation difference matrix is produced by TabDDPM$^{s}_{m}$, which is also reflected in their TabSynDex correlation score in Table 6.9, which is the highest across all model versions. Other model version matrices do not show any noteworthy



(a) TabDDPM-BGM$^{ml}_{q}$ (b) TabDDPM-FT$^{ml}_{q}$ (c) TabDDPM-BGM$^{s}_{m}$

Figure 6.2: Correlation Matrix difference from selected experiment models.

changes and are displayed in Figure A-9.

6.3.2 Principle Component Analysis

A PCA is a dimensionality reduction technique that converts a dataset onto principal components [BHV19]. The components are then sorted according to the amount of variance from the original dataset each component is able to capture. The following visualizations map the first two principal components, i.e. the components that capture the most variance from data.

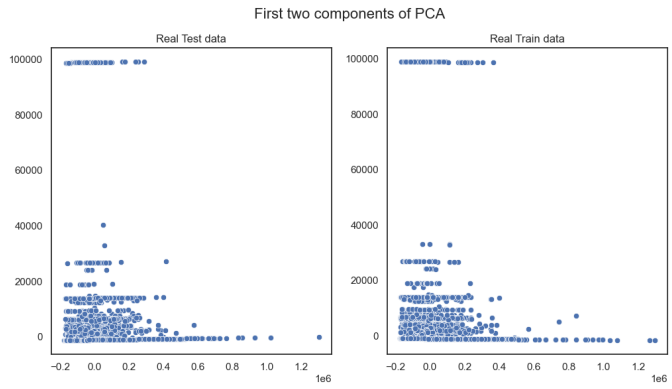
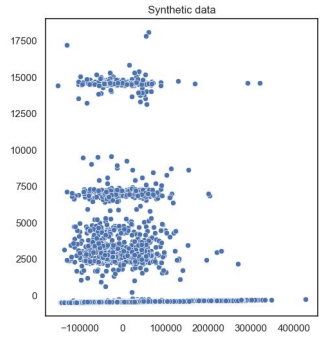
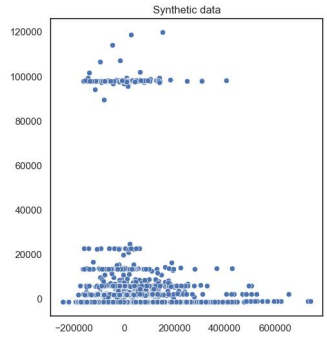


Figure 6.3: PCA for the real training and testing dataset

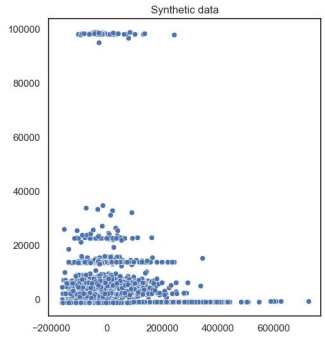
Figure 6.3 shows the two PCA plots of the first two principal components from the real training and testing dataset. One can see that the majority of values in the plots are scattered around the lower left corner (for $y <= 40000$) with two accumulations of values spreading along the x-axis around $y = 100000$ and $y = 0$. The goal of the synthetic data generation method is to produce synthetic data whose PCA plot is similar to the PCA plot of the real testing dataset.



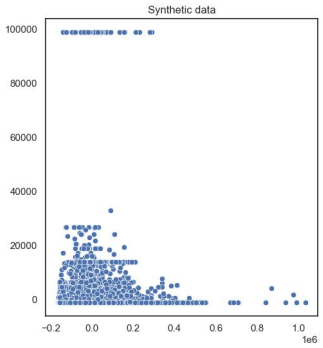
(a) TVAE^{ml}



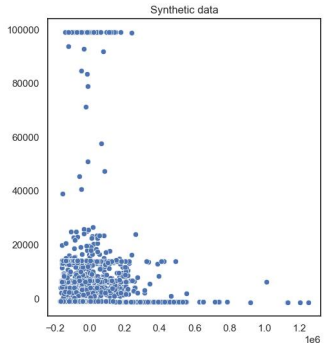
(b) CTABGAN^{ml}



(c) CTABGAN+^{ml}

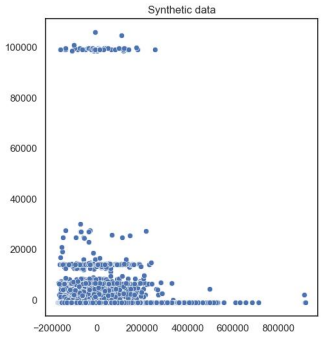


(d) SMOTE

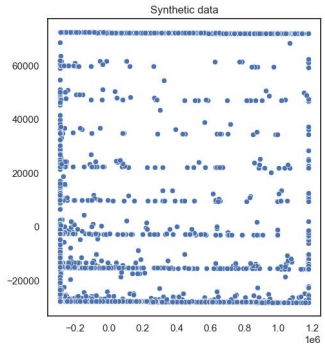


(e) TabDDPM$_q^{ml}$

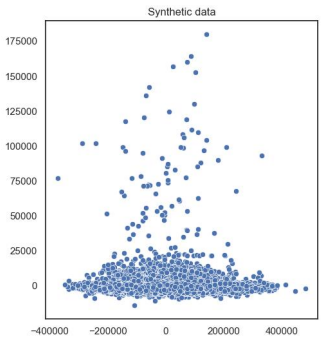
Figure 6.4: PCA for Baseline models.



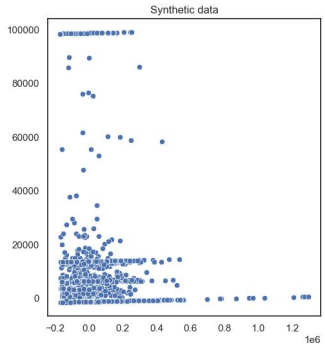
(a) TabDDPM-BGM$_q^{ml}$



(b) TabDDPM-FT$_q^{ml}$



(c) TabDDPM$_m^s$



(d) TabDDPM$_q^s$

Figure 6.5: PCA for selected experiment models.

Figure 6.4 shows the PCA plots for the different baseline models. All plots show some similarity with the PCA-plot from the original dataset, except for the TVAE model. While it seems the CTABGAN, CTABGAN+, and TabDDPM models reproduce the main characteristics of the real PCA plot, they do have a few single outliers scattered. These scatter outliers can be observed less frequently at the SMOTE PCA plot, which, out of the baseline models, recreates the original PCA plot the closest.

Out of the extended versions of TabDDPM, only the BGM tabular processing mechanism seems to be able to produce a realistic PCA-plot (compare Figure 6.5). Additionally, the min-max scaling seems to negatively affect the PCA-plot for the TabDDPM model. However, this did not occur for the TabDDPM-BGM model that used min-max scaling (Figure A-10f). This PCA-plot and the remaining model versions plots can be found in Appendix A.3.2.

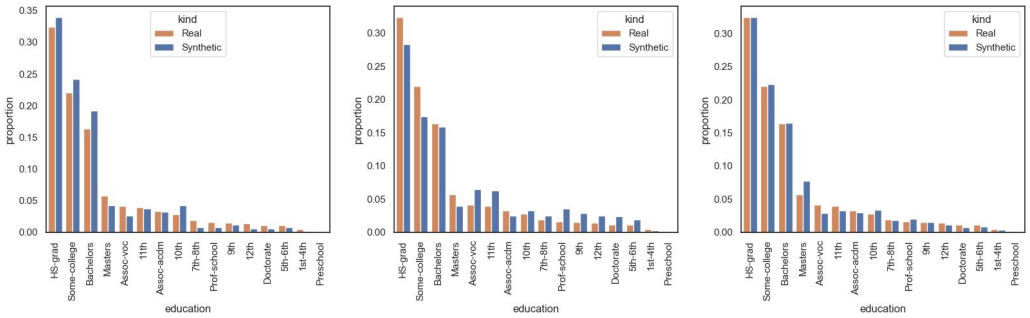
6.3.3 Distribution Plot

For each model version, distribution plots for each column can be found at Appendix A.3.3, comparing the distributions of the synthetic data with the real data. This section will highlight some interesting plots for selected models.

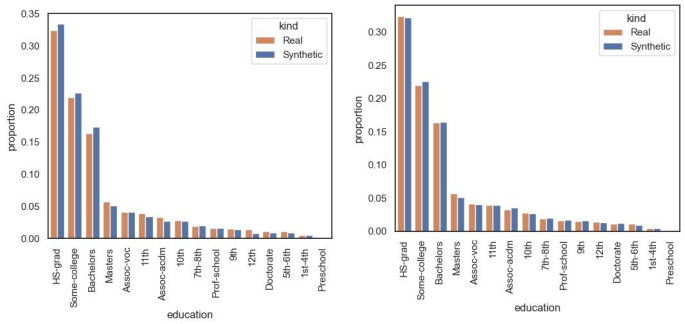
In terms of categorical distributions (Figure 6.6 displays the education column distributions as an example), all baseline models can reproduce the original distributions, with CTABGAN+^{ml} and TabDDPM$_q^{ml}$ replicating the original distribution best. The BGM tabular processing model also reproduced the categorical distributions in a comparable way to the TabDDPM counterpart. On the other hand, the FT tabular processing model could not reproduce the original distribution at all and failed to mimic the real data.

Changing the hyperparameter tuning toward the similarity score did not affect the categorical distributions in any model. The same is valid for changing to the min-max scaling since it only affects continuous values. Figure 6.7 shows that this observation does not extend towards continuous columns, given the column age as an example. Across the baseline models, CTABGAN+^{ml} and TabDDPM$_q^{ml}$ are again best able to reproduce the original data distribution of the example column. All TabDDPM-BGM models produce distributions that are comparable to those produced by the baseline TabDDPM. Again, TabDDPM-FT fails to replicate the target distributions at all. Interestingly, the similarity score hyperparameter optimization did not affect the TabDDPM distributions to any visible extent. However, the Minmax scaling reduced the TabDDPM capability to create a realistic distribution, whose probability distributions rather follows a Gaussian distribution, failing to replicate the detailed peaks of the original dataset. This behavior was not present in the BGM version with min-max scaling.

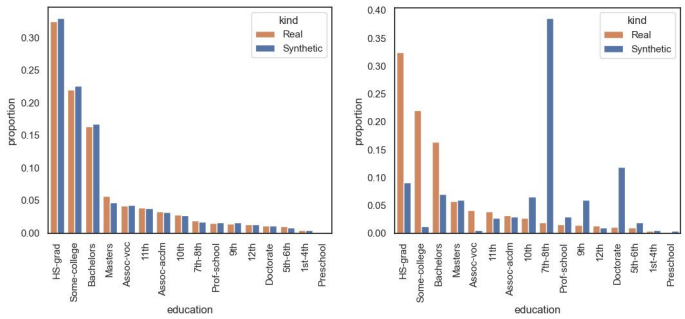
Comparing all BGM versions reveals interesting observations about the effect of applying normalizations after the tabular processing mechanisms. Having no transformation or min-max transformation results in similar plots that can capture the overall distribution but deviate in some cases quite a lot (in the area of x=17 to 28 years).



(a) TVAE^{ml} (b) CTABGAN^{ml} (c) CTABGAN+^{ml}



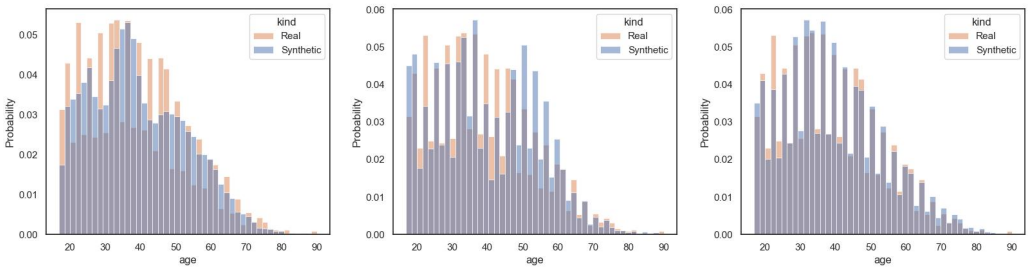
(d) SMOTE (e) TabDDPM$_q^{ml}$



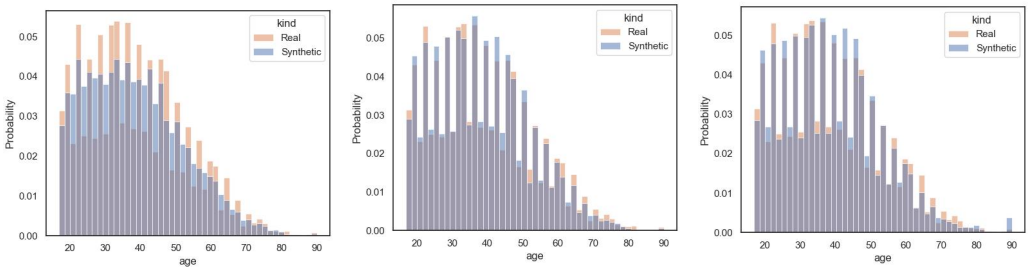
(f) TabDDPM-BGM$_q^{ml}$ (g) TabDDPM-FT$_q^{ml}$

Figure 6.6: *Education* column for baseline models.

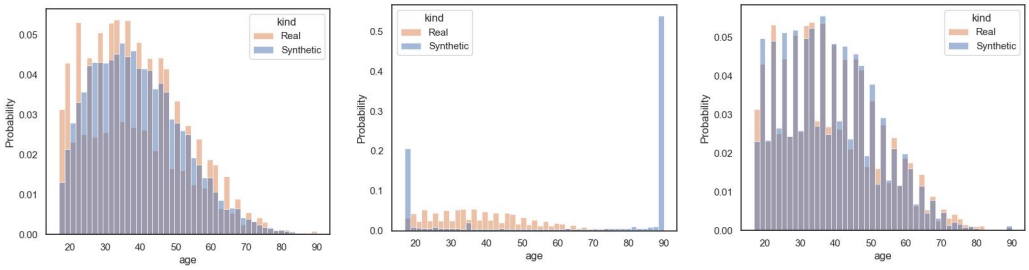
Quantile transformation (TabDDPM-BGM$_q^s$) reduces the difference in this area, but an oversampling of entries with an age of 90 occurred. Interestingly, out of all BGM variations, the model tuned after the machine learning efficacy (TabDDPM-BGM$_q^{ml}$) was best able to capture the distribution. Looking at its *age* distribution plot, it can be observed that big differences from the real data are few. Nevertheless, undersampling of certain ages is present in the synthetic data. However, one can observe that this is usually compensated with an oversampling of an age value close to it (e.g. undersampling of age x=17-18 is compensated by an oversampling of age x=19-20).



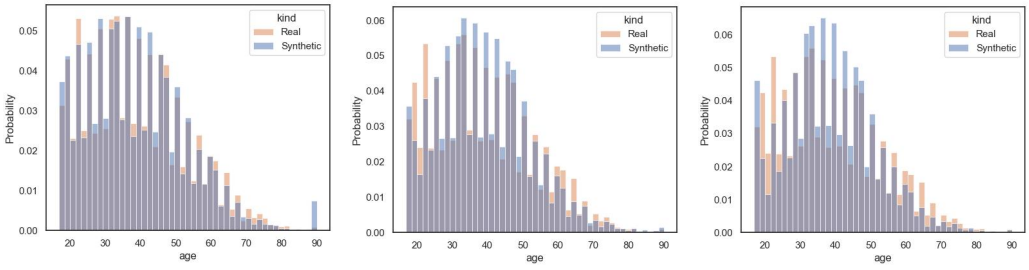
(a) TVAE^{ml} (b) CTABGAN^{ml} (c) CTABGAN+^{ml}



(d) SMOTE (e) TabDDPM$_q^{ml}$ (f) TabDDPM$_q^{s}$



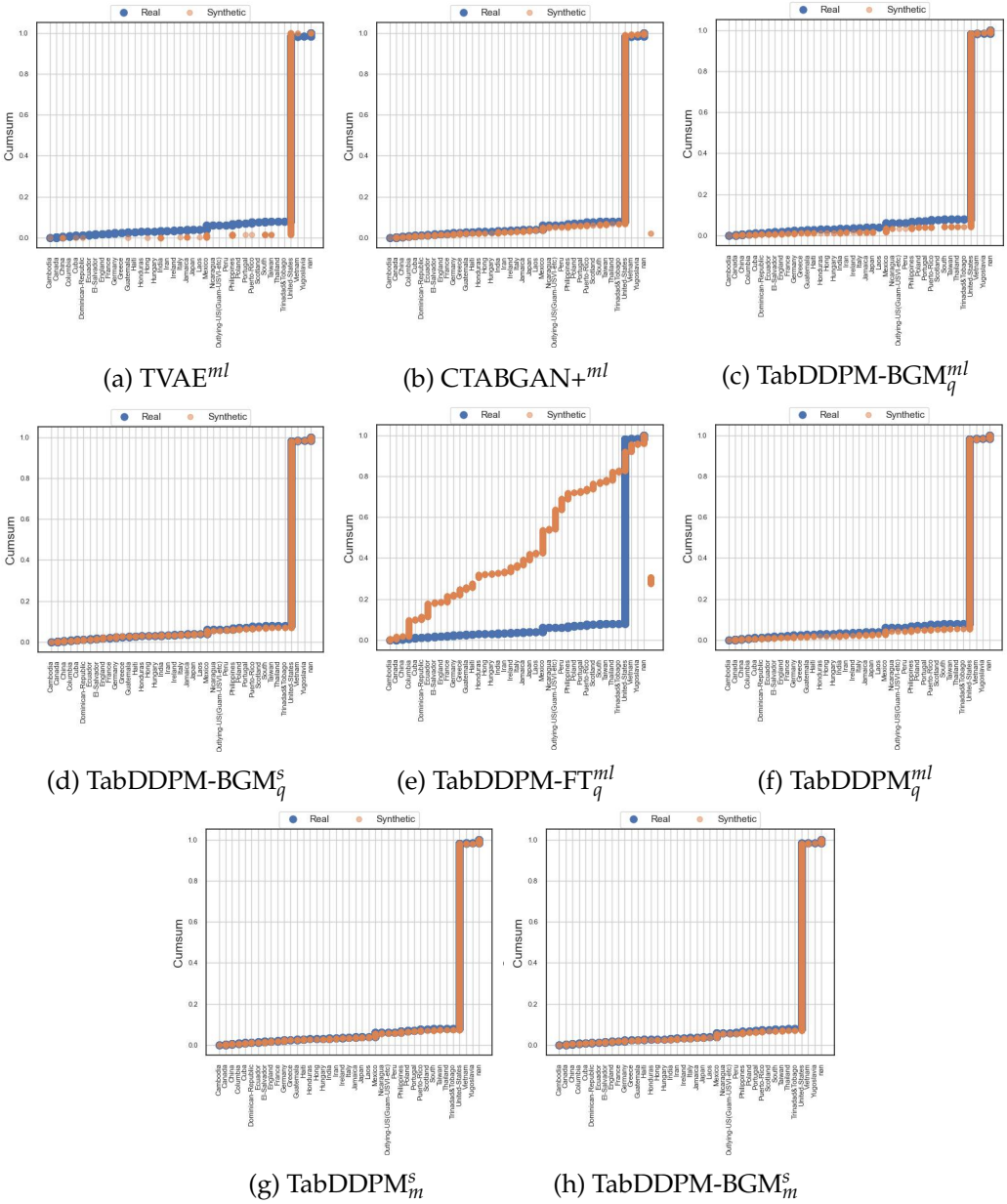
(g) TabDDPM$_m^{s}$ (h) TabDDPM-FT$_q^{ml}$ (i) TabDDPM-BGM$_q^{ml}$



(j) TabDDPM-BGM$_q^{s}$ (k) TabDDPM-BGM$_m^{s}$ (l) TabDDPM-BGM$_n^{s}$

Figure 6.7: *Age* column for selected model versions. Please note that the Y-axis range may vary due to the range of the synthetic data.

