POLITECNICO DI TORINO

Master's Degree in Mathematical Engineering

Master's Degree Thesis

Diffusion models for synthetic generation of tabular data: a case study

Supervisor
Prof. Francesco Vaccarino

Candidate
Luca Panichi

Academic Year 2024-2025

Summary

This thesis presents a case study on the application of diffusion models for the synthetic generation of tabular data. Generative models have shown remarkable results in producing synthetic data, and diffusion models are currently one of the most popular generative models .
After a brief introduction to the main generative models as Generative adversarial networks, Variational autoencoders and Diffusion models, the thesis shows a case study of a comparison of two diffusion models, TabDDPM and FinDiff. These are applied to two difference datasets, and the results are analysed using difference metrics, considering statistical properties and privacy aspects. The study demonstrates very high quality in synthetic generation and robust results for privacy protection for mixed tabular data.

Indice

List of Tables

List of Figures

Chapter 1

Introduction

The main aim of this thesis is to analyze the diffusion models and its applications in the case of tabular data. In this first chapter I am going to introduce the main topics of Artificial Intelligence and Machine Learning useful to deal with Generative AI models.

1.1 Artificial Intelligence and Generative AI

Artificial Intelligence is one of the most innovative fields in the last years. The development of scientific results in AI is happening very quickly, and every month there are new models that improve the models already developed.
There is a lot of definitions of Artificial Intelligence, depending on the field AI has to be applied. Artificial Intelligence can be defined as the the set of technologies and methods that are able to make tasks usually accomplished by human beings.
These tasks involve reasoning, learning, problem solving and others.
Given that the main focus of the thesis is on artificial intelligence application in data, I am going to describe the features that involve data-driven issues.
The division of AI subfields can be deep and very articulated, but it would go over the purpose of the thesis.
The main subfield of Artificial intelligence that I am going to explore is Generative AI (which will be analysed more in-depth in the next chapeter). Many talk about a paradigm

change in the last years, indeed traditional AI was mainly focused on pattern recognition and optimization, while Generative AI leverages large-scale neural networks to produce new and original text, images, music, video, tabular data, and others.
In the next section I am going to analize Machine Learning and its application at the data-driven problems.

1.2 Machine Learning and Deep learning

Machine Learning is a field that uses tools from mathematics, computer science and statistics. Taking the definition larger than we can, Machine Learning is about learning from experience in order to make predictions or compute tasks.

The main distinction that can be made in machine learning is the following.

- **Supervised learning :** the model uses labeled data to predict and to recognise patterns. Some algorithms of this kind of machine learning are linerar regression, decision trees, support vector machines and others. Later, in this section I am going to describe some simple topics in supervised learning.
- **Unsupervised learning :** In this case models learn from data that are not labeled. A classic instance of unsupervised learning is Clustering, where the data are partitioned in different groups (called clusters). For the purpose of this work, it is not necessary to go deep in the description of this models. However, a good reference which describes accurately unsupervised learning is [9].
- **Reinforcement learning :** The concept of this subfield of machine learning is often summarized as *"learning from mistakes"*. Indeed, the process is the following: an agent interacts with a system in order to learn an optimal policy. The agent observes the state s_t of the system and, based on its internal hidden state h_t, chooses an action a_t. After executing this action, the system transitions to a new state s_{t+1}, and the agent receives a reward r_t that indicates how effective the chosen action was. The agent then updates its hidden representation from h_t to h_{t+1}, incorporating

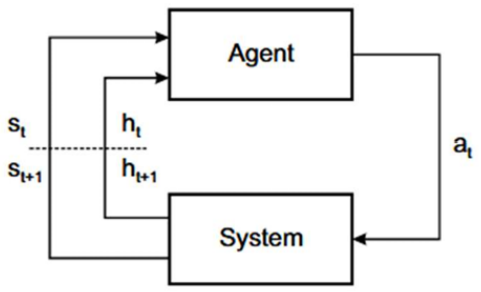


Figure 1.1: Reinforcement Learning

information from the new state and the received reward. Over time, by repeatedly observing s_t, taking actions a_t, and updating h_t, the agent learns a policy that maximizes the expected cumulative reward.

For the purpose of this work, I am going to describe the main features in the case when "experience" is meant to be data.
In particular for supervised learning issues (but it can be generalized), the main purpose is to minimize a function, which is called the **Loss Function**. If we define Y as the set of all labels, and Z as the set of all possible predictions, in most of the cases it happens that $Y = Z$.