6.3.4 Cumulative Distribution Function



(a) TVAE^{ml} (b) CTABGAN+^{ml} (c) TabDDPM-BGM$^{ml}_{q}$

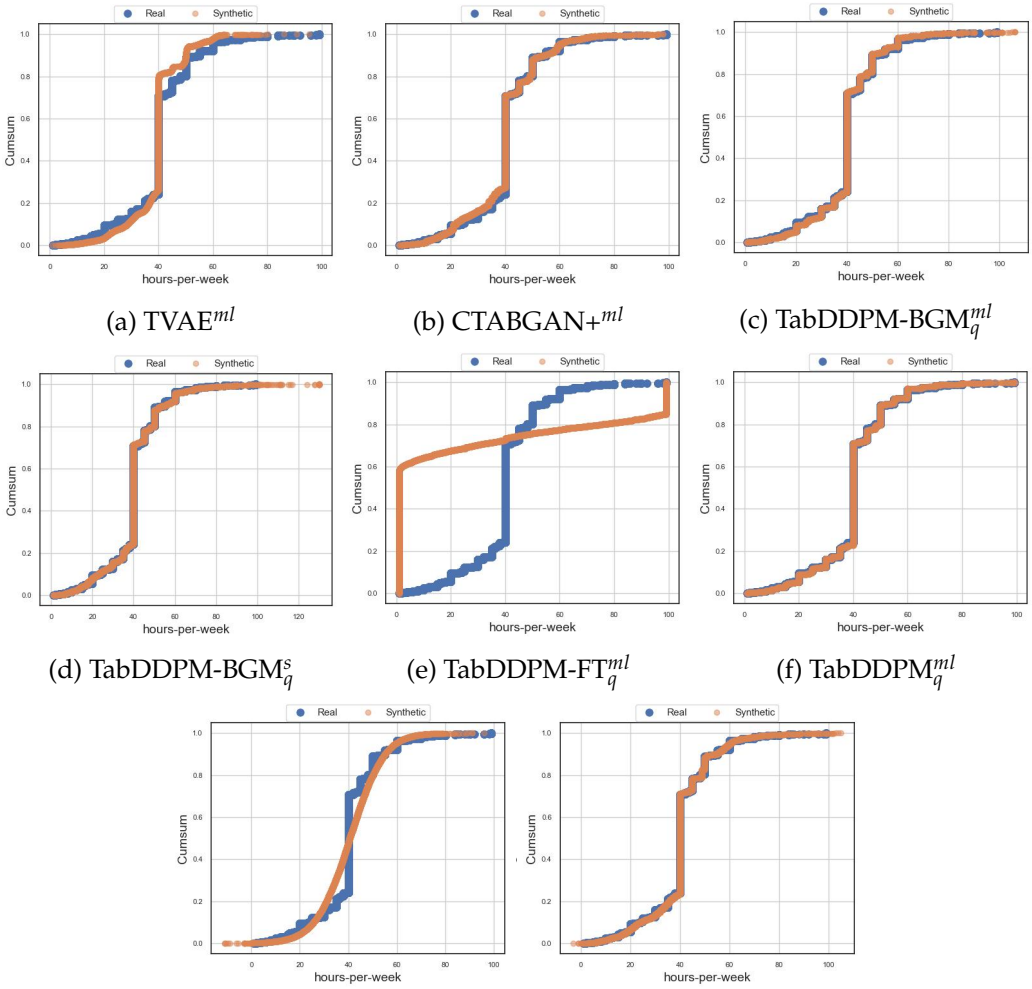
(d) TabDDPM-BGM$^{s}_{q}$ (e) TabDDPM-FT$^{ml}_{q}$ (f) TabDDPM$^{ml}_{q}$

(g) TabDDPM$^{s}_{m}$ (h) TabDDPM-BGM$^{s}_{m}$

Figure 6.8: Cumulative Density Function (CDF) plots for selected model versions for categorical the column "native-country" of the adult income dataset [DG17]

All CDF plots can be found in Appendix A.3.4. The plots show that the CDFs of the synthetic data created by CTABGAN+^{ml} and TabDDPM$^{ml}_{q}$ come closest to the CDFs of the real data across the baseline models (best visible for columns *native-country* or *hours-per-week*, compare Figures 6.8 and 6.9). The selected plots in Figures 6.8 and 6.9 show how the FT tabular processing failed again to reproduce the original datas CDF. TabDDPM-BGM, on the other hand, produced synthetic data, whose CDF correctly mimics the CDF of the real data. One could argue that TabDDPM and TabDDPM-BGM that have been tuned after the similarity score are slightly better able to reproduce CDF plots for categorical columns

whose categories are imbalanced (e.g. see column *native-country*, where the majority of values equal *United-States*). This improvement is more prominent for the BGM version. The min-max scaling greatly affected the TabDDPM version and "smoothed out" its CDFs. This phenomenon was not present in the TabDDPM-BGM version with Minmax scaling.



(a) TVAE^{ml} (b) CTABGAN+^{ml} (c) TabDDPM-BGM$_q^{ml}$

(d) TabDDPM-BGM$_q^{s}$ (e) TabDDPM-FT$_q^{ml}$ (f) TabDDPM$_q^{ml}$

(g) TabDDPM$_m^{s}$ (h) TabDDPM-BGM$_m^{s}$

Figure 6.9: CDF plots for selected model versions for the numerical column "hours-per-week" of the adult income dataset [DG17]

6.4 Discussion

The goal of this thesis was to investigate the performance of adding the tabular processing mechanisms to the TabDDPM model (RQ1) and to expand upon the already existing machine learning efficacy evaluation performed by [Kot+22] through the similarity score metrics proposed by [Chu+22].

Analysis of Visual and Numerical Results

Model	Accuracy	F1	ROC-AUC
Real	0.874	0.815	0.928
TVAE^{ml}	0.845	0.780	0.900
SMOTE	0.858	0.791	0.910
CTABGAN^{ml}	0.850	0.775	0.900
CTABGAN+^{ml}	0.855	0.775	0.907
TabDDPM$^{ml}_{q}$	0.860	0.794	0.913
TabDDPM-BGM$^{ml}_{q}$	**0.863**	**0.798**	**0.916**
TabDDPM-FT$^{ml}_{q}$	0.785	0.552	0.821
CTABGAN$^{s}_{q}$	0.850	0.776	0.900
CTABGAN+^{s}	0.851	0.768	0.902
TVAE^{s}	0.845	0.780	0.900
TabDDPM$^{s}_{q}$	0.856	0.782	0.908
TabDDPM-BGM$^{s}_{q}$	0.859	0.792	0.911
TabDDPM-FT$^{s}_{q}$	0.766	0.451	0.712
TabDDPM$^{s}_{m}$	0.856	0.778	0.910
TabDDPM-BGM$^{s}_{m}$	0.857	0.787	0.909
TabDDPM-BGM$^{s}_{n}$	0.855	0.784	0.907

Table 6.10: Overview of all CatBoost machine learning efficacy results for all tested model versions.

Based on the obtained results, it is evident that the processing mechanism for FT has failed and is not advisable. Despite having the highest overall support score, all other measurements and visual representations indicate that the model could not generate synthetic data that closely resembles the original dataset. The reason why the FT processing does not work well even though it performed well in the cell imputation task [ZC22] remains unclear. As a consequence, the TabDDPM-FT variations will not be further discussed or analyzed.

The BGM processor variant was able to produce not only good numerical but also visual results. In terms of machine learning efficacy, the TabDDPM-BGM$^{ml}_{q}$ version shows the best machine learning efficacy performance across all tested metrics (Table 6.10: Accuracy, F1 and AUC-ROC and Table 6.11: ML). However, the advantage over its simpler counterpart TabDDPM$^{ml}_{q}$ is only marginal (+0.003%-points). Overall, models with hyperparameters tuned towards machine learning efficacy outperform models with hyperparameters tuned

Model	Similarity Score	Basic	Correlation	ML	Support	pMSE
Real	0.960	0.992	0.943	0.998	0.984	0.882
TVAE^{ml}	0.658	0.854	0.814	0.962	0.657	0.000
SMOTE	0.723	0.953	0.865	0.992	0.804	0.000
CTABGAN^{ml}	0.741	0.940	0.832	0.984	0.947	0.000
CTABGAN+^{ml}	0.750	0.969	0.882	0.990	0.892	0.019
TabDDPM$^{ml}_{q}$	0.759	0.973	0.919	0.992	0.874	0.035
TabDDPM-BGM$^{ml}_{q}$	0.742	0.964	0.918	**0.996**	0.831	0.000
TabDDPM-FT$^{ml}_{q}$	0.595	0.495	0.648	0.869	0.963	0.000
CTABGAN^{s}	0.740	0.938	0.833	0.984	0.947	0.000
CTABGAN+^{s}	0.784	0.970	0.888	0.987	0.908	0.167
TVAE^{s}	0.658	0.856	0.815	0.962	0.656	0.000
TabDDPM$^{s}_{q}$	0.852	0.976	0.921	0.991	0.952	0.420
TabDDPM-BGM$^{s}_{q}$	0.857	**0.982**	0.858	0.991	0.920	0.532
TabDDPM-FT$^{s}_{q}$	0.588	0.512	0.619	0.818	**0.992**	0.000
TabDDPM$^{s}_{m}$	**0.869**	0.938	**0.930**	0.990	0.928	**0.558**
TabDDPM-BGM$^{s}_{m}$	0.856	0.981	0.913	0.992	0.915	0.476
TabDDPM-BGM$^{s}_{n}$	0.833	0.975	0.916	0.992	0.915	0.369

Table 6.11: Overview of all TabSynDex score results for all tested model versions.

towards similarity score in the machine learning efficacy scores, as one would expect. The results of the non-diffusion baseline models show that theSMOTEapproach is a reliable sampling technique when the synthetic data will be used in a machine-learning scenario. Its machine learning efficacy results are superior to the other non-diffusion baseline models, while the model is overall less complex. For other scenarios, CTABGAN+ achieves the best overall performance in terms of numerical and visual results for non-diffusion-based models. Nevertheless, the best-found CTABGAN+ version is inferior to the best-found diffusion model.

Changing the tuning strategy to the similarity score allowed several observations to be made. Firstly for the non-diffusion models', metric results had not significantly changed after switching to similarity score tuning. All of their metric scores stayed roughly the same, with the exception of CTABGAN+. Only for the this model, a similarity tuning resulted in a noteworthy metric increase, especially for its pMSE score, which increased by +0.14%-points.

Diffusion models, on the other hand, showed more noteworthy changes in their numerical metric results. Overall, diffusion models with a similarity score tuning show reduced machine learning efficacy metric scores compared to their machine learning efficacy counterparts. While this reduction is relatively small, the similarity metrics have increased significantly. Specifically, through the adjustment of the hyperparameters towards the similarity score, which includes the PMSE score, an increase in the pMSE score was notably achieved. Especially for diffusion models, this big increase in the pMSE score is worth highlighting. For Models such as TabDDPM and TabDDPM-BGM, this similarity tuning

resulted in pMSE values increasing to a score between 0.42 and 0.55. The same tuning strategy did not affect the pMSE score of baseline models like TVAE or CTABGAN; only CTABGAN+ was affected but at a lower amplitude compared to diffusion models. These glspmse scores indicate, that almost all non-diffusion baseline models failed to produce data that is non-differentiable from the real data by a logistic regression model. This finding aligns with the observation made by Chundawat et al. [Chu+22], who also observe that all of their tested models (mainly GAN-based) achieve a pMSE of 0. The experiments of Chundawat et al. [Chu+22] only identified one model, a GAN model with gradient penalty and Wasserstein distance, which was able to achieve a peak pMSE score of 0.4 (scores are not comparable, since a different dataset was used). Overall, this indicates that diffusion-based models are better able to produce synthetic data that is harder to differentiate from its real counterpart.

In addition to the increase of the pMSE score, the other metrics that make up the similarity score have also increased, but on a smaller scale. Only correlation values got decreased slightly through the similarity tuning. This observation holds for TabDDPM and TabDDPM-BGM but not for the baseline models (TVAE and CTABGAN), whose metric results remained unchanged after switching the tuning strategy. This indicates that the importance of hyperparameter tuning is much higher for diffusion models in tabular data synthesis, as their tuning strategy affects the metric outcomes significantly.

Replacing the quantile transform function with min-max scaling for the TabDDPM (TabDDPM^s_m) has led to the highest overall similarity score. It scored highest in both correlation and pMSE, leading to a high overall similarity score. According to [dev23c], the quantile transform function is a non-linear transformation that may distort correlations between variables. This is also observable in the correlation metric score, where TabDDPM^s_q and TabDDPM-BGM^s_q have a lower correlation score than their min-max scaling counterpart. However, it is essential to also take into consideration the visual results. Although the correlation difference plot of TabDDPM^s_m appears satisfactory, the distribution and CDF plot of continuous columns (compare Figures A-14c and A-20c) look worse when compared to those of other models. The continuous plots of TabDDPM^s_m lack details and rather approximate the original distribution. The model's quantile transform counterpart TabDDPM^s_q does not show this behavior, producing overall more realistic plots. This indicates that the min-max scaling, which is applied to the numerical columns, causes this kind of behavior. Since min-max scaling scales the data according to the minimum and maximum value with the column, it is susceptible to outliers. This results in a bad normalization if even one value in the column is unusually large or small. The BGM processor also makes use of a form of min-max encoding in their "general transform" [Zha+22, p. 7] (compare Subsection 5.1.2). However, Zhao et al. [Zha+22] emphasizes as well that this encoding should only be applied to simple continuous distributions like a single-mode Gaussian distribution. TabDDPM-BGM^s_m does not show this behavior even though min-max scaling is applied. A possible explanation for why the TabDDPM-BGM

variants are not affected by a min-max scaling in the same way as TabDDPM is related to the mode-specific normalization. Recall that mode-specific normalization transforms numerical columns into a vector consisting of an α and a β part. The α part is the normalized value according to the most likely mode, which is indicated in the One-Hot vector β. The *TabularProcessor* splits the α and β parts up and assigns them to the numerical and categorical output arrays respectively. This was required since the diffusion for numerical and categorical values is different (Gaussian diffusion for numerical and Multinomial diffusion for categorical). Only after this, the normalization through min-max is applied (and only to the numerical values). But since all the α values are already normalized, the impact of an additional normalization through the min-max function is most likely reduced.

In summary, the following observations could be made:

- The processing mechanism for FT has failed and is not advisable, as it could not generate synthetic data that closely resembles the original dataset.
- The BGM processor produced good numerical and visual results.
- TabDDPM-BGM$_q^{ml}$ shows the best machine learning efficacy performance across all tested models.
- Out of the baseline models,SMOTEis a reliable sampling technique for non-diffusion baseline models in machine learning scenarios. In other scenarios, CTABGAN+^{s} is the preferred non-diffusion model.
- Diffusion models outperform non-diffusion models in tabular data synthesis.
- Diffusion models with similarity score tuning led to models with a non-zero pMSE score.
- Hyperparameter tuning is more important for diffusion models in tabular data synthesis than for non-diffusion models, as their tuning strategy significantly affects metric outcomes.
- Replacing the quantile transform function with min-max scaling for TabDDPM (TabDDPM$_m^s$) led to the highest overall similarity score but also resulted in worse distributions and CDF plots of continuous columns.
- TabDDPM-BGM is not affected by quantile or min-max scaling to the same extent as the plain TabDDPM.

Analysis of Research Questions

RQ1

The results obtained in this study provide insights into the research questions posed in Section 1.2. RQ1 is concerned with the effects of additional tabular data processing mechanisms on TabDDPM. By incorporating additional tabular data processing mechanisms, the TabDDPM model improved performance compared to the baseline models. However, this was only the case for the BGM processing but not the FT processing. It was found that the TabDDPM-BGM$^{ml}_{q}$ is the best-suited model for producing synthetic data that will be used in a machine-learning scenario. For other scenarios, a TabDDPM-BGM variation tuned towards the similarity score should be preferred. Even though TabDDPM$^{s}_{m}$ achieved the highest overall similarity score, its visual results lack detail for numerical columns. Hence when using TabDDPM-BGM, TabDDPM-BGM$^{s}_{q}$ should be preferred when slightly better Basic, Support, and pMSE scores are desired and TabDDPM-BGM$^{s}_{m}$ when good correlations are of importance.

The visual results reinforced these findings, with correlation differences and PCA plots produced by TabDDPM-BGM variants closely resembling the original dataset. In addition to that, TabDDPM-BGM versions are reliably able to reproduce distributions and CDFs of the original dataset. The most significant difference was observable when replacing the quantile transformation with the min-max scaling. While the plain TabDDPM$^{s}_{m}$ model struggled to reproduce the detail in the original continuous columns, the BGM processing variant was able to.

To summarize, the tabular processor encodes the data into a different data format, which will be processed by the TabDDPM afterward. Depending on the encoded data format, the generative capability of TabDDPM changed. From the FT processed data, the TabDDPM model could not generate realistic synthetic data. On the other hand, the BGM technique encoded the data reliably, such that the diffusion model could produce synthetic data that closely resembled the original dataset in multiple aspects. With the BGM technique, the TabDDPM showed slightly improved generative capabilities, better performance across various metrics, and more realistic synthetic data generation compared to TabDDPM without BGM. Hence, the results suggest that introducing a tabular processor to the synthesis pipeline can improve the overall quality of the produced synthetic data.

RQ2

RQ2 is concerned with an extended evaluation of Diffusion based tabular data synthesis that goes beyond a machine learning efficacy evaluation. In this thesis, the TabSynDex evaluation metric [Chu+22], in addition to a CatBoost machine learning efficacy, was applied. This evaluation revealed new insights into the performance of diffusion-based models compared to other model types in tabular data synthesis. Overall, it could be observed that the superior performance of TabDDPM compared to the non-diffusion models found in [Kot+22] could not only be confirmed with the results of this thesis but it could be shown that the superior performance also extends to the other tested metrics (basic statistics, correlations, ml, support, pMSE).

Furthermore, the evaluation also highlights the importance of hyperparameter tuning for diffusion models in tabular data synthesis. While a change in the hyperparameter tuning strategy hardly affected non-diffusion models, it greatly affected the TabDDPM approach. If hyperparameters are tuned towards the TabSynDex Similarity score, diffusion models outperform all other models, including VAE and GAN-based approaches, in all tested metrics. The most significant effect was visible in the pMSE score, indicating the differentiability of the synthetic data from the real data. Non-diffusion models consistently produced synthetic data that resulted in a near-zero pMSE score, even if their hyperparameters have been tuned to maximize it. However, diffusion model variants tested in this thesis were able to produce pMSE scores of up to 0.55%.

The evaluation strategy in this thesis also included analysis of visualizations of the synthetic data in multiple ways. This evaluation revealed that TabDDPM and its variances produce synthetic data whose plots resemble the original data plots more closely than the synthetic data plots of non-diffusion models. Additionally, the visual inspection revealed the effects of changing the hyperparameter tuning strategy or data preprocessing strategy has on diffusion models. For example, TabDDPM$^{s}_{m}$ metrics result suggest that it produces the best model overall to produce synthetic data, as it has the highest overall similarity score. However, its plots show a reduced capability to recreate continuous columns accurately.

In summary, the evaluation performed in the thesis showed that TabDDPM and its BGM variant outperform other generative models, including VAE and GAN-based approaches, in an extended similarity evaluation based on the TabSynDex metric, machine learning efficacy, and a visualization analysis. Hyperparameter tuning greatly affected the performance of diffusion models, and visualizations revealed that diffusion models produce synthetic data that more closely resemble the original data than non-diffusion models. Ultimately, this thesis showed the importance of an elaborate evaluation of synthetic data, as new insights on the behavior of diffusion models for tabular data synthesis could be gained, which are of benefit for the development of potential future diffusion-based models. Additionally, such an extended evaluation allows for a holistic view of the produced synthetic data, uncovering strengths and weaknesses from different perspectives.

6.5 Limitations

This study has some limitations that need to be acknowledged. The most significant limitation is that only one real dataset (the adult dataset [DG17]) has been used to evaluate the performance of different models. Therefore, it is possible that some model versions may perform better or worse on other datasets with different characteristics such as size, dimensionality, or complexity. The reason for this was the limited access to computing resources. Consequently, all observations and results made in this thesis only hold for the adult dataset and should not be generalized until they can be confirmed on multiple datasets.

A second limitation is that only a fixed set of hyperparameters were tested during hyperparameter tuning. Therefore, it is possible that some methods may benefit from further optimization or customization.

Furthermore, although the set of evaluation metrics has been larger compared to the original authors [Kot+22], an additional, even more extensive evaluation should be considered. In particular, this experiment did not consider privacy- and model-complexity-related metrics that would likely reveal new valuable insights.

Lastly, diffusion models, like most other deep neural networks, suffer from a lack of interpretability, as they are considered a black-box model [BCR97].

7 Conclusion and Future Work

7.1 Conclusion

This thesis investigates the effect of different tabular processing mechanisms on diffusion-based tabular data synthesis. An already existing tabular data generation pipeline has been extended to implement different tabular encoding and decoding strategies easily. The pipeline of TabDDPM [Kot+22] was extended by two tabular processing mechanisms that encode the tabular data before the training and revert the data back after sampling synthetic data. The extended pipeline was evaluated using a CatBoost-based machine learning efficacy evaluation, a TabSynDex metric, and visualization of the synthetic data. The TabSynDex metric comprises five sub-metrics that capture different similarity aspects of the synthetic data. Several observations could be made in a set of experiments. Firstly, the results in terms of machine learning efficacy stated by the authors of the TabDDPM pipeline could be reproduced [Kot+22]. Thanks to the addition of the TabSynDex metrics, it has been shown that TabDDPM superior performance against a set of non-diffusion baseline models also extends to the TabSynDex similarity metrics.

Adding a Feature Tokenization (FT) processing mechanism into the TabDDPM generation pipeline transforms the tabular data in such a way, that the diffusion model was not able to generate any meaningful data, which could be seen in the visual result. Adopting a more complex encoding/decoding strategy from [Zha+22], which uses a Bayesian Gaussian Mixture Model (BGM), improved TabDDPM's generative capabilities in producing synthetic data that is useful in machine-learning scenarios. Hence, it could be shown that changing the tabular data format greatly affects the diffusion model's generative capability, in a negative (for the FT approach) and positive way (for the BGM approach).