Definition 1.2.1. *Given a Machine Learning problem, the Loss function is a mapping*

$$L : Y \times Z \to \mathbb{R}$$

that provides the difference between a true label and a predicted label.

Now I am going to describe the main feature of deep learning, that is a subset of Machine learning which studies the functioning of Neural Networks.
This part is very important, since deep learning is the foundation of a large part of Generative AI works.

1.2.1 Deep Learning algorithms

The main reference for this section is [13]. Now I am going to describe the **MLP** neural newtorks, since these are the one used in the diffusion models that have been explored in

the next chapters.
MLP(Multilayer perceptron), or Feedforward neural networks, are one of the main framework of deep learning. The name neural networks comes from the idea of comparing the functioning of these machine learning models to the functioning of human brain. Indeed a neural network is composed by neurons and edges, which are supposed to imitate the neurons and the synapses of a human brain. Furthermore, feedforward refers to the fact that the information flows in only one direction, from the input to the output.
Let there be L the number of hidden layers, the description of the layer is the following

$$\textbf{Input:} \quad \mathbf{x} \in \mathbb{R}^{d_0}$$

$$\textbf{Hidden layers:} \quad \begin{cases} \mathbf{h}^{(1)} = f^{(1)}(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}), \\ \mathbf{h}^{(2)} = f^{(2)}(W^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}), \\ \vdots \\ \mathbf{h}^{(L)} = f^{(L)}(W^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}), \end{cases}$$

$$\textbf{Output layer:} \quad \hat{\mathbf{y}} = f^{(L+1)}(W^{(L+1)}\mathbf{h}^{(L)} + \mathbf{b}^{(L+1)})$$

where f^l : is the activation function, $\hat{\mathbf{y}}$ is the output of the model, W^l is the weight matrix, and b^l is the bias vector, and $l \in \{1, .., L\}$ is a layer.
The most common activation function are usually:

$$Sigmoid: \quad \sigma(x) = \frac{1}{1 + e^{-x}}, \qquad \sigma(x) \in (0, 1)$$

$$Hyperbolic\ \ Tangent: \quad \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \tanh(x) \in (-1, 1)$$

$$Rectified\ Linear\ Unit(ReLU): \quad \mathrm{ReLU}(x) = \max(0, x), \qquad \mathrm{ReLU}(x) \in [0, \infty)$$

The purpose of the model is to find the parameters $\theta = \{W^l, b^l\}_{l=1}^{L+1}$, and the framework is the optimization of the loss function, which is

$$\theta^* = \arg\min_{\theta} \ \mathcal{L}(\theta)$$

and the most common optimization algorithm used in deep learning is Gradient Descent. This consists of iteration as

$$\theta^{(t+1)} = \theta^{(t)} - \eta \, \nabla_\theta \mathcal{L}(\theta^{(t)})$$

where η controls the step size on each iteration.

Deep learning is a huge field with several aspects to consider, carachterized by constant and continuous development. Nevertheless, for the sake of this work, it is not necessary to go in depth for every single instance of deep learning model, but rather figure out the main feature useful for generative AI. A good reference for other deep learning concepts is [13].

Chapter 2

Generative Models

Generative models are AI models that are able to generate text, images, sounds, videos and other things. Nowadays, generative models are gaining increadible popularity and are implemented by companies for industrial purposes. One main example is the diffusion of LLMs (Large Language Models), that are being used by companies and people for daily tasks.

An area where generative models have demonstrated remarkable success is in the generation of images and videos. Furthermore, these models can be used for the generation of synthetic dataset as tabular data, and this case will be deeply analyzed in the next chapter.

This chapter provides an overview of the leading generative model architectures currently shaping the field.

2.1 Generative adversarial network

Generative Adversarial Networks (GANs) constitute an innovative framework within deep generative modeling, which has recently attracted significant research interest. These kind of generative algorithms have gained great performances mainly for images generation.

First time I studied how GANs work, I was surprised and interested about the concept of competitive training among neural networks, that I am goin to describe now.
Let's provide an overview of the architecture and the mathematical structure of this generative models.

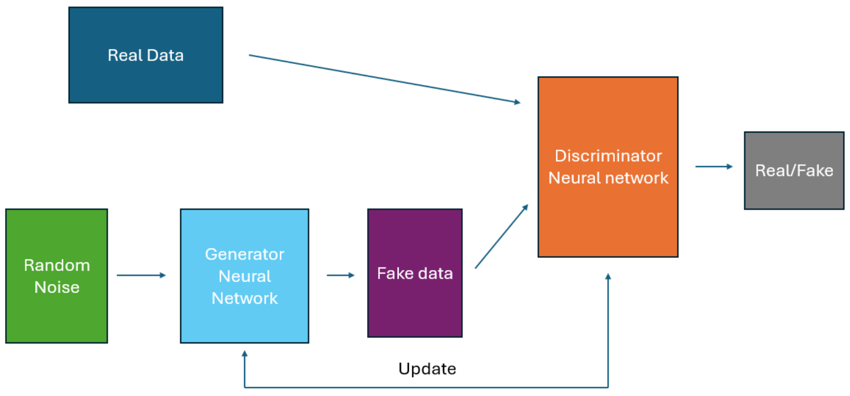


Figure 2.1: Generative adversarial network: a scheme of the architecture

The picture above describes accurately the main aspects of the GAN models. There are two neural networks (Generator and Discriminator) in a competitive game that lead to the training of both.

2.1.1 Mathematical structure

- It is defined a probability space $(\Omega, \mathcal{B}, \mu_{data})$, i.e. Ω is a sample space, $\mathcal{B}$ is a $\sigma-algebra$ on the sample space, and μ_{data} is a probability measure.

- **Generator** The generator is a measurable function

  $$G : \mathcal{Z} \to \Omega,$$

  where $\mathcal{Z}$ is a latent space equipped with a prior distribution μ_Z (e.g. a Gaussian or uniform distribution).

Therefore there is a distribution induced by the generator, which is called *pushforward*, and defined as it follows:

$$\mu_G = \mu_Z \circ G^{-1}.$$

- **Discriminator** The discriminator is a function

  $$D : \Omega \to [0,1],$$

  which represents the probability that a sample $x \in \Omega$ comes from the real distribution μ_{data} rather than μ_G.

The training process is a two-player zero-sum game between G and D, with value function

$$V(G, D) = \mathbb{E}_{x \sim \mu_{\text{data}}}\big[\log D(x)\big] + \mathbb{E}_{z \sim \mu_Z}\big[\log(1 - D(G(z)))\big].$$

The discriminator seeks to maximize $V(G, D)$, while the generator seeks to minimize it:

$$\min_G \max_D V(G, D).$$

Actually, this corresponds to the following loss functions:

$$\mathcal{L}_{Discriminator} = -\mathbb{E}_{x \sim \mu_{\text{data}}}[\log D(x)] - \mathbb{E}_{z \sim \mu_Z}[\log(1 - D(G(z)))]$$

for the discriminator, and

$$\mathcal{L}_{Generator} = -\mathbb{E}_{z \sim \mu_Z}[\log D(G(z))]$$

for the generator. This mathematic description defines formally what is happening in the model: the two neural networks (Generator and Discriminator) are training in competition, the Generator looks for producing data more difficult to regognise if they are fake or true, while the discriminator is training to understand if the data is fake or true.
CTGAN : Since this work is focused on tabular data, it is interesting to analyse a GAN model applied to tabular data. A remarkable model in the GAN studies is CTGAN (Conditional Tabular Generative Adversarial Network), and a good reference is [15].
The main characteristic of CTGAN is that this has been designed specifically for tabular data. In particular, the structure is similar but it has two main differences:

- The Generator network is replaced by a **Conditional Generator**;
- The loss function is modified to handle mixed data types (continuous and categorical) effectively.

Conditional Generator: The idea is to condition the generation process on specific discrete columns, allowing the model to learn complex dependencies between categorical and continuous variables. In this modified framework, generator becomes a conditional function

$$G : \mathcal{Z} \times \mathcal{C} \to \Omega,$$

where $\mathcal{C}$ represents the set of possible conditions (for example, categories in a discrete column). The generator takes as input both a latent vector $z \sim \mu_Z$ and a condition $c \in \mathcal{C}$, producing synthetic samples consistent with that condition.

The discriminator, in turn, also receives the condition c as an input, i.e.

$$D : \Omega \times \mathcal{C} \to [0,1],$$

and it learns to distinguish whether a sample (x, c) comes from the real data distribution or from the generator.

Modified loss function: The conditional adversarial loss is defined as

$$V(G, D) = \mathbb{E}_{(x,c)\sim\mu_{\text{data}}}\big[\log D(x, c)\big] + \mathbb{E}_{(z,c)\sim(\mu_Z,\mu_{\mathcal{C}})}\big[\log(1 - D(G(z, c), c))\big].$$

As in standard GANs, the discriminator maximizes this objective while the generator minimizes it:

$$\min_G \max_D V(G, D).$$

In addition, CTGAN introduces techniques to handle the specific challenges of tabular data:

- *Mode-specific normalization* for continuous columns, which allows better modeling of multi-modal continuous distributions;
- *Conditional sampling* during training, ensuring that rare categories are sufficiently represented and the generator learns their distributions accurately.