Furthermore, the importance of hyperparameter tuning for diffusion models was highlighted. Diffusion models tuned towards the TabSynDex similarity score greatly improved several of the TabSynDex metrics. In the performed experiments, diffusion models have been the first models to achieve a non-zero pMSE score, which other current state-of-art models have not been capable of. Hence, diffusion models are better at producing synthetic data that is more non-differentiable from the real data [Chu+22] compared to tested non-diffusion models. Further experiments showed how changing the data normalization strategy can, on the one hand, increase metric results but, on the other hand, worsen visual results. This insight underlines the importance of a visual evaluation of synthetic data to estimate the quality of the produced synthetic data properly. From all performed experiments, it can be concluded that diffusion-based tabular data synthesis

is superior to the synthesis of other model types, such as GANs or VAEs. Moreover, diffusion models have not only shown overall better metric and visual results, but they are also more flexible, as a hyperparameter tuning already affects the outcome of the metrics greatly. Even though TabDDPM$^{s}_{m}$ achieved the highest overall similarity score, it is not able to reproduce the underlying characteristics of the real data, which could be seen in the several of the comparative visualizations presented in this thesis. TabDDPM$^{s}_{m}$ lacks the ability to properly recreate important characteristics of the real data, such as column distributions (Figure 6.7) or the real datasets principal components (Figure 6.5). This was not the case for TabDDPM-BGM$^{s}_{[q|m]}$ models, which achieved only slightly worse similarity scores but better capture the underlying characteristics of the real data, as indicated by the visual plots in this thesis. As a consequence, the TabDDPM-BGM$^{s}_{[q|m]}$ variants should be considered as the best overall models of this thesis. Nevertheless, potential users should alter the model pipeline design depending on the use case of the synthetic data, since TabDDPM-BGM$^{s}_{m}$ produced the best correlation score, TabDDPM-BGM$^{ml}_{q}$ the best machine learning efficacy scores and TabDDPM-BGM$^{s}_{q}$ the best basic statistical values and pMSE results.

7.2 Future Work

As implementing diffusion models for synthesizing tabular data represents a relatively innovative approach with limited research to date, numerous promising pathways for future investigations exist. Firstly, to overcome the limitations presented in Section 6.5, a more extensive set of datasets is required to confirm the results observed in this thesis. More datasets with diverse characteristics to generalize the findings and identify potential sources of variation or bias.

Moreover, incorporating a variety of metrics with diverse perspectives is essential for evaluating the performance and trade-offs of distinct tabular data generation methods. This is particularly relevant for privacy-related metrics, as recent research suggests that diffusion models may not effectively protect privacy and allow to extract training data samples [Car+23].

Furthermore, as the definition of synthetic tabular data generation in Subsection 2.2.2 stated, the synthetic data T_{syn} is usually created by training the generator on T_{train} and comparing T_{syn} with T_{test}. The quality of T_{syn} naturally depends on the quality of T_{train} and, most importantly, how representative the T_{train}-split is from the overall joint distribution (the combination of T_{train} and T_{test}). Hence, future research could investigate the impact different train-test-splits potentially have on the quality of T_{syn}. This could be achieved by testing different train-test-splits of the same dataset and comparing the quality of the generated T_{syn} across the different splits. Additionally, most evaluation techniques heavily rely on the comparison of T_{syn} with T_{test}. In this context, the comparison of T_{syn} with T_{train} might provide valuable insights as well.

Moreover, it is likely that diffusion models can be further improved through several changes. First, the hyperparameter search space could be broader, such that more hyperparameter variations are explored. Second, the neural network itself could be replaced by a more sophisticated one. The proposed network by [Kot+22], which predicts the noise in the reverse process, follows a simple architecture. In the image generation domain, a more complex U-Net [RFB15] architecture with attention mechanisms at different layers was proposed [DN21]. [DN21] showed that such changes led to a better generation quality. Hence, it is possible that similar changes to the model's architecture would lead to better generative capabilities. Other changes could be made towards further improving the loss function. It might be worth investigating whether some adversarial loss could be implemented, similar to GAN models. For example, an adversarial network could be tasked to tell apart which noise (the noise that will be removed from the noised image) was predicted by the neural network and which was the actual noise. This information could be used to further enhance the capability of the diffusion network to remove noise from x_t to get to x_{t-1}. Lastly, depending on the results of a detailed privacy evaluation, introducing differential privacy [Dwo11] into the diffusion model might be worth investigating, which has been done extensively in the GAN domain [JYS18; Xin+20; Kun+21; TFR22].

Regarding the tabular processor mechanism, several changes and different techniques could be implemented to improve the overall generation pipeline. Currently, the tabular processor mechanism is fully separated from the diffusion processes. Fully incorporating the tabular processor into the neural network architecture for some mechanisms would allow a gradient flow through the processing mechanism. This would be especially interesting for embedding mechanisms, as they could learn in this way to create a meaningful representation of the data. Alternatively, a pre-training mechanism for the tabular processor could be possible so that embedding layers can learn meaningful representations before the actual tabular data synthesis starts.

There exist several different tabular processor techniques that could be worth investigating. For example, the models TABBIE [Iid+21] and TURL [Den+21] show different ways to create meaningful-context aware representations of tabular data. Future researchers could adapt their work and use the pretrained embeddings created by these models to encode the tabular data before processing it in the diffusion pipeline. One of the biggest challenges here is to correctly revert the data back into its original data format after it has been synthesized. A possible solution for this could be training a decoder module, as in [Rom+22], adapting an encoder-decoder architecture. This decoder could be trained explicitly towards reverting the previously encoded data back into its human-readable format. In this solution, the tabular processing inverse function is not implemented directly by the developer, but it is learned through a neural network.

As one can see, many possible research directions exist for diffusion-based tabular data synthesis. Future research should first focus on removing the possible limitations of this thesis through a more elaborate experimental setup with more datasets. Here, potential privacy issues are required to be investigated. Depending on the results, architectural changes and additional tabular processing mechanisms are worth investigating.

References

[Aba+16] Martin Abadi et al. “Deep Learning with Differential Privacy”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. New York, NY, USA: Association for Computing Machinery, Oct. 24, 2016, pp. 308–318. ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978318.

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein Generative Adversarial Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, July 17, 2017, pp. 214–223.

[ACS17] Moustafa Alzantot, Supriyo Chakraborty, and Mani Srivastava. “SenseGen: A Deep Learning Architecture for Synthetic Sensor Data Generation”. In: *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 2017 IEEE International Conference on Pervasive Computing and Communications: Workshops (PerCom Workshops). Kona, HI: IEEE, Mar. 2017, pp. 188–193. ISBN: 978-1-5090-4338-5. DOI: 10.1109/PERCOMW.2017.7917555.

[Agg18] Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Cham, Switzerland: Springer, 2018. 497 pp. ISBN: 978-3-319-94462-3 978-3-030-06856-1. DOI: 10.1007/978-3-319-94463-0.

[Ago+23] Andrea Agostinelli et al. *MusicLM: Generating Music From Text*. Jan. 26, 2023. arXiv: 2301.11325 [cs, eess]. URL: http://arxiv.org/abs/2301.11325 (visited on 02/13/2023). preprint.

[Aki+19] Takuya Akiba et al. “Optuna: A next-Generation Hyperparameter Optimization Framework”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[Aki23] Akim. *TabDDPM: Modelling Tabular Data with Diffusion Models*. Mar. 5, 2023.

[BC13] Jonathan M. Borwein and Richard E. Crandall. “Closed Forms: What They Are and Why We Care”. In: *Notices of the American Mathematical Society* 60.01 (Jan. 1, 2013), p. 50. ISSN: 0002-9920, 1088-9477. DOI: 10.1090/noti936.

[BCB15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[BCK13] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. 3rd ed. SEI Series in Software Engineering. Upper Saddle River, NJ: Addison-Wesley, 2013. 589 pp. ISBN: 978-0-321-81573-6.

[BCR97] J.M. Benitez, J.L. Castro, and I. Requena. "Are Artificial Neural Networks Black Boxes?" In: *IEEE Transactions on Neural Networks* 8.5 (Sept. 1997), pp. 1156–1164. ISSN: 1941-0093. DOI: 10.1109/72.623216.

[BHV19] Bauke Brenninkmeijer, Youri Hille, and Arjen Vries. "On the Generation and Evaluation of Tabular Data Using GANs". MA thesis. Nijmegen, Netherlands: Radboud University, Oct. 14, 2019. 71 pp.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. New York: Springer, 2006. 738 pp. ISBN: 978-0-387-31073-2.

[Bis98] Christopher M Bishop. "Latent Variable Models." In: *Learning in graphical models* 371 (1998).

[BK21] Manfred Broy and Marco Kuhrmann. *Einführung in die Softwaretechnik*. Xpert.press. Berlin, Heidelberg: Springer Berlin Heidelberg, 2021. ISBN: 978-3-662-50262-4 978-3-662-50263-1. DOI: 10.1007/978-3-662-50263-1.

[BKG20] Dor Bank, Noam Koenigstein, and Raja Giryes. "Autoencoders". In: *Deep Learning in Science*. Vol. abs/2003.05991. Cambridge: Cambridge University Press, 2020. ISBN: 978-1-108-84535-9.

[Bor+22] Vadim Borisov et al. "Deep Neural Networks and Tabular Data: A Survey". In: *IEEE Transactions on Neural Networks and Learning Systems* (2022), pp. 1–21. ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2022.3229161.

[Bou+21] Stavroula Bourou et al. "A Review of Tabular Data Synthesis Using GANs on an IDS Dataset". In: *Information* 12.9 (Sept. 14, 2021), p. 375. ISSN: 2078-2489. DOI: 10.3390/info12090375.

[Cap22] Hernandez Capel. "Master Thesis : Denoising Diffusion Probabilistic Models Applied to Energy Forecasting in Power Systems". MA thesis. Liege, Belgium: University of Liège, 2022. 75 pp.

[Car+23] Nicholas Carlini et al. *Extracting Training Data from Diffusion Models*. Version 1. Jan. 30, 2023. arXiv: 2301.13188 [cs]. URL: http://arxiv.org/abs/2301.13188 (visited on 03/24/2023). preprint.

[Cha+02] Nitesh V. Chawla et al. “SMOTE: Synthetic Minority over-Sampling Technique”. In: *Journal of Artificial Intelligence Research* 16.1 (June 1, 2002), pp. 321–357. ISSN: 1076-9757.

[Cho+14] Kyunghyun Cho et al. “On the Properties of Neural Machine Translation: Encoder–Decoder Approaches”. In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. SSST 2014. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 103–111. DOI: 10.3115/v1/W14-4012.

[Cho+17] Edward Choi et al. “Generating Multi-label Discrete Patient Records Using Generative Adversarial Networks”. In: *Proceedings of the 2nd Machine Learning for Healthcare Conference*. Machine Learning for Healthcare Conference. PMLR, Nov. 6, 2017, pp. 286–305.

[Cho20] Kaushik Choudhury. *Feature Scaling — Effect Of Different Scikit-Learn Scalers: Deep Dive*. Medium. Aug. 19, 2020. URL: https://towardsdatascience.com/feature-scaling-effect-of-different-scikit-learn-scalers-deep-dive-8dec775d4946 (visited on 03/26/2023).

[Chu+22] Vikram S Chundawat et al. “A Universal Metric for Robust Evaluation of Synthetic Tabular Data”. In: *IEEE Transactions on Artificial Intelligence* (2022), pp. 1–11. ISSN: 2691-4581. DOI: 10.1109/TAI.2022.3229289.

[Chu22] Vikram Singh Chundawat. *A Universal Metric for Robust Evaluation of Synthetic Tabular Data*. Dec. 11, 2022.

[Cod70] E. F. Codd. “A Relational Model of Data for Large Shared Data Banks”. In: *Communications of the ACM* 13.6 (June 1970), pp. 377–387. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/362384.362685.

[DC19] Jessamyn Dahmen and Diane Cook. “SynSys: A Synthetic Data Generation System for Healthcare Applications”. In: *Sensors* 19.5 (Mar. 8, 2019), p. 1181. ISSN: 1424-8220. DOI: 10.3390/s19051181.

[dCar+17] María del Carmen Rodríguez-Hernández et al. “DataGenCARS: A Generator of Synthetic Data for the Evaluation of Context-Aware Recommendation Systems”. In: *Pervasive and Mobile Computing* 38 (July 2017), pp. 516–541. ISSN: 15741192. DOI: 10.1016/j.pmcj.2016.09.020.

[Den+21] Xiang Deng et al. “TURL: Table Understanding through Representation Learning”. In: *Proceedings of the VLDB Endowment* 14.3 (Dec. 9, 2021), pp. 307–319. ISSN: 2150-8097. DOI: 10.14778/3430915.3430921.

[Dev+19] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7,*

2019, Volume 1 (Long and Short Papers). 2019, pp. 4171–4186. DOI: 10.18653/v1/n19-1423.

[dev23a] Scikit-learn developers. *Bayesian Gaussian Mixture*. scikit-learn. 2023. URL: https://scikit-learn/stable/modules/generated/sklearn.mixture.BayesianGaussianMixture.html (visited on 03/09/2023).

[dev23b] Scikit-learn developers. *Preprocessing Data*. 2023. URL: https://scikit-learn/stable/modules/preprocessing.html (visited on 02/13/2023).

[dev23c] Scikit-learn developers. *QuantileTransformer*. scikit-learn. 2023. URL: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.QuantileTransformer.html (visited on 03/22/2023).

[DG17] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017.

[DKS95] James Dougherty, Ron Kohavi, and Mehran Sahami. "Supervised and Unsupervised Discretization of Continuous Features". In: *Machine Learning Proceedings 1995*. Ed. by Armand Prieditis and Stuart Russell. San Francisco (CA): Morgan Kaufmann, Jan. 1, 1995, pp. 194–202. ISBN: 978-1-55860-377-6. DOI: 10.1016/B978-1-55860-377-6.50032-3.

[DN21] Prafulla Dhariwal and Alex Nichol. *Diffusion Models Beat GANs on Image Synthesis*. June 1, 2021. arXiv: 2105.05233 [cs, stat]. URL: http://arxiv.org/abs/2105.05233 (visited on 10/12/2022). preprint.

[DS22] Yi Dong and Emanuel Scoullos. *Generating Synthetic Data with Transformers: A Solution for Enterprise Data Challenges*. NVIDIA Technical Blog. Sept. 5, 2022. URL: https://developer.nvidia.com/blog/generating-synthetic-data-with-transformers-a-solution-for-enterprise-data-challenges/ (visited on 02/07/2023).

[Dwo08] Cynthia Dwork. "Differential Privacy: A Survey of Results". In: *Theory and Applications of Models of Computation*. Ed. by Manindra Agrawal et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 1–19. ISBN: 978-3-540-79228-4. DOI: 10.1007/978-3-540-79228-4_1.

[Dwo11] Cynthia Dwork. "Differential Privacy". In: *Encyclopedia of Cryptography and Security*. Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Boston, MA: Springer US, 2011, pp. 338–340. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_752.

[EHR17] Cristóbal Esteban, Stephanie L. Hyland, and Gunnar Rätsch. *Real-Valued (Medical) Time Series Generation with Recurrent Conditional GANs*. Dec. 3, 2017. DOI: 10.48550/arXiv.1706.02633. arXiv: 1706.02633 [cs, stat].

[El 20] Khaled El Emam. "Seven Ways to Evaluate the Utility of Synthetic Data". In: *IEEE Security & Privacy* 18.4 (July 2020), pp. 56–59. ISSN: 1540-7993, 1558-4046. DOI: 10.1109/MSEC.2020.2992821.

[Esm+22] Mohammad Esmaeilpour et al. “Bi-Discriminator GAN for Tabular Data Synthesis”. In: *Pattern Recognition Letters* 159 (July 2022), pp. 204–210. ISSN: 01678655. DOI: 10.1016/j.patrec.2022.05.023.

[Eur16] European Commission. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the Protection of Natural Persons with Regard to the Processing of Personal Data and on the Free Movement of Such Data, and Repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA Relevance)*. 2016.

[Fac23a] Hugging Face. *Community Examples*. GitHub. 2023. URL: https://github.com/huggingface/diffusers (visited on 03/02/2023).

[Fac23b] Hugging Face. *Diffusers Pipelines*. Github. Feb. 3, 2023. URL: https://github.com/huggingface/diffusers (visited on 03/02/2023).

[Fac23c] Hugging Face. *Hugging Face – The AI Community Building the Future.* Feb. 25, 2023. URL: https://huggingface.co/ (visited on 03/02/2023).

[Fan+20] Ju Fan et al. *Relational Data Synthesis Using Generative Adversarial Networks: A Design Space Exploration*. Aug. 28, 2020. DOI: 10.48550/arXiv.2008.12763. arXiv: 2008.12763 [cs].

[FM22] Seth* Forsgren and Hayk* Martiros. *Riffusion - Stable Diffusion for Real-Time Music Generation*. 2022.

[Fro+15] Charlie Frogner et al. *Learning with a Wasserstein Loss*. Dec. 29, 2015. arXiv: 1506.05439 [cs, stat]. URL: http://arxiv.org/abs/1506.05439 (visited on 01/09/2023). preprint.

[FVH12] Elena Fitkov-Norris, Samireh Vahid, and Chris Hand. “Evaluating the Impact of Categorical Data Encoding and Scaling on Neural Network Classification Performance: The Case of Repeat Consumption of Identical Cultural Goods”. In: *Engineering Applications of Neural Networks*. Ed. by Chrisina Jayne, Shigang Yue, and Lazaros Iliadis. Communications in Computer and Information Science. Berlin, Heidelberg: Springer, 2012, pp. 343–352. ISBN: 978-3-642-32909-8. DOI: 10.1007/978-3-642-32909-8_35.

[Gag17] Paul A. Gagniuc. *Markov Chains: From Theory to Implementation and Experimentation*. Hoboken, NJ: John Wiley & Sons, 2017. 1 p. ISBN: 978-1-119-38757-2 978-1-119-38758-9.

[Gam+94] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st ed. Addison-Wesley Professional, 1994. ISBN: 0-201-63361-2.

[Gar+16] Salvador García et al. “Big Data Preprocessing: Methods and Prospects”. In: *Big Data Analytics* 1.1 (Nov. 1, 2016), p. 9. ISSN: 2058-6345. DOI: 10.1186/s41044-016-0014-0.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. ISBN: 978-0-262-03561-3.

[Ge+21] Chang Ge et al. "Kamino: Constraint-Aware Differentially Private Data Synthesis". In: *Proceedings of the VLDB Endowment* 14.10 (June 2021), pp. 1886–1899. ISSN: 2150-8097. DOI: 10.14778/3467861.3467876.

[Gil+20] Anthony Gillioz et al. "Overview of the Transformer-based Models for NLP Tasks". In: *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*. 2020 15th Conference on Computer Science and Information Systems (FedCSIS). Sept. 2020, pp. 179–183. DOI: 10.15439/2020F20.

[Gon+20] Andre Goncalves et al. "Generation and Evaluation of Synthetic Patient Data". In: *BMC Medical Research Methodology* 20.1 (Dec. 2020), p. 108. ISSN: 1471-2288. DOI: 10.1186/s12874-020-00977-1.

[Gon+22] Yuan Gong et al. "SSAST: Self-Supervised Audio Spectrogram Transformer". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.10 (10 June 28, 2022), pp. 10699–10709. ISSN: 2374-3468. DOI: 10.1609/aaai.v36i10.21315.

[Goo+14] Ian Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014.

[Goo+20] Ian Goodfellow et al. "Generative Adversarial Networks". In: *Communications of the ACM* 63.11 (Oct. 22, 2020), pp. 139–144. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3422622.

[Gor+21] Yury Gorishniy et al. "Revisiting Deep Learning Models for Tabular Data". In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 18932–18943.

[Gor23] Yura Gorishniy. *On Embeddings for Numerical Features in Tabular Deep Learning*. Feb. 28, 2023.

[GR92] Andrew Gelman and Donald B. Rubin. "Inference from Iterative Simulation Using Multiple Sequences". In: *Statistical Science* 7.4 (Nov. 1992), pp. 457–472. ISSN: 0883-4237, 2168-8745. DOI: 10.1214/ss/1177011136.

[GRB22] Yury Gorishniy, Ivan Rubachev, and Artem Babenko. *On Embeddings for Numerical Features in Tabular Deep Learning*. Mar. 15, 2022. arXiv: 2203.05556 [cs]. URL: http://arxiv.org/abs/2203.05556 (visited on 01/30/2023). preprint.

[Gul+17] Ishaan Gulrajani et al. "Improved Training of Wasserstein GANs". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017.

[Guo+22] Meng-Hao Guo et al. “Attention Mechanisms in Computer Vision: A Survey”. In: *Computational Visual Media* 8.3 (Sept. 2022), pp. 331–368. ISSN: 2096-0433, 2096-0662. DOI: 10.1007/s41095-022-0271-y.

[Hao+17] Mariem Haoues et al. “A Guideline for Software Architecture Selection Based on ISO 25010 Quality Related Characteristics”. In: *International Journal of System Assurance Engineering and Management* 8.S2 (Nov. 2017), pp. 886–909. ISSN: 0975-6809, 0976-4348. DOI: 10.1007/s13198-016-0546-8.

[Har+20] Charles R. Harris et al. “Array Programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[Has19] Muneeb ul Hassan. *ResNet (34, 50, 101): Residual CNNs for Image Classification Tasks*. Jan. 23, 2019. URL: https://neurohive.io/en/popular-networks/resnet/ (visited on 04/17/2023).

[He+16] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90.

[Her+22] Mikel Hernandez et al. “Synthetic Data Generation for Tabular Health Records: A Systematic Review”. In: *Neurocomputing* 493 (July 2022), pp. 28–45. ISSN: 09252312. DOI: 10.1016/j.neucom.2022.04.053.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. Dec. 16, 2020. arXiv: 2006.11239 [cs, stat]. URL: http://arxiv.org/abs/2006.11239 (visited on 01/09/2023). preprint.

[Ho+22] Jonathan Ho et al. *Video Diffusion Models*. June 22, 2022. arXiv: 2204.03458 [cs]. URL: http://arxiv.org/abs/2204.03458 (visited on 02/13/2023). preprint.

[Hoo+21] Emiel Hoogeboom et al. “Argmax Flows and Multinomial Diffusion: Learning Categorical Distributions”. In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 12454–12465.

[HS22] Jonathan Ho and Tim Salimans. *Classifier-Free Diffusion Guidance*. July 25, 2022. arXiv: 2207.12598 [cs]. URL: http://arxiv.org/abs/2207.12598 (visited on 01/10/2023). preprint.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1, 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer Feedforward Networks Are Universal Approximators". In: *Neural Networks* 2.5 (Jan. 1989), pp. 359–366. ISSN: 08936080. DOI: 10.1016/0893-6080(89)90020-8.

[Hua+20] Xin Huang et al. *TabTransformer: Tabular Data Modeling Using Contextual Embeddings*. Dec. 11, 2020. DOI: 10.48550/arXiv.2012.06678. arXiv: 2012.06678 [cs]. URL: http://arxiv.org/abs/2012.06678 (visited on 02/20/2023). preprint.

[Iid+21] Hiroshi Iida et al. "TABBIE: Pretrained Representations of Tabular Data". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. NAACL-HLT 2021. Online: Association for Computational Linguistics, June 2021, pp. 3446–3456. DOI: 10.18653/v1/2021.naacl-main.270.

[ISO] ISO/IEC. *Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models*. standard ISO/IEC 25010:2011. ISO.

[Izo+22] Ivan Izonin et al. "A Two-Step Data Normalization Approach for Improving Classification Accuracy in the Medical Diagnosis Domain". In: *Mathematics* 10.11 (11 Jan. 2022), p. 1942. ISSN: 2227-7390. DOI: 10.3390/math10111942.

[JYS18] James Jordon, Jinsung Yoon, and M. Schaar. "PATE-GAN: Generating Synthetic Data with Differential Privacy Guarantees". In: International Conference on Learning Representations. Sept. 27, 2018.

[KAD21] Sanket Kamthe, Samuel Assefa, and Marc Deisenroth. *Copula Flows for Synthetic Data Generation*. Jan. 3, 2021. arXiv: 2101.00598 [cs, stat]. URL: http://arxiv.org/abs/2101.00598 (visited on 03/02/2023). preprint.

[Kal+20] Ioannis Kaloskampis et al. "Synthetic Data in the Civil Service". In: *Significance* 17.6 (2020), pp. 18–23. ISSN: 1740-9713. DOI: 10.1111/1740-9713.01466.

[Kha+22] Salman Khan et al. "Transformers in Vision: A Survey". In: *ACM Computing Surveys* 54 (10s Sept. 13, 2022), 200:1–200:41. ISSN: 0360-0300. DOI: 10.1145/3505244.

[Kim+21] Jayoung Kim et al. "OCT-GAN: Neural ODE-based Conditional Tabular GANs". In: *Proceedings of the Web Conference 2021*. WWW '21: The Web Conference 2021. Ljubljana Slovenia: ACM, Apr. 19, 2021, pp. 1506–1515. ISBN: 978-1-4503-8312-7. DOI: 10.1145/3442381.3449999.

[Koh95] Ron Kohavi. "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection". In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Aug. 20, 1995, pp. 1137–1143. ISBN: 978-1-55860-363-9.

[Koh96] Ron Kohavi. "Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid". In: Knowledge Discovery and Data Mining. Aug. 2, 1996.

[Kot+22] Akim Kotelnikov et al. *TabDDPM: Modelling Tabular Data with Diffusion Models*. Sept. 30, 2022. arXiv: 2209.15421 [cs]. URL: http://arxiv.org/abs/2209.15421 (visited on 10/11/2022). preprint.

[Kun+21] Aditya Kunar et al. *DTGAN: Differential Private Training for Tabular GANs*. Aug. 2, 2021. DOI: 10.48550/arXiv.2107.02521. arXiv: 2107.02521 [cs].

[KW13] Diederik P. Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. arXiv: 1312.6114 [cs, stat]. URL: http://arxiv.org/abs/1312.6114 (visited on 01/09/2023). preprint.

[KW19] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (Nov. 27, 2019), pp. 307–392. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000056.

[KWT22] Peter Kowalczyk, Giacomo Welsch, and Frédéric Thiesse. "Towards a Taxonomy for the Use of Synthetic Data in Advanced Analytics". Version 1. In: (2022). DOI: 10.48550/ARXIV.2212.02622.

[Lan03] David M. Lane. *Introduction to Statistics*. 2003.

[Lec+98] Y. Lecun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791.

[LHB22] Gael Lederrey, Tim Hillel, and Michel Bierlaire. *DATGAN: Integrating Expert Knowledge into Deep Learning for Synthetic Tabular Data*. Mar. 7, 2022. DOI: 10.48550/arXiv.2203.03489. arXiv: 2203.03489 [cs].

[Li+22] Xiaomin Li et al. *TTS-GAN: A Transformer-based Time-Series Generative Adversarial Network*. June 26, 2022. arXiv: 2202.02691 [cs]. URL: http://arxiv.org/abs/2202.02691 (visited on 06/30/2022). preprint.

[Lin+22] Tianyang Lin et al. "A Survey of Transformers". In: *AI Open* 3 (Jan. 1, 2022), pp. 111–132. ISSN: 2666-6510. DOI: 10.1016/j.aiopen.2022.10.001.

[Lit+21] Claire Little et al. *Generative Adversarial Networks for Synthetic Data Generation: A Comparative Study*. Dec. 3, 2021. DOI: 10.48550/arXiv.2112.01925. arXiv: 2112.01925 [cs]. URL: http://arxiv.org/abs/2112.01925 (visited on 07/10/2022). preprint.

[LL21] Khanh-Hoi Le Minh and Kim-Hung Le. "AirGen: GAN-based Synthetic Data Generator for Air Monitoring in Smart City". In: *2021 IEEE 6th International Forum on Research and Technology for Society and Industry (RTSI)*. 2021 IEEE 6th International Forum on Research and Technology for Society and Industry (RTSI). Naples, Italy: IEEE, Sept. 6, 2021, pp. 317–322. ISBN: 978-1-66544-135-3. DOI: 10.1109/RTSI50628.2021.9597364.

[Lu+21] Kevin Lu et al. *Pretrained Transformers as Universal Computation Engines*. June 30, 2021. DOI: 10.48550/arXiv.2103.05247. arXiv: 2103.05247 [cs]. URL: http://arxiv.org/abs/2103.05247 (visited on 02/20/2023). preprint.

[Luo22] Calvin Luo. *Understanding Diffusion Models: A Unified Perspective*. Aug. 25, 2022. arXiv: 2208.11970 [cs]. URL: http://arxiv.org/abs/2208.11970 (visited on 04/24/2023). preprint.

[Mic23] Microsoft. *Cloud Computing Services | Microsoft Azure*. 2023. URL: https://azure.microsoft.com/de-de/ (visited on 03/29/2023).

[Mik+13] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and Their Compositionality". In: *Advances in Neural Information Processing Systems*. Vol. 26. Curran Associates, Inc., 2013.

[MLA18] Alejandro Mottini, Alix Lheritier, and Rodrigo Acuna-Agost. *Airline Passenger Name Record Generation Using Generative Adversarial Networks*. July 17, 2018. DOI: 10.48550/arXiv.1807.06657. arXiv: 1807.06657 [cs, stat]. URL: http://arxiv.org/abs/1807.06657 (visited on 07/16/2022). preprint.

[MMS22] Anega Maheshwari, Priyanka Mitra, and Bhavna Sharma. "Autoencoder: Issues, Challenges and Future Prospect". In: *Recent Innovations in Mechanical Engineering*. Ed. by Meghanshu Vashista et al. Lecture Notes in Mechanical Engineering. Singapore: Springer, 2022, pp. 257–266. ISBN: 9789811692369. DOI: 10.1007/978-981-16-9236-9_24.

[MO14] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. Nov. 6, 2014. arXiv: 1411.1784 [cs, stat]. URL: http://arxiv.org/abs/1411.1784 (visited on 01/09/2023). preprint.

[MWT20] Susan McKeever, Manhar Singh Walia, and Brendan Tierney. "Synthesising Tabular Datasets Using Wasserstein Conditional GANS with Gradient Penalty (WCGAN-GP)". In: vol. 2771. CEUR Workshop Proceedings. Technological University Dublin, 2020, pp. 325–336. DOI: 10.21427/E6WA-SZ92.

[ND21] Alex Nichol and Prafulla Dhariwal. *Improved Denoising Diffusion Probabilistic Models*. Feb. 18, 2021. arXiv: 2102.09672 [cs, stat]. URL: http://arxiv.org/abs/2102.09672 (visited on 01/10/2023). preprint.

[NZY21] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. "A Review on the Attention Mechanism of Deep Learning". In: *Neurocomputing* 452 (Sept. 2021), pp. 48–62. ISSN: 09252312. DOI: 10.1016/j.neucom.2021.03.091.

[Ope22] OpenAI. *ChatGPT: Optimizing Language Models for Dialogue*. OpenAI. Nov. 30, 2022. URL: https://openai.com/blog/chatgpt/ (visited on 02/13/2023).

[Out22] Outlier, director. *Diffusion Models | Paper Explanation | Math Explained*. June 6, 2022.

[Pad+21] Inkit Padhi et al. "Tabular Transformers for Modeling Multivariate Time Series". In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). Toronto, ON, Canada: IEEE, June 6, 2021, pp. 3565–3569. ISBN: 978-1-72817-605-5. DOI: 10.1109/ICASSP39728.2021.9414142.

[Pan22] Evgeniya Panova. *How Synthetic Data Enhances Enterprise Data Strategy*. May 12, 2022. URL: https://www.anonos.com/how-synthetic-data-enhances-enterprise-data-strategy (visited on 03/26/2023).

[Pap+17] Nicolas Papernot et al. *Semi-Supervised Knowledge Transfer for Deep Learning from Private Training Data*. Mar. 3, 2017. arXiv: 1610.05755 [cs, stat]. URL: http://arxiv.org/abs/1610.05755 (visited on 06/06/2023). preprint.

[Par+18] Noseong Park et al. "Data Synthesis Based on Generative Adversarial Networks". In: *Proceedings of the VLDB Endowment* 11.10 (June 1, 2018), pp. 1071–1083. ISSN: 2150-8097. DOI: 10.14778/3231751.3231757.

[Par21] Chris Parsons. *What's a Machine Learning Model?* NVIDIA Blog. Aug. 16, 2021. URL: https://blogs.nvidia.com/blog/2021/08/16/what-is-a-machine-learning-model/ (visited on 03/30/2023).

[Ped+11] F. Pedregosa et al. "Scikit-Learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[Pil+22] José Pilaluisa et al. "Contextual Word Embeddings for Tabular Data Search and Integration". In: *Neural Computing and Applications* (Nov. 30, 2022). ISSN: 1433-3058. DOI: 10.1007/s00521-022-08066-8.

[Pla18] E. Plaut. *From Principal Subspaces to Principal Components with Linear Autoencoders*. Apr. 26, 2018. URL: https://www.semanticscholar.org/paper/From-Principal-Subspaces-to-Principal-Components-Plaut/3d4dc879166e902ede476a079684b9f5b5143e13 (visited on 02/16/2023). preprint.

[Pre21] Tom Preston-Werner. *TOML: Tom's Obvious Minimal Language*. Version 1.0.0. Jan. 12, 2021.

[Pro+18] Liudmila Prokhorenkova et al. "CatBoost: Unbiased Boosting with Categorical Features". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., Dec. 3, 2018, pp. 6639–6649.

[PyT23] PyTorch. *Embedding — PyTorch 1.13 Documentation*. 2023. URL: https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html (visited on 03/10/2023).

[Rad+18] Alec Radford et al. *Improving Language Understanding by Generative Pre-Training*. 2018. preprint.

[Rad17] Pranoy Radhakrishnan. *Introduction to Recurrent Neural Network*. Towards Data Science. Aug. 20, 2017. URL: https://towardsdatascience.com/introduction-to-recurrent-neural-network-27202c3945f3 (visited on 04/17/2023).

[Raz+22] Mina Razghandi et al. *Variational Autoencoder Generative Adversarial Network for Synthetic Data Generation in Smart Home*. Jan. 18, 2022. DOI: 10.48550/arXiv.2201.07387. arXiv: 2201.07387 [cs, eess].

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Vol. 9351. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24573-7 978-3-319-24574-4. DOI: 10.1007/978-3-319-24574-4_28.

[RHW86a] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, Jan. 3, 1986, pp. 318–362. ISBN: 978-0-262-68053-0.

[RHW86b] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-Propagating Errors". In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/323533a0.

[Roc19] Joseph Rocca. *Understanding Variational Autoencoders (VAEs)*. Medium. Sept. 24, 2019. URL: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73 (visited on 06/13/2023).

[Rom+22] Robin Rombach et al. “High-Resolution Image Synthesis with Latent Diffusion Models”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). New Orleans, LA, USA: IEEE, June 2022, pp. 10674–10685. ISBN: 978-1-66546-946-3. DOI: 10.1109/CVPR52688.2022.01042.

[Ros21] Adrian Rosebrock. *Convolutional Neural Networks (CNNs) and Layer Types*. PyImageSearch. May 14, 2021. URL: https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/ (visited on 04/17/2023).

[Ros58] F. Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.” In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 1939-1471, 0033-295X. DOI: 10.1037/h0042519.

[Run] Runway. *Gen-1 by Runway*. Runway. URL: https://research.runwayml.com/gen1 (visited on 02/13/2023).

[Sno+18] Joshua Snoke et al. “General and Specific Utility Measures for Synthetic Data”. In: *Journal of the Royal Statistical Society. Series A: Statistics in Society* 181.3 (June 2018), pp. 663–688. ISSN: 0964-1998. DOI: 10.1111/rssa.12358.

[Soh+15] Jascha Sohl-Dickstein et al. *Deep Unsupervised Learning Using Nonequilibrium Thermodynamics*. Nov. 18, 2015. arXiv: 1503.03585 [cond-mat, q-bio, stat]. URL: http://arxiv.org/abs/1503.03585 (visited on 01/09/2023). preprint.

[Som+21] Gowthami Somepalli et al. *SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training*. June 2, 2021. arXiv: 2106.01342 [cs, stat]. URL: http://arxiv.org/abs/2106.01342 (visited on 01/31/2023). preprint.

[SR07] Anders Skrondal and Sophia Rabe-Hesketh. “Latent Variable Modelling: A Survey”. In: *Scandinavian Journal of Statistics* 34.4 (Dec. 5, 2007), pp. 712–745. ISSN: 03036898. DOI: 10.1111/j.1467-9469.2007.00573.x.

[Sun+19] Baohua Sun et al. “Supertml: Two-dimensional Word Embedding for the Precognition on Structured Tabular Data”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2019, pp. 2973–2981.

[Tas+21] Yusuke Tashiro et al. “CSDI: Conditional Score-based Diffusion Models for Probabilistic Time Series Imputation”. In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 24804–24816.

[TFR22] Amirsina Torfi, Edward A. Fox, and Chandan K. Reddy. "Differentially Private Synthetic Medical Data Generation Using Convolutional GANs". In: *Information Sciences* 586 (Mar. 2022), pp. 485–500. ISSN: 00200255. DOI: 10.1016/j.ins.2021.12.018.

[Vas+17] Ashish Vaswani et al. "Attention Is All You Need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., Dec. 4, 2017, pp. 6000–6010. ISBN: 978-1-5108-6096-4.

[VB19] Andreas Vogelsang and Markus Borg. *Requirements Engineering for Machine Learning: Perspectives from Data Scientists*. Aug. 13, 2019. arXiv: 1908.04674 [cs]. URL: http://arxiv.org/abs/1908.04674 (visited on 03/06/2023). preprint.

[VD95] Guido Van Rossum and Fred L Drake Jr. *Python Reference Manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[vMSF22] vikancha-MSFT. *NC-series - Azure Virtual Machines*. Dec. 21, 2022. URL: https://learn.microsoft.com/en-us/azure/virtual-machines/nc-series (visited on 03/13/2023).

[W E09] David W. Eembley. "Relational Model". In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M. Tamer Özsu. Boston, MA: Springer US, 2009, pp. 2372–2376. ISBN: 978-0-387-35544-3 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_306.

[Wen18] Lilian Weng. *From Autoencoder to Beta-VAE*. Dec. 8, 2018. URL: https://lilianweng.github.io/posts/2018-08-12-vae/ (visited on 01/16/2023).

[Wen21] Lilian Weng. *What Are Diffusion Models?* July 11, 2021. URL: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/ (visited on 01/16/2023).

[Wes10] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[WHV08] Mark A. Whiting, Jereme Haack, and Carrie Varley. "Creating Realistic, Scenario-Based Synthetic Data for Test and Evaluation of Information Analytics Software". In: *Proceedings of the 2008 Conference on BEyond Time and Errors Novel evaLuation Methods for Information Visualization - BELIV '08*. The 2008 Conference. Florence, Italy: ACM Press, 2008, p. 1. ISBN: 978-1-60558-016-6. DOI: 10.1145/1377966.1377977.

[Xie+18] Liyang Xie et al. *Differentially Private Generative Adversarial Network*. Feb. 19, 2018. DOI: 10.48550/arXiv.1802.06739. arXiv: 1802.06739 [cs, stat]. URL: http://arxiv.org/abs/1802.06739 (visited on 02/27/2023). preprint.

[Xin+20] Bangzhou Xin et al. "Private FL-GAN: Differential Privacy Synthetic Data Generation Based on Federated Learning". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. May 2020, pp. 2927–2931. DOI: 10.1109/ICASSP40776.2020.9054559.

[Xu+19] Lei Xu et al. "Modeling Tabular Data Using Conditional GAN". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[XV18] Lei Xu and Kalyan Veeramachaneni. *Synthesizing Tabular Data Using Generative Adversarial Networks*. Nov. 27, 2018. DOI: 10.48550/arXiv.1811.11264. arXiv: 1811.11264 [cs, stat]. URL: http://arxiv.org/abs/1811.11264 (visited on 07/16/2022). preprint.

[YDvdS20] Jinsung Yoon, Lydia N. Drumright, and Mihaela van der Schaar. "Anonymization through Data Synthesis Using Generative Adversarial Networks (ADS-GAN)". In: *IEEE Journal of Biomedical and Health Informatics* 24.8 (Aug. 2020), pp. 2378–2388. ISSN: 2168-2208. DOI: 10.1109/JBHI.2020.2980262.

[Yoo+20] Jinsung Yoon et al. "VIME: Extending the Success of Self- and Semi-supervised Learning to Tabular Domain". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 11033–11043.

[Zbi22] Robin Zbinden. "Implementing and Experimenting with Diffusion Models for Text-to-Image Generation". MA thesis. Lausanne, Switzerland: École Polytechnique Fédérale de Lausanne, Sept. 22, 2022. 63 pp. arXiv: 2209.10948 [cs].

[ZC22] Shuhan Zheng and Nontawat Charoenphakdee. *Diffusion Models for Missing Value Imputation in Tabular Data*. Oct. 31, 2022. arXiv: 2210.17128 [cs]. URL: http://arxiv.org/abs/2210.17128 (visited on 11/16/2022). preprint.

[ZC23] Shuhan Zheng and Nontawat Charoenphakdee. *Diffusion Models for Missing Value Imputation in Tabular Data*. pfnet-research, Mar. 3, 2023.

[Zha+17] Jun Zhang et al. "PrivBayes: Private Data Release via Bayesian Networks". In: *ACM Transactions on Database Systems* 42.4 (Oct. 27, 2017), 25:1–25:41. ISSN: 0362-5915. DOI: 10.1145/3134428.

[Zha+21a] Yishuo Zhang et al. "GANBLR: A Tabular Data Generation Model". In: *2021 IEEE International Conference on Data Mining (ICDM)*. 2021 IEEE International Conference on Data Mining (ICDM). Auckland, New Zealand: IEEE, Dec.

2021, pp. 181–190. ISBN: 978-1-66542-398-4. DOI: 10.1109/ICDM51629.2021.00103.

[Zha+21b] Zilong Zhao et al. “CTAB-GAN: Effective Table Data Synthesizing”. In: *Proceedings of The 13th Asian Conference on Machine Learning*. Asian Conference on Machine Learning. PMLR, May 31, 2021, pp. 97–112.

[Zha+21c] Zilong Zhao et al. “CTAB-GAN: Effective Table Data Synthesizing”. In: *Proceedings of The 13th Asian Conference on Machine Learning*. Asian Conference on Machine Learning. PMLR, Nov. 28, 2021, pp. 97–112.

[Zha+22] Zilong Zhao et al. *CTAB-GAN+: Enhancing Tabular Data Synthesis*. Apr. 1, 2022. arXiv: 2204.00401 [cs]. URL: http://arxiv.org/abs/2204.00401 (visited on 07/04/2022). preprint.

[Zha+23] Yifan Zhang et al. “Deep Long-Tailed Learning: A Survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023), pp. 1–20. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2023.3268118.

[Zhu+21] Yitan Zhu et al. “Converting Tabular Data into Images for Deep Learning with Convolutional Neural Networks”. In: *Scientific Reports* 11.1 (May 31, 2021). ISSN: 2045-2322. DOI: 10.1038/s41598-021-90923-y.

[ZHW18] Yucan Zhou, Qinghua Hu, and Yu Wang. “Deep Super-Class Learning for Long-Tail Distributed Image Classification”. In: *Pattern Recognition* 80 (Aug. 2018), pp. 118–128. ISSN: 00313203. DOI: 10.1016/j.patcog.2018.03.003.

8 Appendix

A.1 Extended derivations

A.1.1 Diffusion propabilistic models

Several derivations have been presented in a short format for better readability. This section will cover left-out derivations. For a more detailed explanation [Wen21; Out22] is recommended. For further details, please refer to [Soh+15; HJA20].

Derivation to get from Equation 2.15 to Equation 2.17:

Given:

$$KL(p \parallel q) = \int p(x) \cdot log(\frac{p(x)}{q(x)})dx \tag{A.1}$$

and

$$\begin{aligned} p_\theta(x_{1:T}|x_0) &= \frac{p_\theta(x_0|x_{1:T})p_\theta(x_{1:T})}{p_\theta(x_0)} && \text{using bayes rule} \\ &= \frac{p_\theta(x_0, x_{1:T})}{p_\theta(x_0)} && \text{summarize to joint probability} \\ &= \frac{p_\theta(x_{0:T})}{p_\theta(x_0)} && \text{summarize to joint probability} \end{aligned} \tag{A.2}$$

Derivation:

$$-log(p_\theta(x_0)) \leq -log(p_\theta(x_0)) + KL(q(x_{1:T}|x_0) \parallel p_\theta(x_{1:T}|x_0)) \tag{A.3a}$$

$$= -log(p_\theta(x_0)) + log(\frac{q(x_{1:T})|x_0}{p_\theta(x_{1:T}|x_0)}) \quad \text{using Equation A.1} \tag{A.3b}$$

$$= -log(p_\theta(x_0)) + log(\frac{q(x_{1:T})|x_0}{\frac{p_\theta(x_{0:T})}{p_\theta(x_0)}}) \quad \text{using Equation A.2} \tag{A.3c}$$

$$= -log(p_\theta(x_0)) + log(\frac{q(x_{1:T})|x_0}{p_\theta(x_{0:T})}) + log(p_\theta(x_0)) \quad \text{Fraction rule} \tag{A.3d}$$

$$= log(\frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}) \quad \text{canceling out} \tag{A.3e}$$

$$\text{Let } L_{VLB} = \mathbb{E}_{q(x_{0:T})}[log(\frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})})] \geq -\mathbb{E}_{q(x_0)} log(p_\theta(x_0)) \tag{A.3f}$$

Derivation to get from Equation 2.17 to Equation 2.18:

$$L_{VLB} = \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{0:T}\right)}\right] \tag{A.4a}$$

$$= \mathbb{E}_q\left[\log \frac{\prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)}{p_\theta\left(\mathbf{x}_T\right) \prod_{t=1}^{T} p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)}\right] \tag{A.4b}$$

$$= \mathbb{E}_q\left[-\log p_\theta\left(\mathbf{x}_T\right) + \sum_{t=1}^{T} \log \frac{q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)}{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)}\right] \tag{A.4c}$$

$$= \mathbb{E}_q\left[-\log p_\theta\left(\mathbf{x}_T\right) + \sum_{t=2}^{T} \log \frac{q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)}{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)} + \log \frac{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}\right] \tag{A.4d}$$

$$= \mathbb{E}_q\left[-\log p_\theta\left(\mathbf{x}_T\right) + \sum_{t=2}^{T} \log \left(\frac{q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)} \cdot \frac{q\left(\mathbf{x}_t \mid \mathbf{x}_0\right)}{q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_0\right)}\right) + \log \frac{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}\right] \tag{A.4e}$$

$$= \mathbb{E}_q\left[-\log p_\theta\left(\mathbf{x}_T\right) + \sum_{t=2}^{T} \log \frac{q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)} + \sum_{t=2}^{T} \log \frac{q\left(\mathbf{x}_t \mid \mathbf{x}_0\right)}{q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_0\right)} + \log \frac{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}\right] \tag{A.4f}$$

$$= \mathbb{E}_q\left[-\log p_\theta\left(\mathbf{x}_T\right) + \sum_{t=2}^{T} \log \frac{q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)} + \log \frac{q\left(\mathbf{x}_T \mid \mathbf{x}_0\right)}{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)} + \log \frac{q\left(\mathbf{x}_1 \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}\right] \tag{A.4g}$$

$$= \mathbb{E}_q\left[\log \frac{q\left(\mathbf{x}_T \mid \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_T\right)} + \sum_{t=2}^{T} \log \frac{q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)} - \log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)\right] \tag{A.4h}$$

$$= \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_T\right)\right)}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)}_{L_{t-1}} - \underbrace{\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}] \tag{A.4i}$$

A.2 Activity diagrams

The following section will display the procedures of different scripts. Firstly, the scripts' most important processes are displayed as implemented by [Kot+22]. The processes of the scripts with the implemented changes are displayed afterward.

A.2.1 Original Implementation

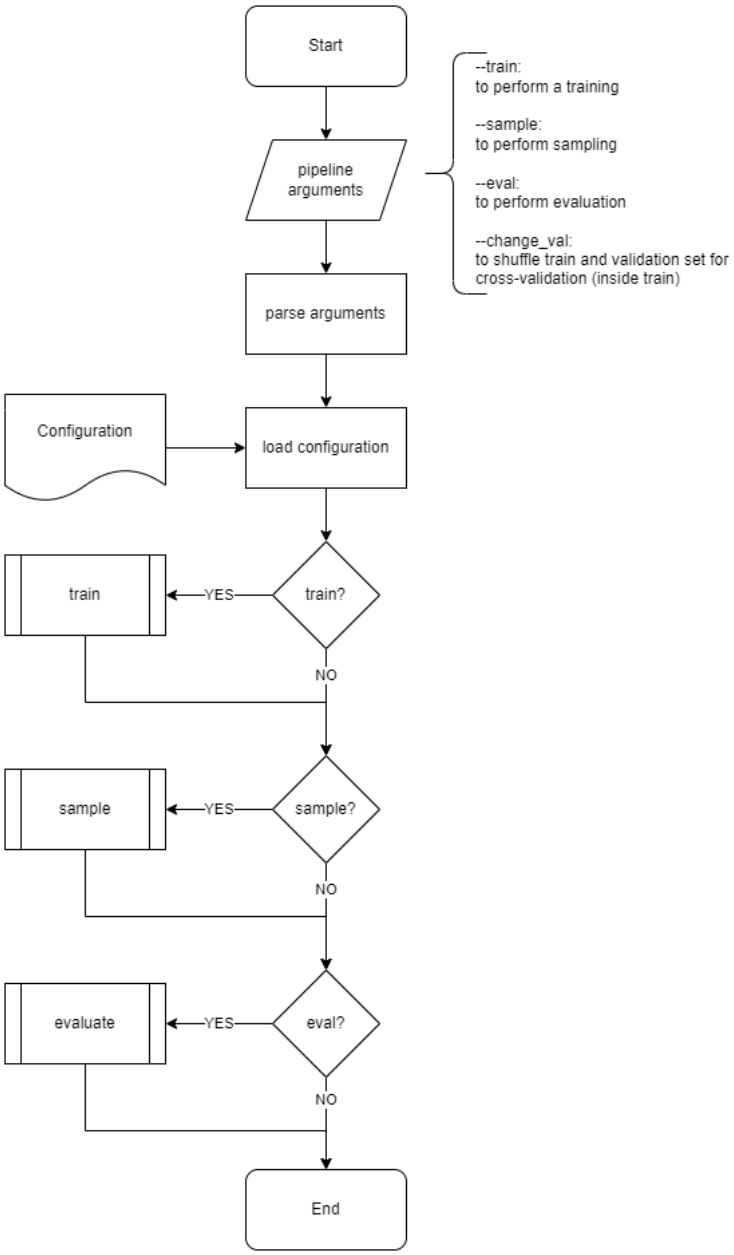


Figure A-1: simplified visualization of the pipeline_.py script

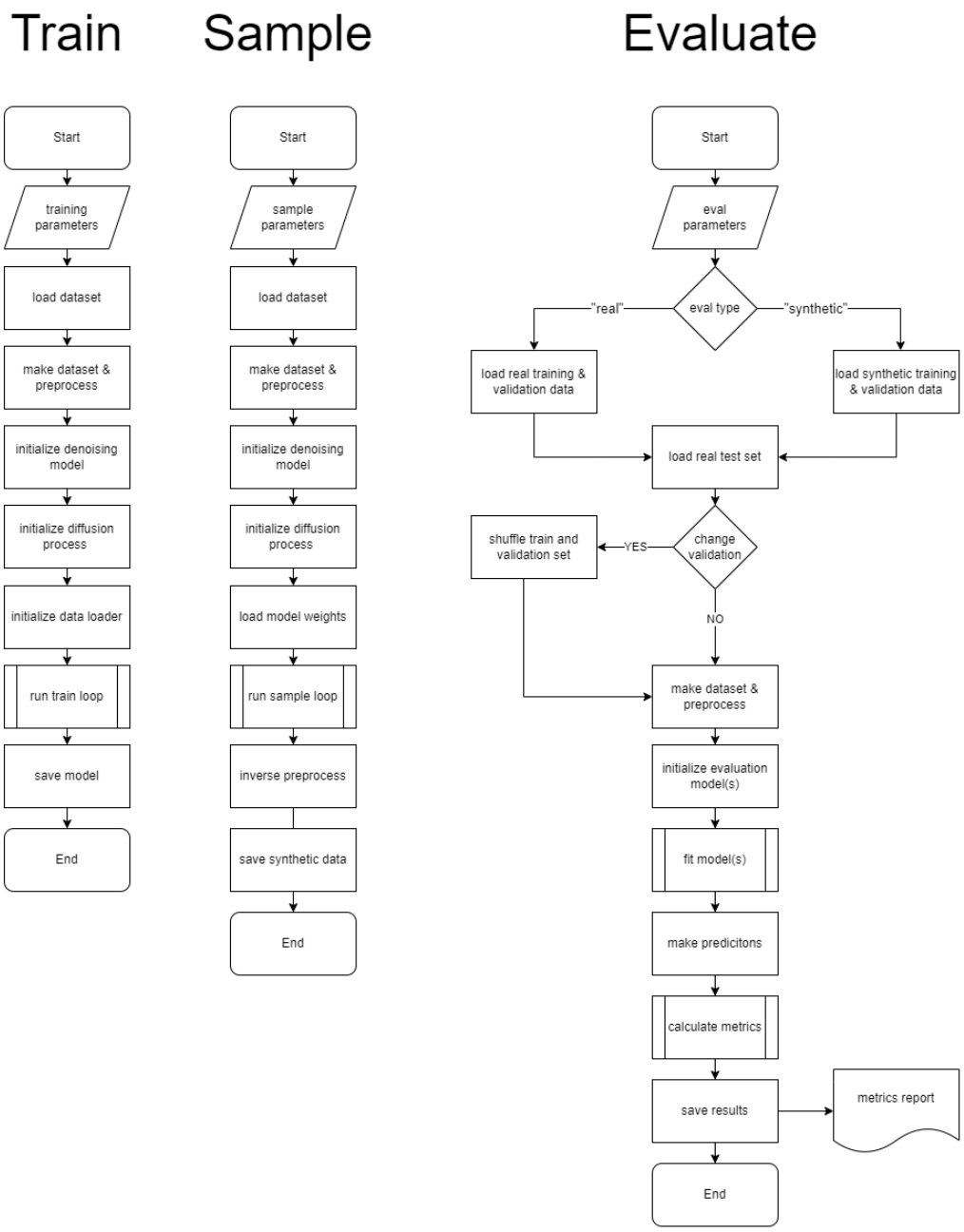


Figure A-2: simplified visualization of the train.py, sample.py and eval_[catboost | mlp | simple].py scripts

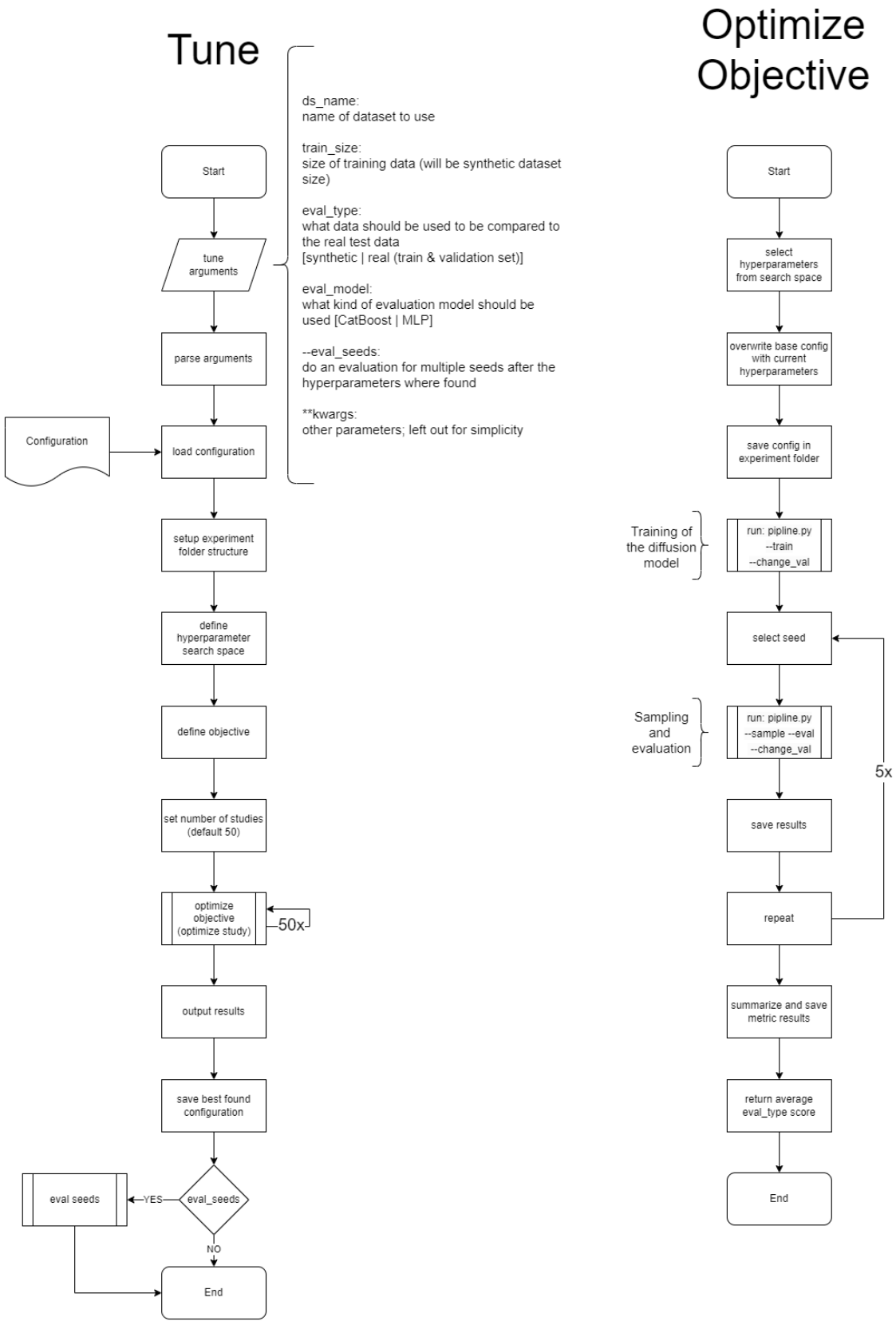


Figure A-3: simplified visualization of the tune_*.py scripts

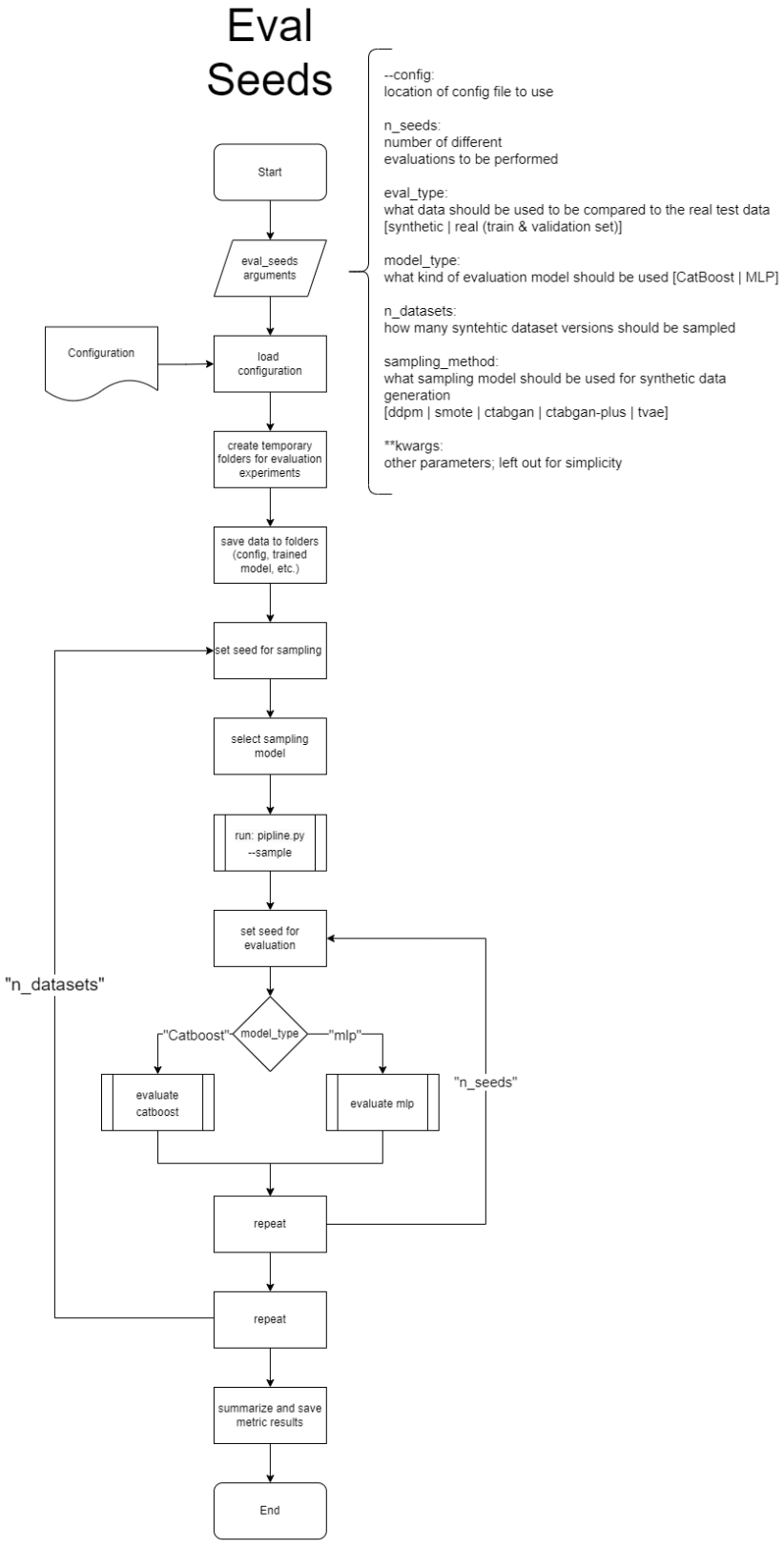


Figure A-4: simplified visualization of the eval_seeds.py script

A.2.2 Changed Implementation

Several changes have been made to the different scripts. The following figures will display the most important changes, highlighted in green. Please note that not all changes to the original implementation can be included, and minor modifications or bug fixes are not displayed.

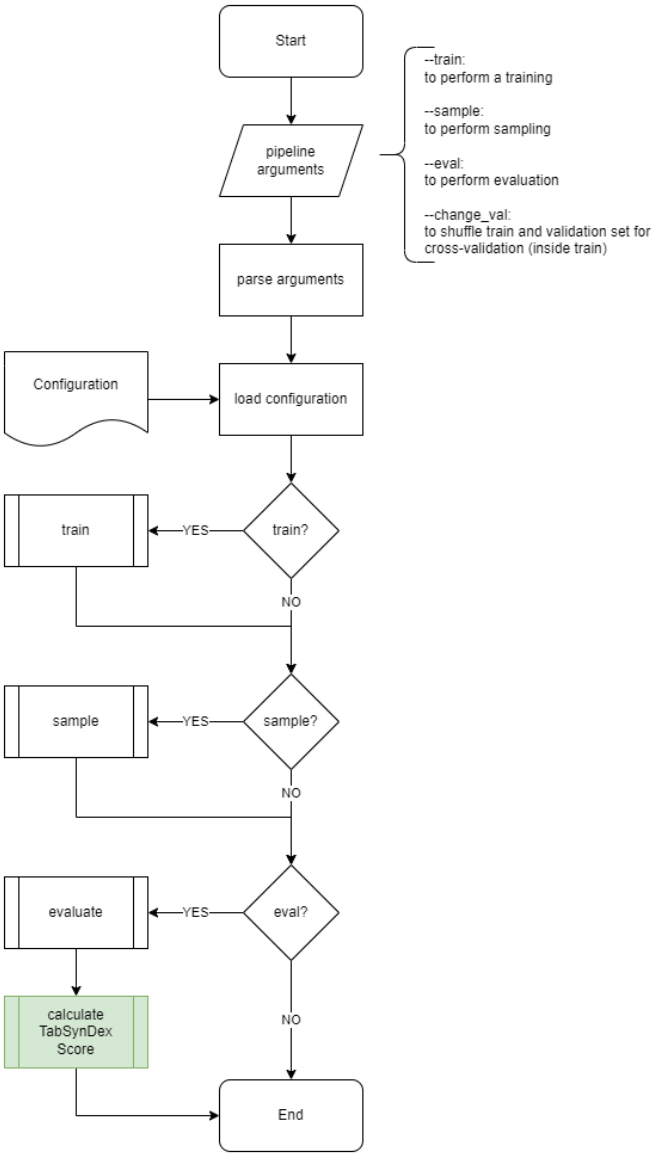


Figure A-5: simplified visualization of the changed pipeline_.py script

The basic pipeline script hardly changed. However, the individual components did change and are displayed in Figure A-6. The previous evaluation part in Figure A-2 is unchanged. Instead, Figure A-6 displays only the new similarity evaluation script, which is executed after the previous evaluation script (compare Figure A-5).

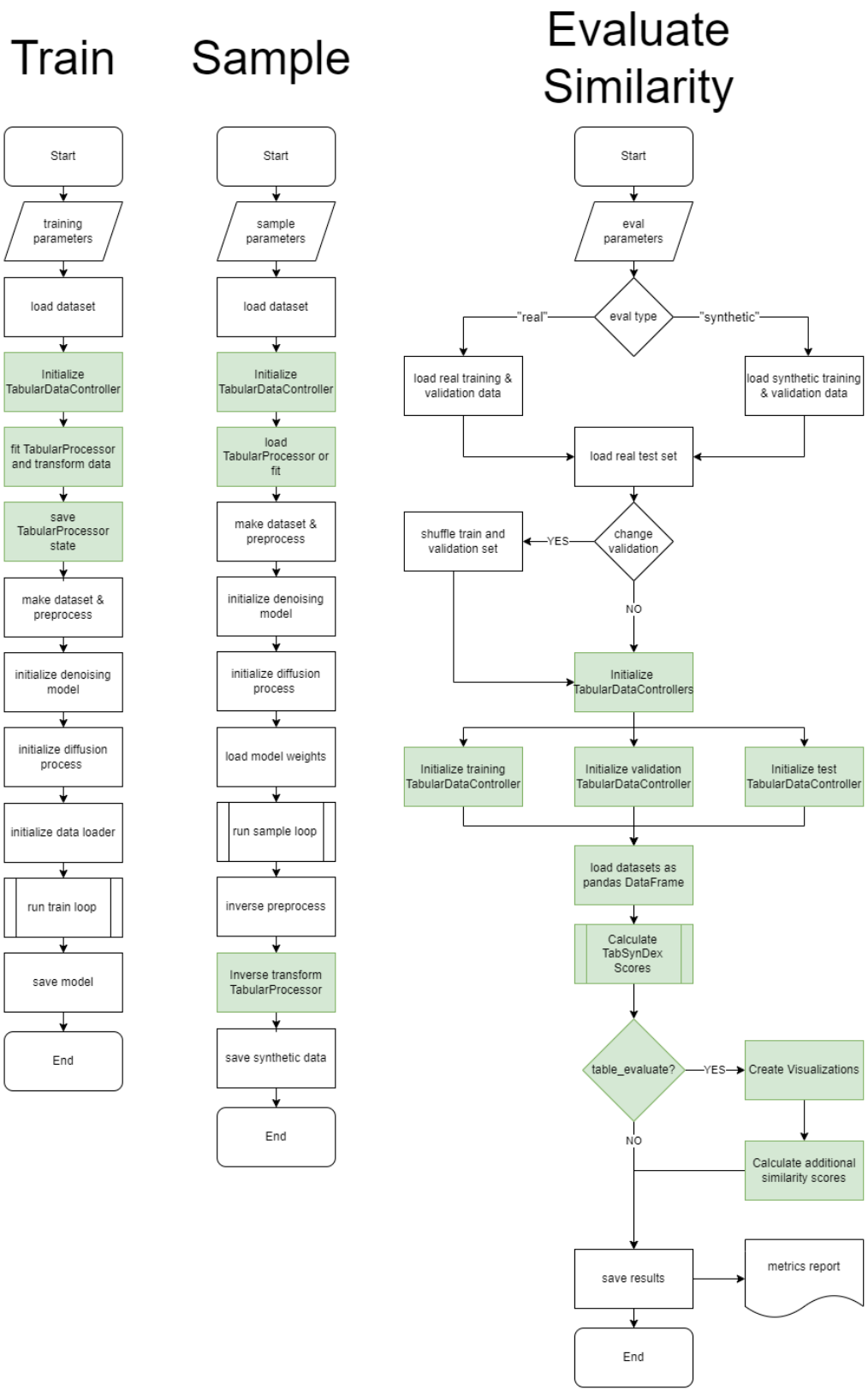


Figure A-6: simplified visualization of the train.py, sample.py and eval_[catboost | mlp | simple].py scripts

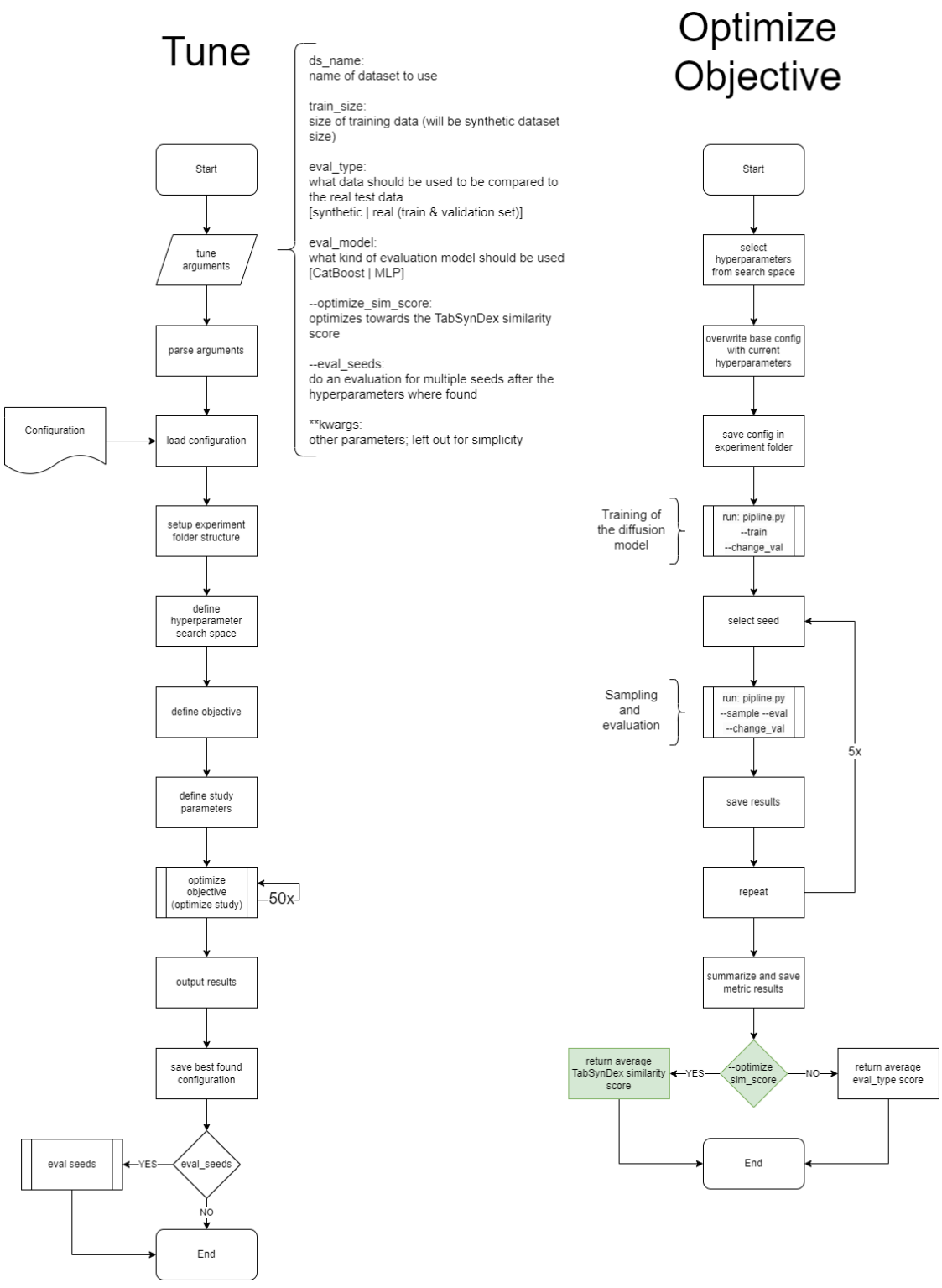


Figure A-7: simplified visualization of the changed tune_*.py scripts

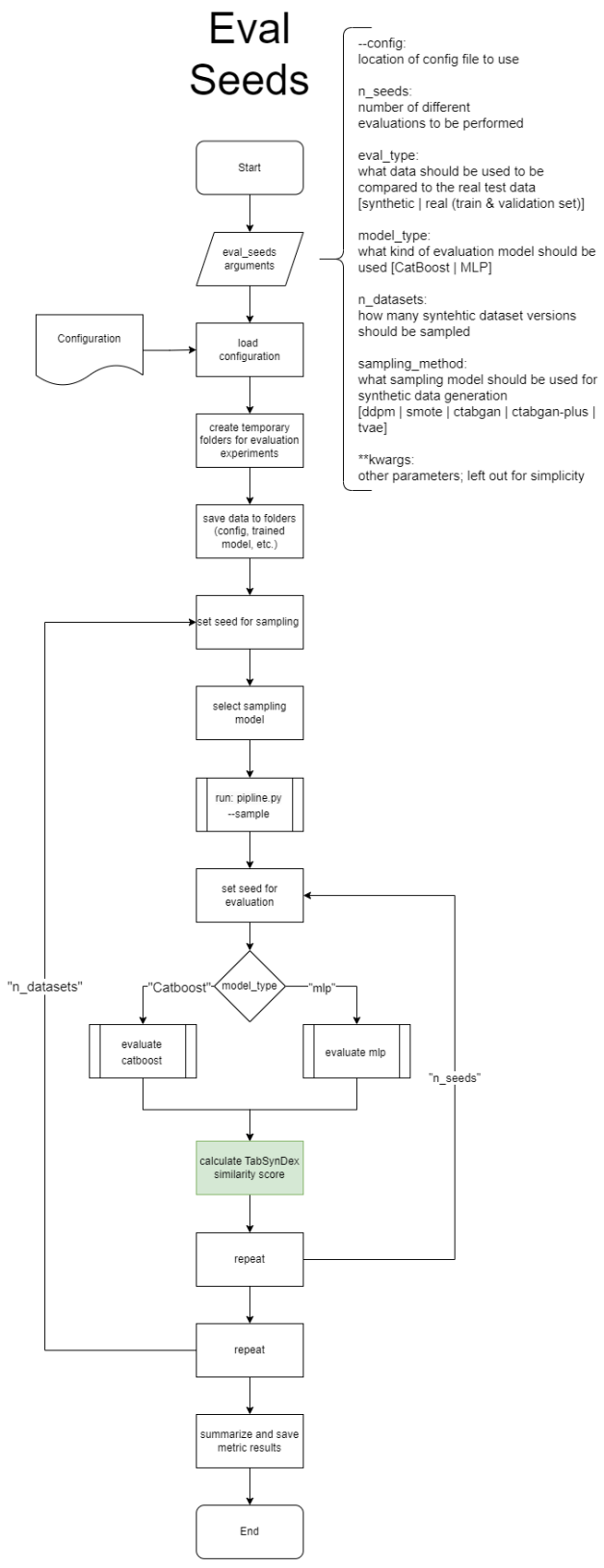
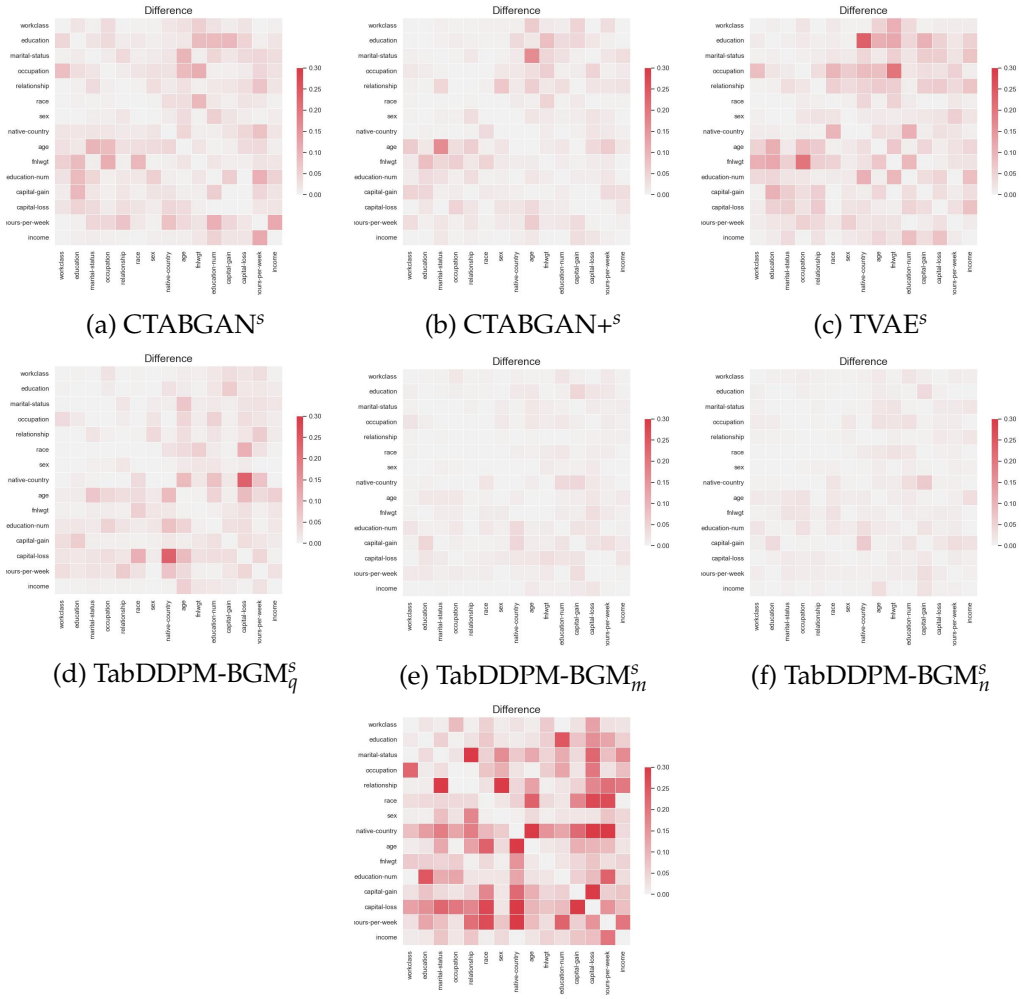


Figure A-8: simplified visualization of the changed eval_seeds.py script

A.3 Visual Results

A.3.1 Correlation Difference Matrix



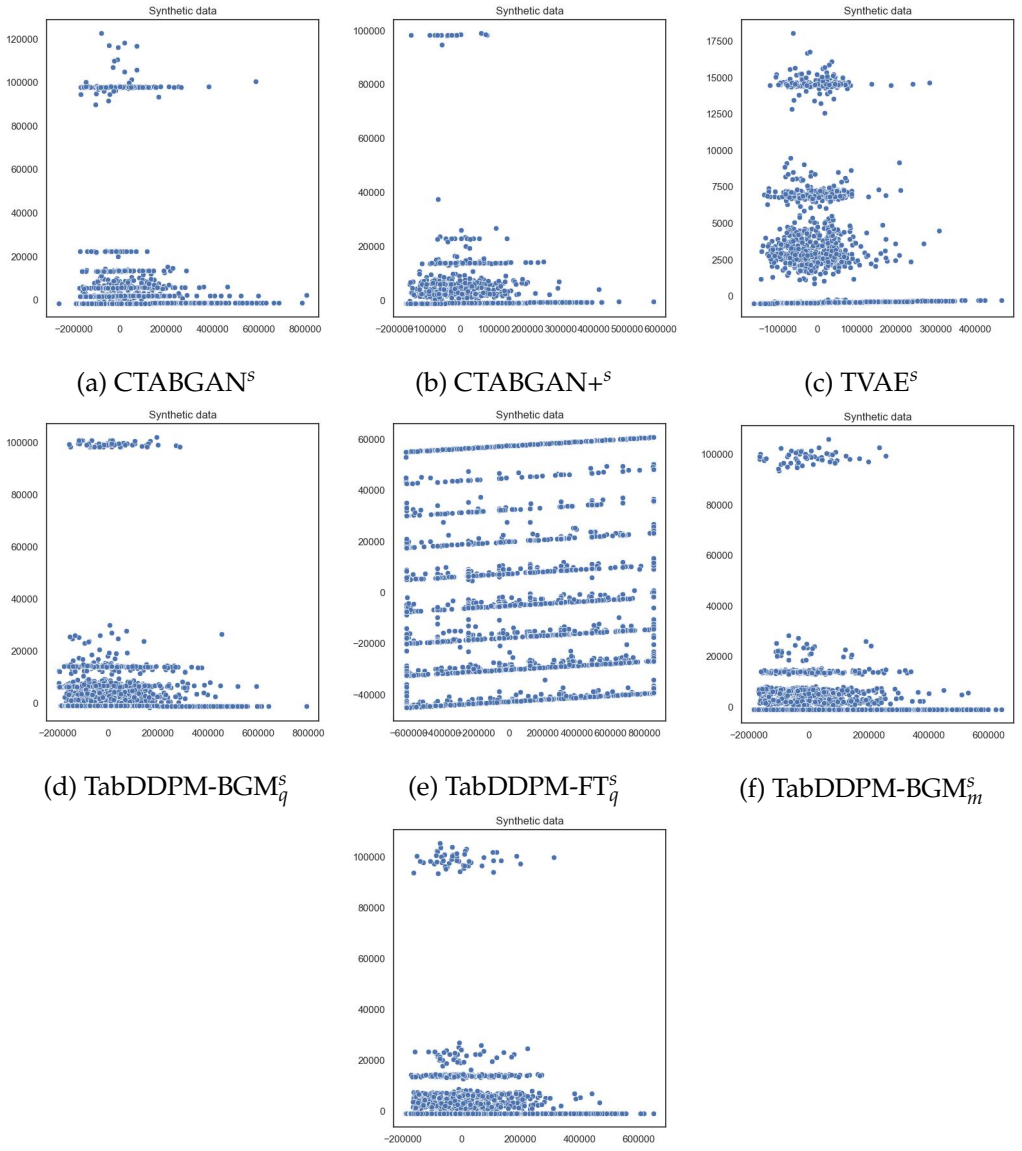
(a) CTABGAN^s (b) CTABGAN+^{s} (c) TVAE^s

(d) TabDDPM-BGM^s_q (e) TabDDPM-BGM^s_m (f) TabDDPM-BGM^s_n

(g) TabDDPM-FT^s_q

Figure A-9: Correlation difference matrix for different model versions

A.3.2 Principle Component Analysis



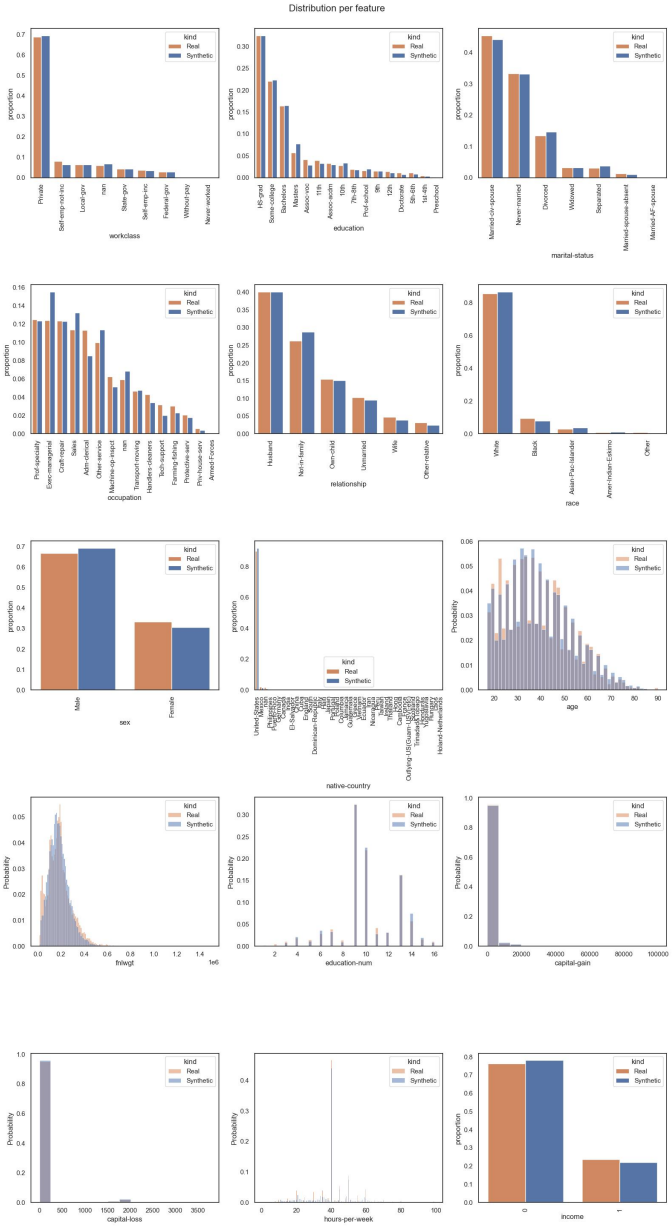
(a) CTABGAN^s (b) CTABGAN+^{s} (c) TVAE^s

(d) TabDDPM-BGM^s_q (e) TabDDPM-FT^s_q (f) TabDDPM-BGM^s_m

(g) TabDDPM-BGM^s_n

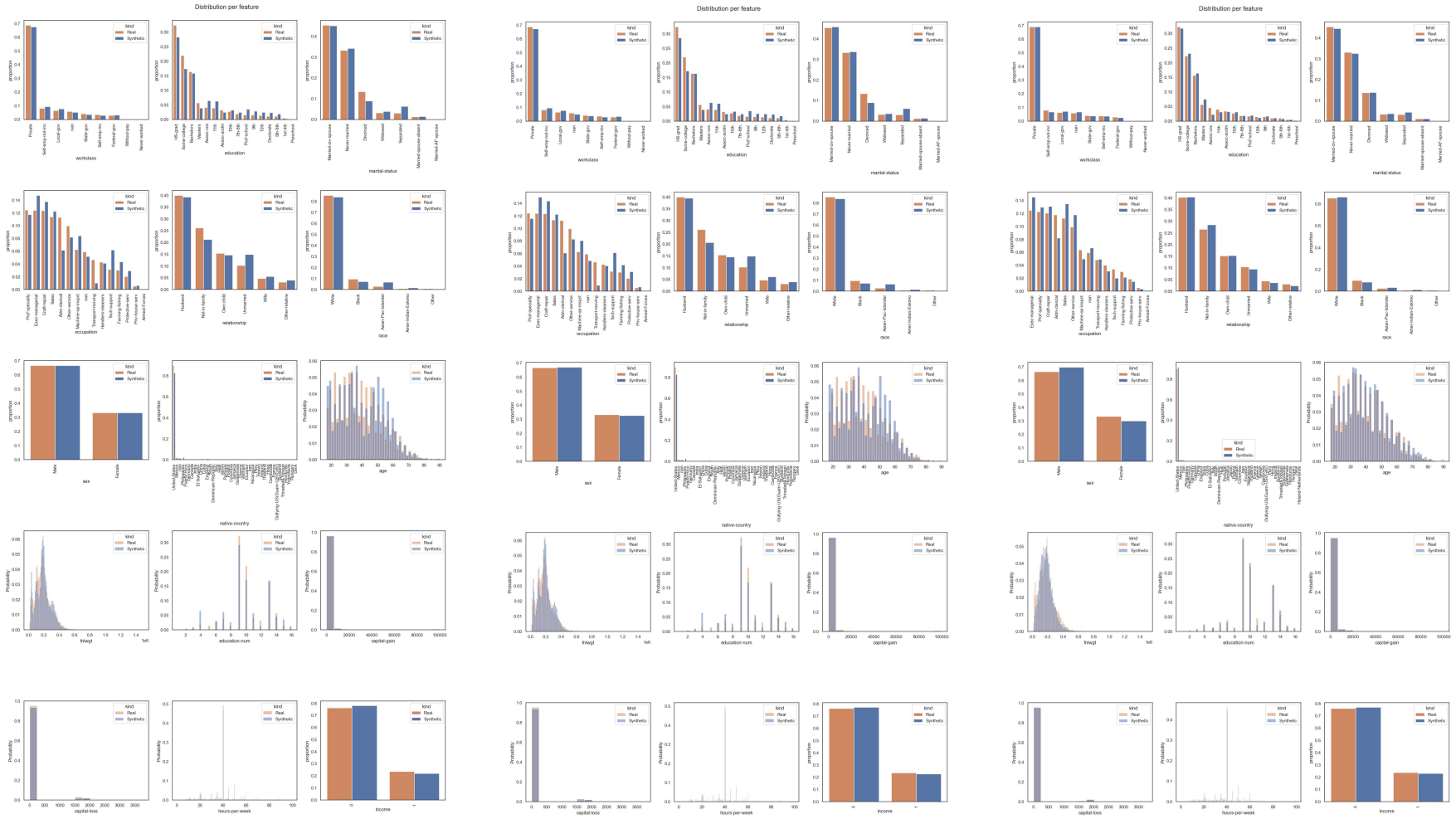
Figure A-10: Principle Component Analysis for different model versions

A.3.3 Distribution Plots



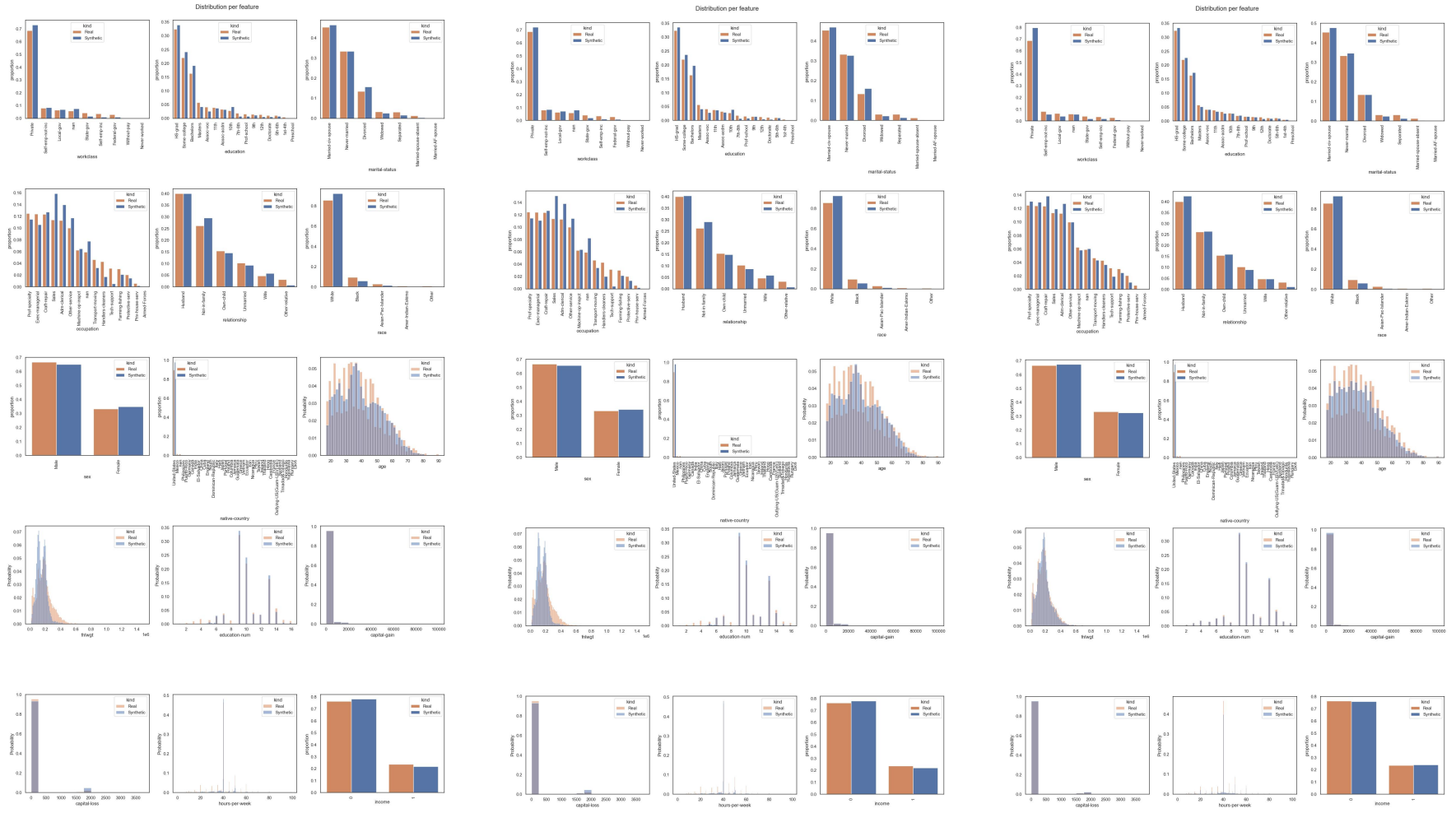
(a) CTABGAN+^{ml}

Figure A-11: Distribution plots for CTABGAN+^{ml}



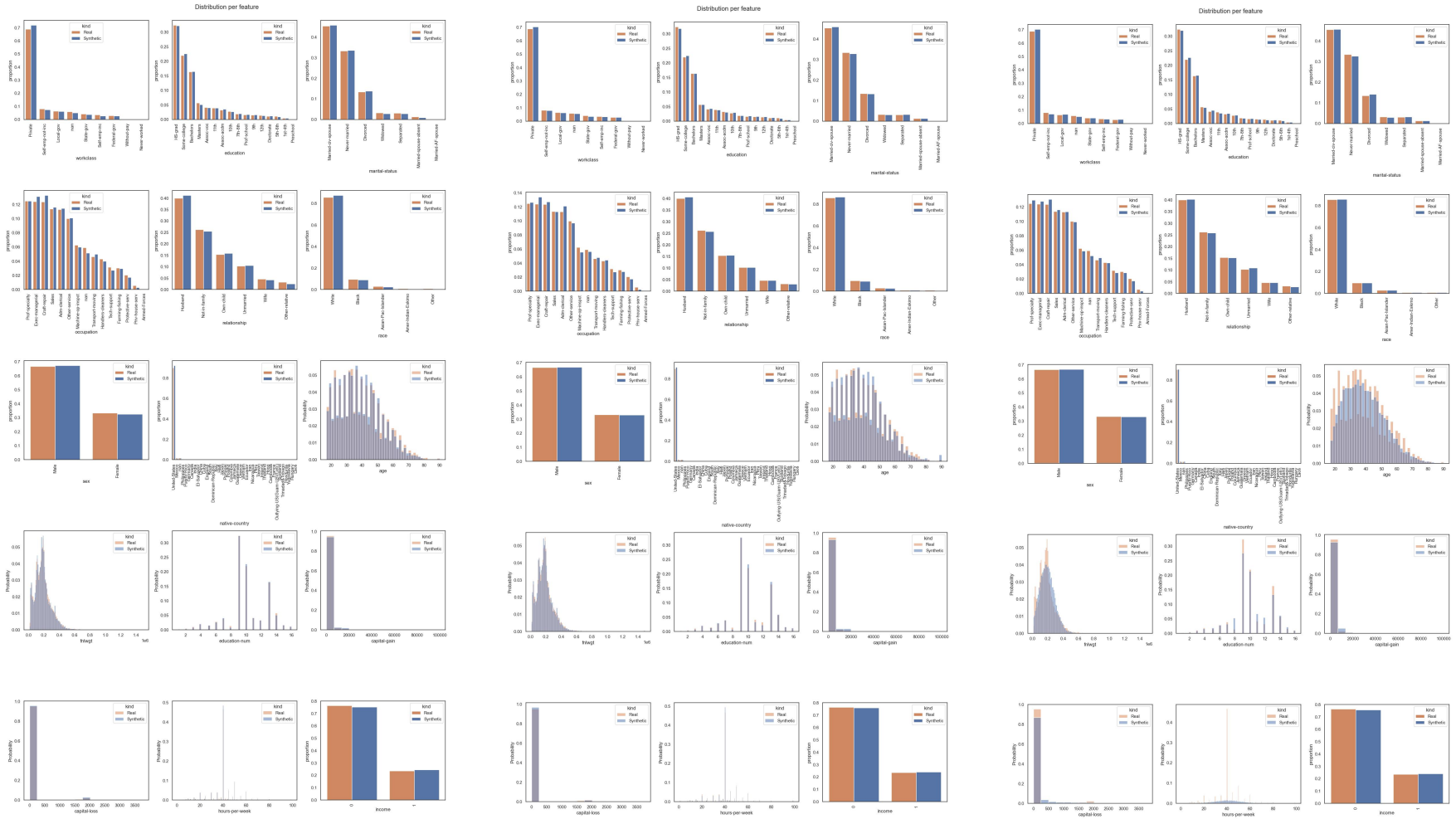
(a) CTABGAN^{ml} (b) CTABGAN^{s} (c) CTABGAN+^{s}

Figure A-12: Distribution plots for CTABGAN^{ml}, CTABGAN^{s} and CTABGAN+^{s}



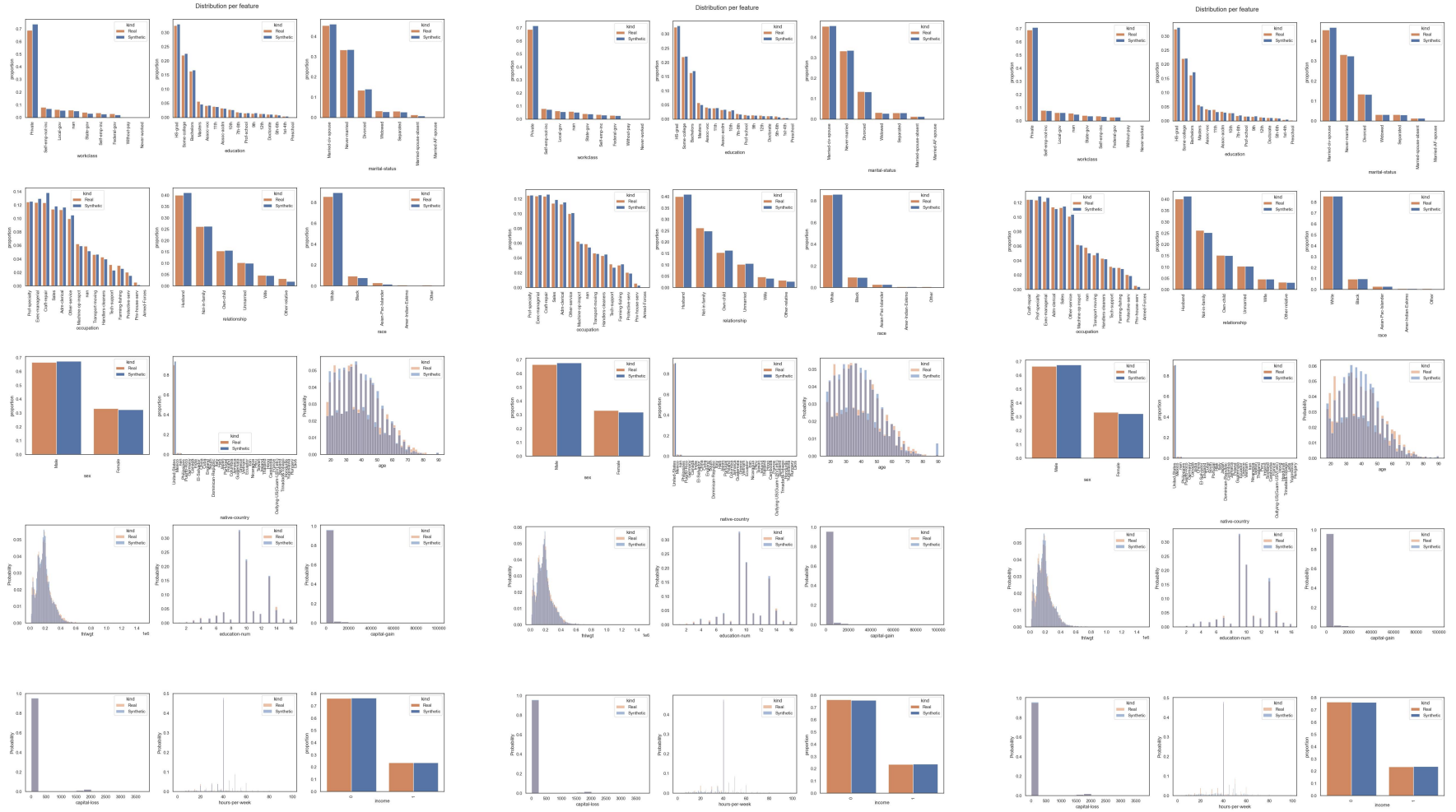
(a) TVAE^{ml} (b) TVAE^{s} (c) SMOTE

Figure A-13: Distribution plots for TVAE^{ml}, TVAE^{s} and SMOTE



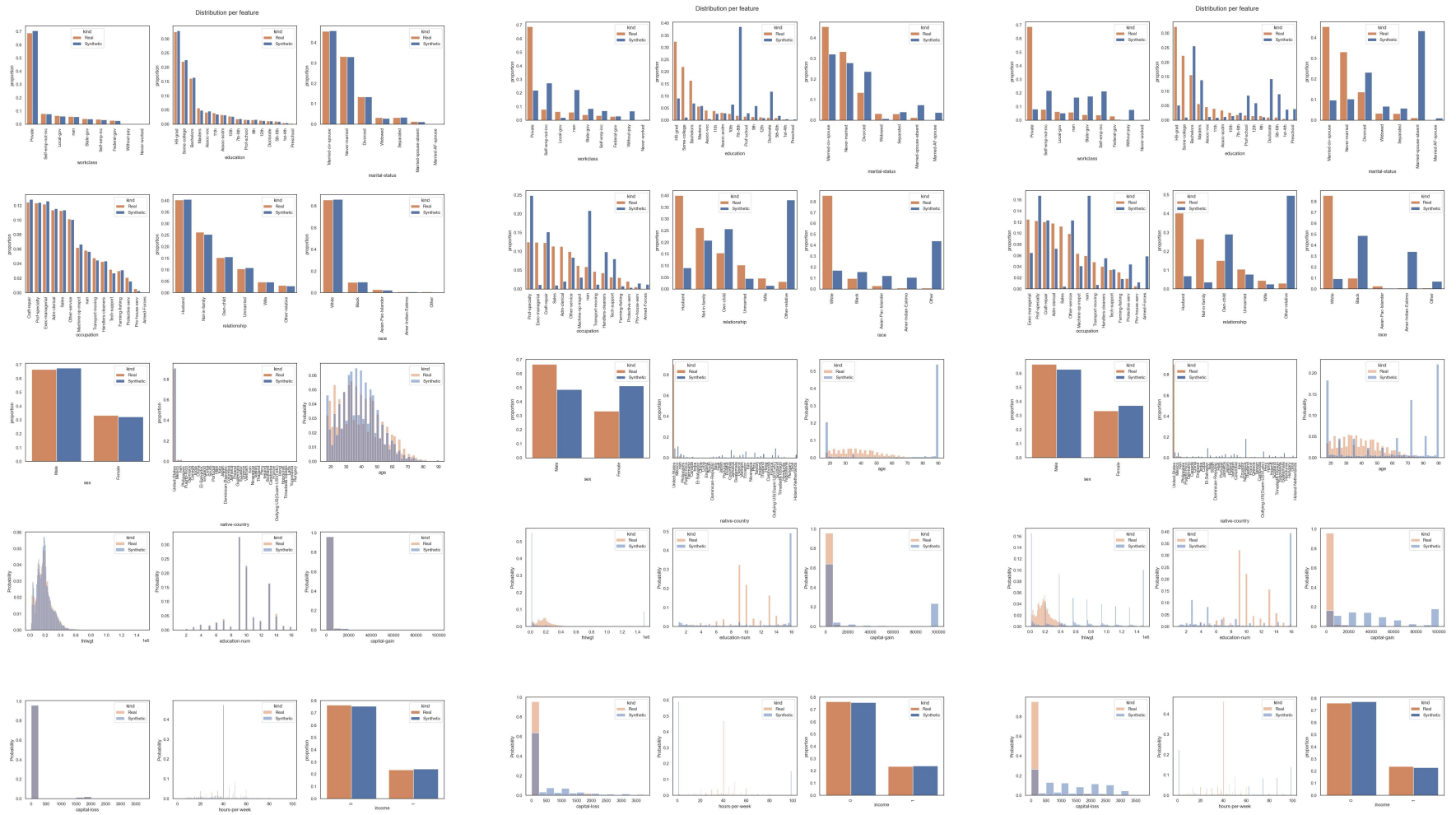
(a) TabDDPM$_q^{ml}$ (b) TabDDPM$_q^{s}$ (c) TabDDPM$_m^{s}$

Figure A-14: Distribution plots for TabDDPM variations



(a) TabDDPM-BGM$_q^{ml}$ (b) TabDDPM-BGM$_q^{s}$ (c) TabDDPM-BGM$_m^{s}$

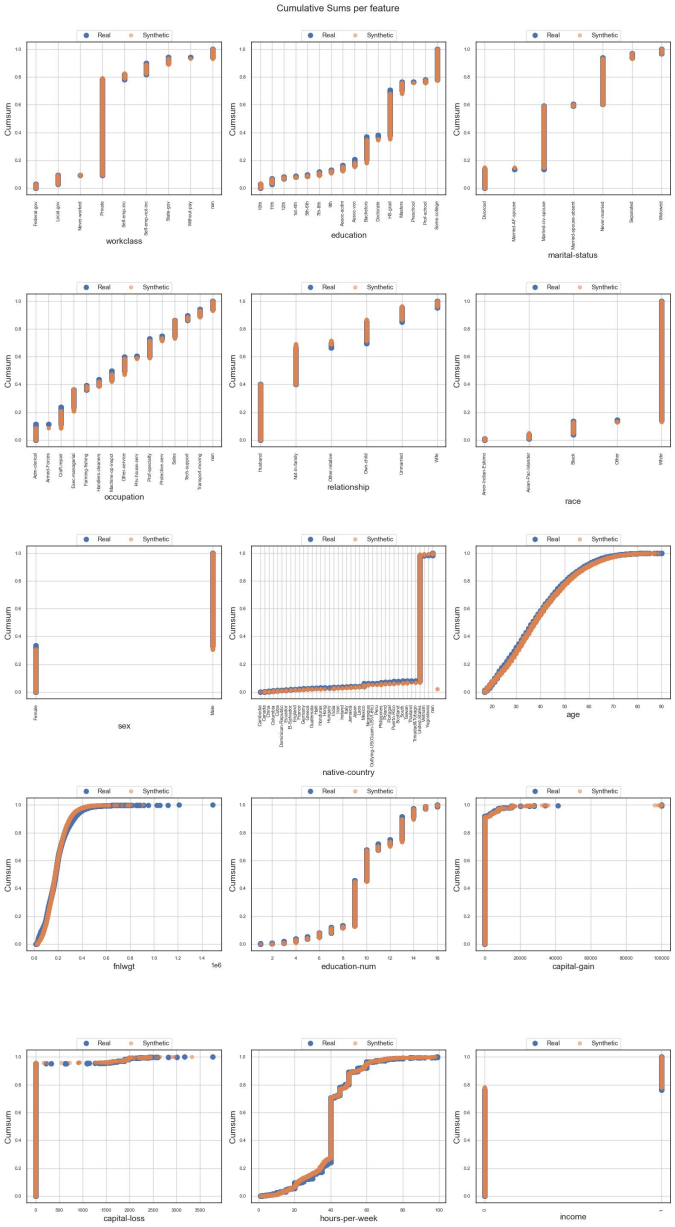
Figure A-15: Distribution plots for TabDDPM-BGM variations



(a) TabDDPM-BGM$^{s}_{n}$ (b) TabDDPM-FT$^{ml}_{q}$ (c) TabDDPM-FT$^{ml}_{q}$

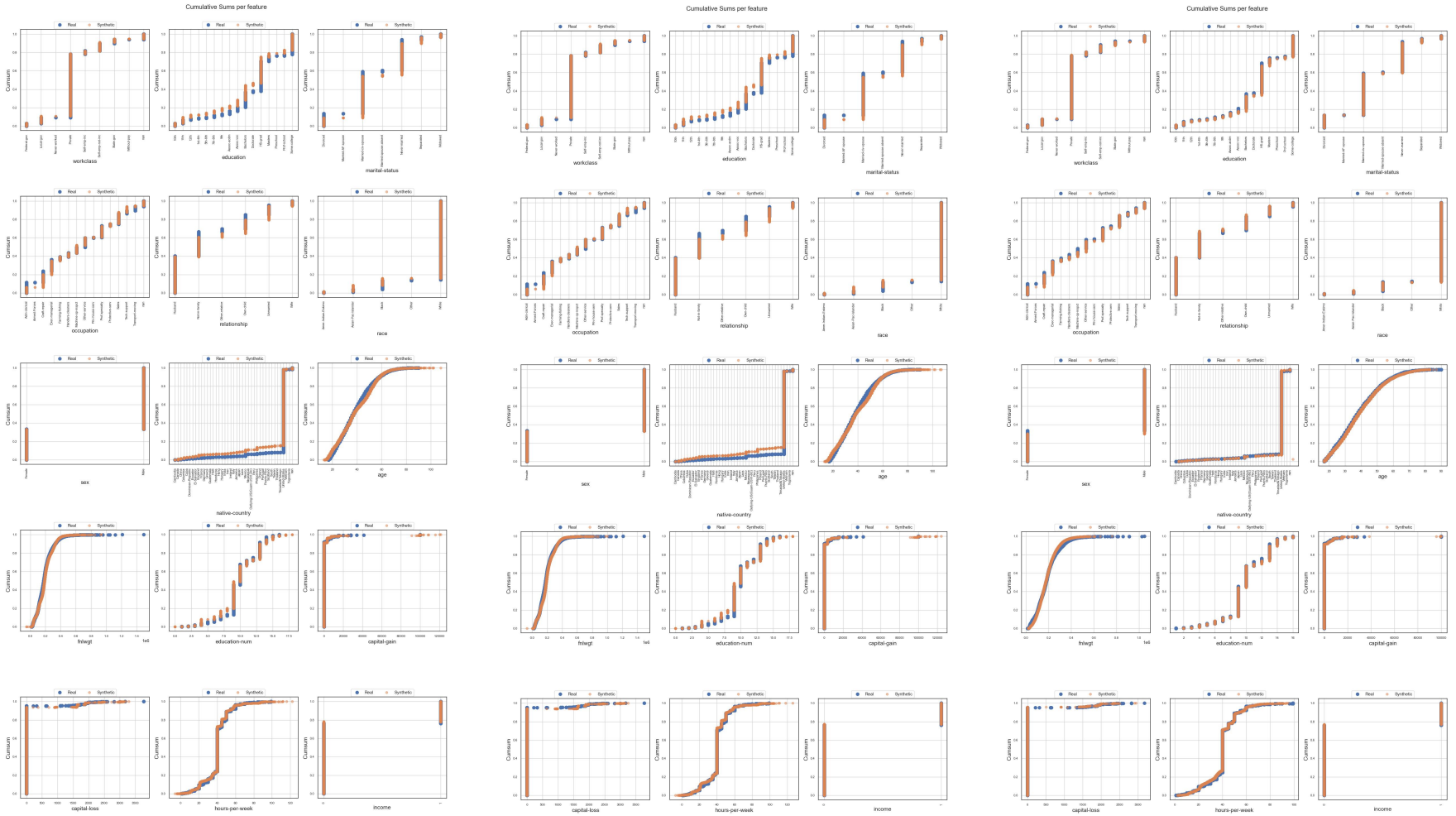
Figure A-16: Distribution plots for TabDDPM-FT variations

A.3.4 Cumulative Distribution Plots



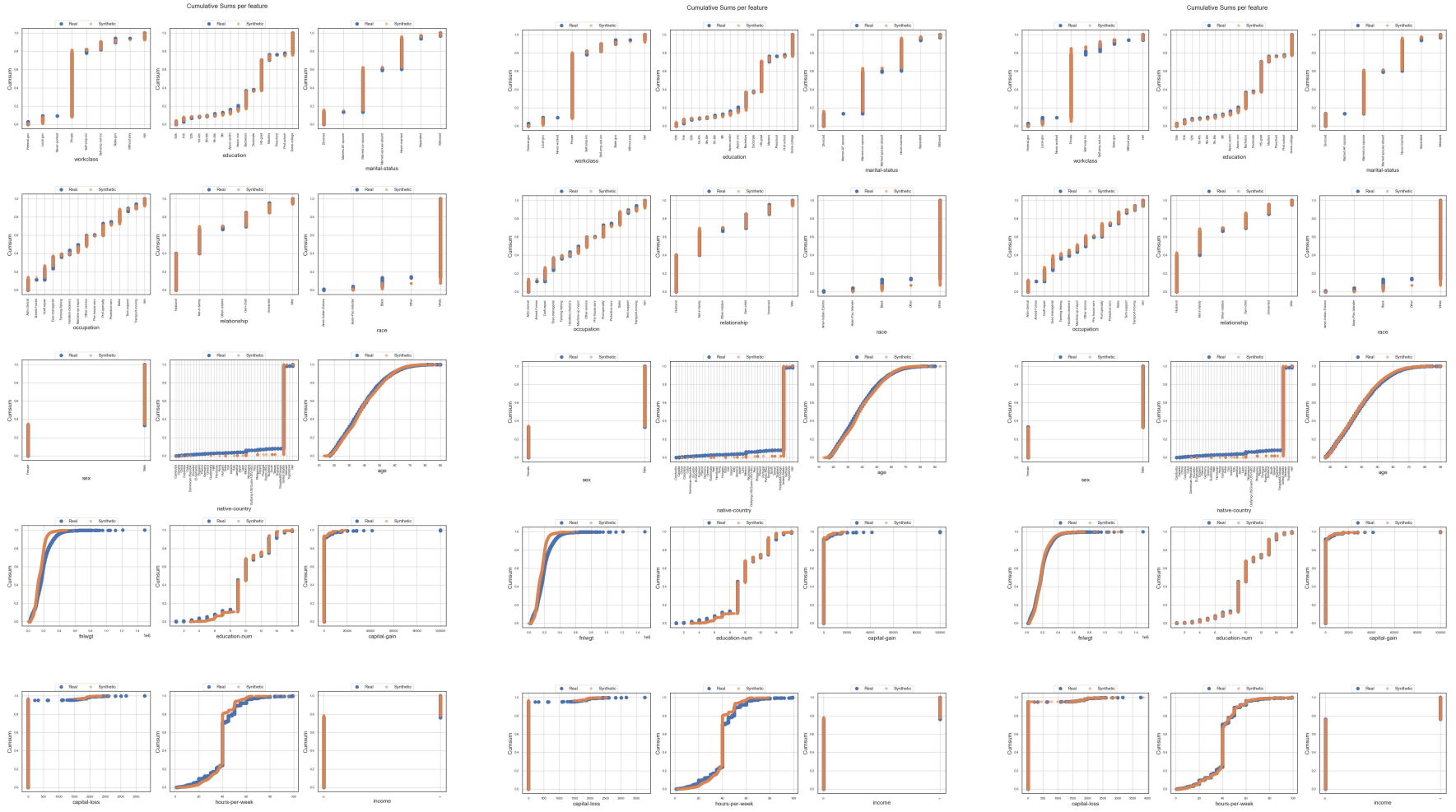
(a) CTABGAN+^{ml}

Figure A-17: Cumulative Distribution plots for CTABGAN+^{ml}



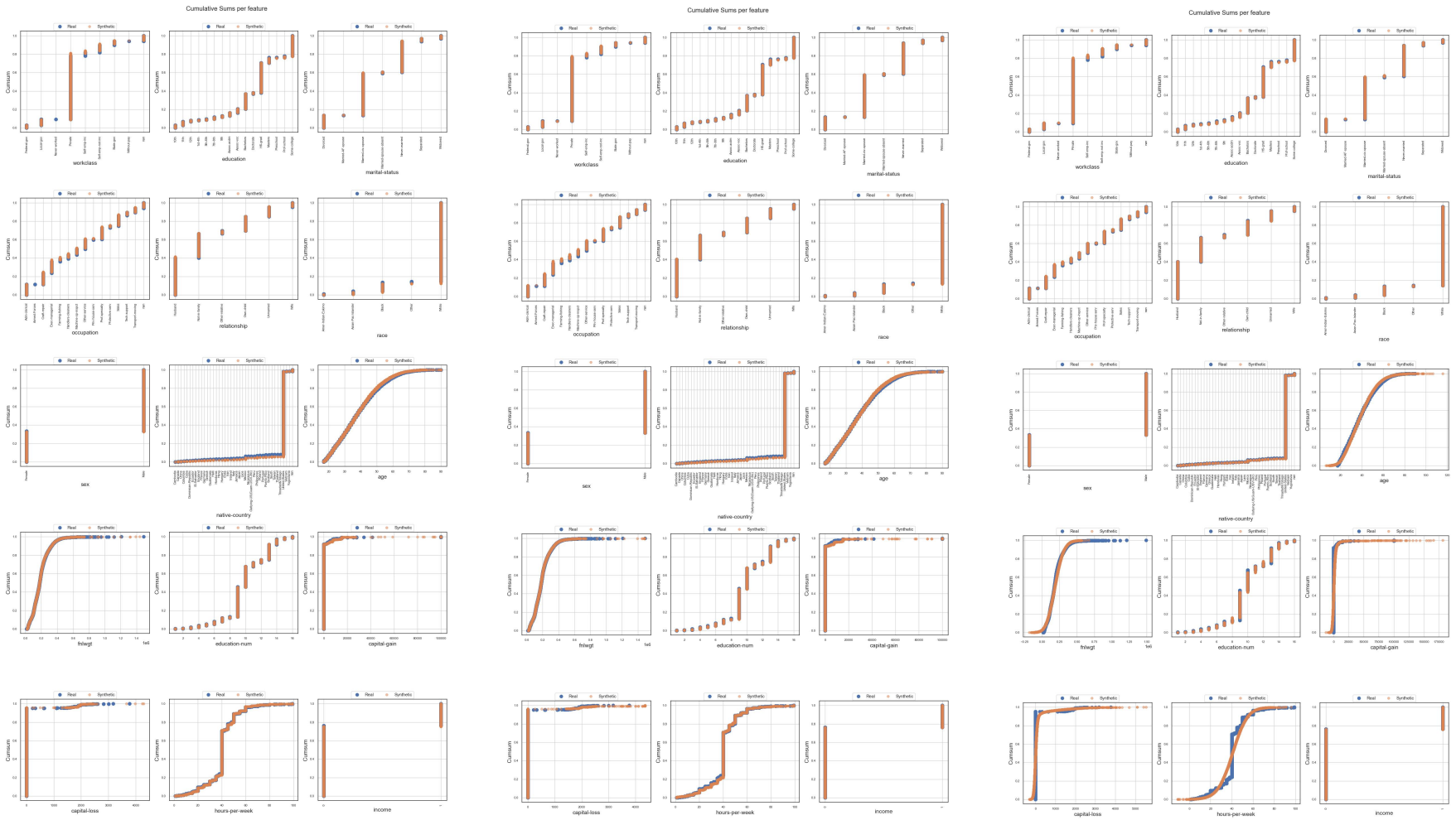
(a) CTABGAN^{ml} (b) CTABGAN^{s} (c) CTABGAN+^{s}

Figure A-18: Cumulative Distribution plots for CTABGAN^{ml}, CTABGAN^{s} and CTABGAN+^{s}



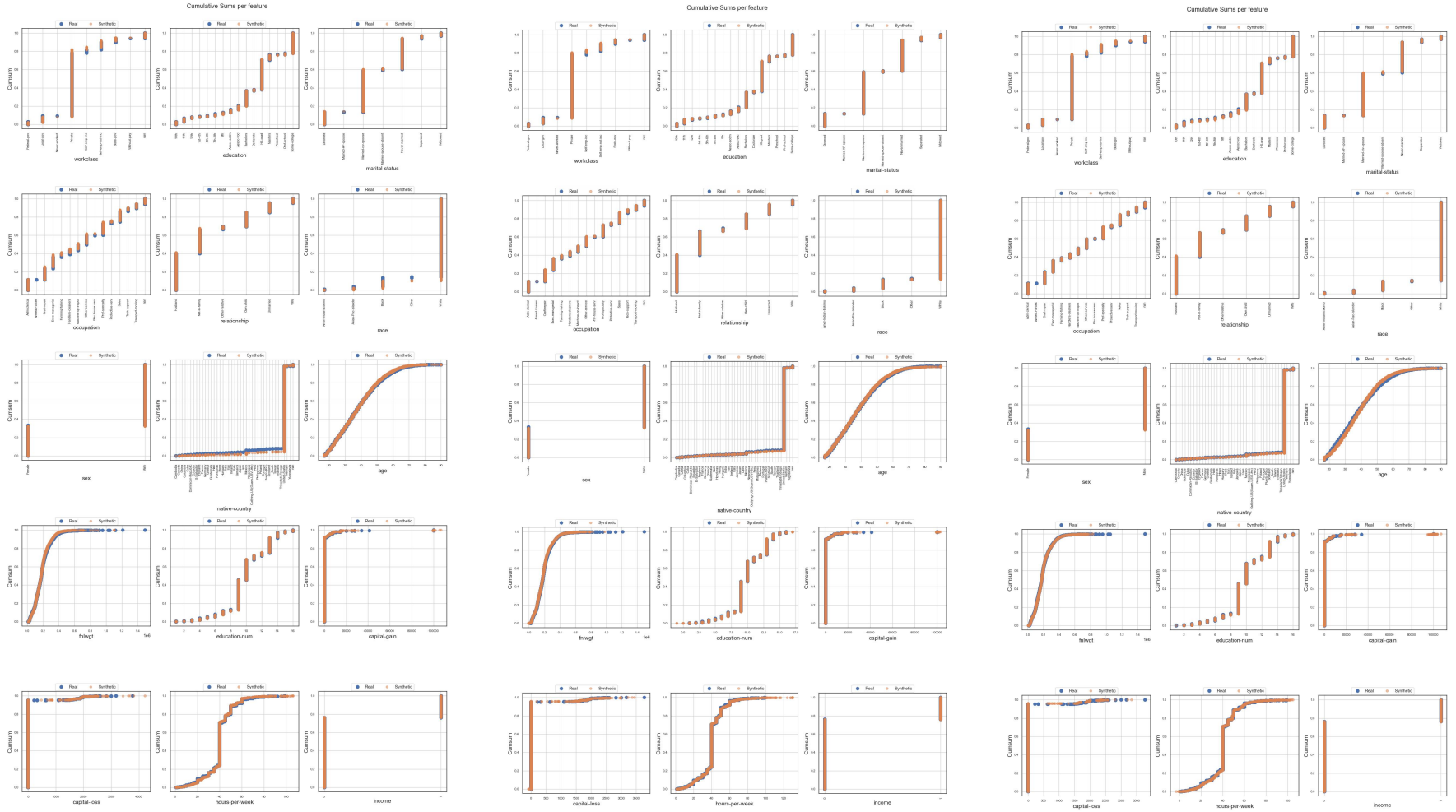
(a) TVAE^{ml} (b) TVAE^{s} (c) SMOTE

Figure A-19: Cumulative Distribution plots for TVAE^{ml}, TVAE^{s} and SMOTE



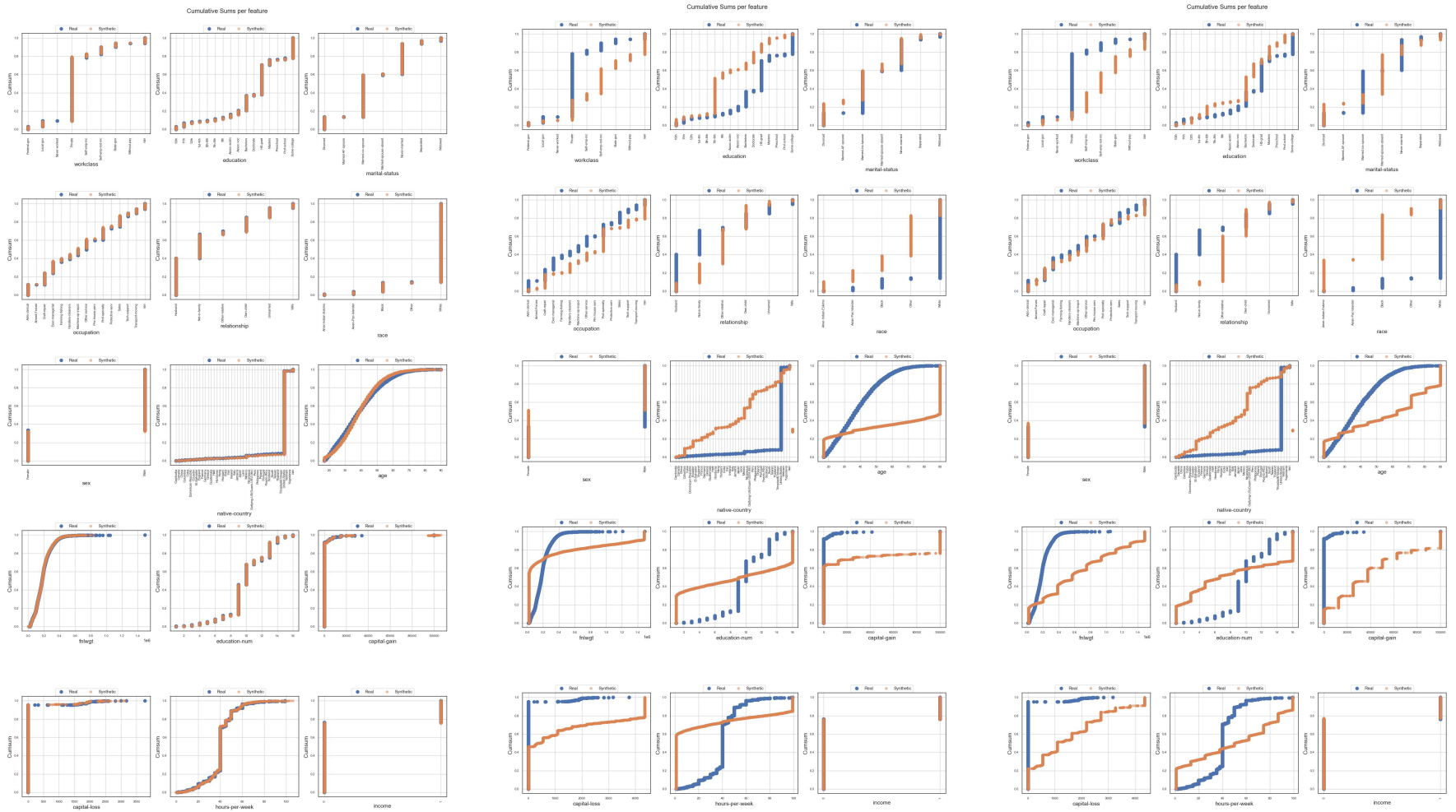
(a) TabDDPM$_q^{ml}$ (b) TabDDPM$_q^{s}$ (c) TabDDPM$_m^{s}$

Figure A-20: Cumulative Distribution plots for TabDDPM variations



(a) TabDDPM-BGM$_q^{ml}$ (b) TabDDPM-BGM$_q^{s}$ (c) TabDDPM-BGM$_m^{s}$

Figure A-21: Cumulative Distribution plots for TabDDPM-BGM variations



(a) TabDDPM-BGM$^{s}_{n}$ (b) TabDDPM-FT$^{ml}_{q}$ (c) TabDDPM-FT$^{ml}_{q}$

Figure A-22: Cumulative Distribution plots for TabDDPM-FT variations

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang IT Management und -Consulting selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten InternetQuellen - benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der elektronischen Abgabe entspricht.

Hamburg, den _____ Unterschrift: _____