These modifications make CTGAN particularly effective in generating high-quality synthetic tabular data that preserves both marginal and conditional dependencies among variables.

2.2 Variational autoencoder

The second generative model I am going to describe in this chapter is the variational autoencoder. This kind of generative models have been introduced in 2014 in the paper [11] and then they have gained popularity in various field of generative AI.
The main references for this section will be [11].

The main parts of the architecture of a Variational autoencoder (or VAE) are three (in addition to input and output):

- **Encoder**: it receives data and basically compresses them into a lower-dimensional representation. In a Variational Autoencoder (VAE), instead of mapping inputs to a fixed point, the encoder learns the parameters of a probability distribution, typically a Gaussian, in the latent space

$$q_\phi(z|x) = \mathcal{N}(z;\, \mu_\phi(x),\, \Sigma_\phi(x))$$

  where $\mu_\phi(x)$ and $\Sigma_\phi(x)$ are the mean and covariance predicted by the encoder network. From this distribution, latent variables are sampled and later used by the decoder to reconstruct the input data. This probabilistic method allows the model to learn smooth and continuous latent representations, producing realistic data generation.

- **Latent space**: The latent space is a continuous space in which the encoded representations of the data are mapped. Each point in this space corresponds to a possible “code” that captures the essential features and variations of the input data.
  During training, the VAE learns a latent distribution that approximates the underlying data distribution. A key mechanism enabling this process is the reparameterization trick, which allows sampling from the latent space in a differentiable

way, making it possible to train the entire model through backpropagation. To enable backpropagation through the sampling process, the reparameterization trick is applied:

$$z = \mu_\phi(x) + \Sigma_\phi^{1/2}(x) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

This formulation allows the gradient to flow through μ_ϕ and Σ_ϕ while maintaining stochasticity via ϵ.

- **Decoder**: The decoder performs the inverse operation of the encoder: starting from a point (or vector) in the latent space, it reconstructs an output that should resemble the original input as closely as possible. In this sense, the decoder translates the compressed representation back into the data space.

Loss function : During the training phase, the Variational Autoencoder aims to learn both how to accurately reconstruct the input data and how to organize the latent space in a structured way. This is achieved by minimizing a specific loss function, known as the ***Evidence Lower Bound* (ELBO)**.

The ELBO combines two complementary objectives:

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{\mathrm{KL}}\big(q_\phi(z|x) \,\|\, p(z)\big)$$

The first term,

$$\mathbb{E}_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)],$$

is the **reconstruction loss**: it measures how well the decoder can reproduce the input data starting from the latent variables z. A higher value of this term indicates that the model is successfully reconstructing the original examples.

The second term,

$$D_{\mathrm{KL}}\big(q_\phi(z|x) \,\|\, p(z)\big),$$

is a **regularization term** that enforces the latent distribution learned by the encoder, $q_\phi(z|x)$, to remain close to a chosen prior $p(z)$, which is typically a standard Gaussian $\mathcal{N}(0, I)$. This constraint prevents the model from overfitting and encourages the latent space to be continuous and smooth.

When both terms are combined, the overall objective balances **reconstruction accuracy** and **latent space regularity**. In practice, the loss function is minimized by taking the negative ELBO:

$$\mathcal{L}_{\text{VAE}} = -\mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] + D_{\text{KL}}\big(q_\phi(z|x) \,\|\, p(z)\big)$$

If both $q_\phi(z|x)$ and $p(z)$ are Gaussian, the Kullback–Leibler divergence can be computed analytically as:

$$D_{\text{KL}}\big(q_\phi(z|x) \,\|\, p(z)\big) = \frac{1}{2}\sum_{j=1}^{d}\Big(\mu_j^2 + \sigma_j^2 - \log\sigma_j^2 - 1\Big)$$

This analytical form makes the optimization efficient and numerically stable. By minimizing this loss, the VAE learns to generate new data that are not only realistic but also smoothly distributed in the latent space.

Tabular Variational Autoencoder (TVAE)

The *Tabular Variational Autoencoder (TVAE)* extends the Variational Autoencoder framework to tabular datasets that include both continuous and categorical features. Unlike the standard VAE, which assumes a single continuous distribution for all variables, the TVAE adapts the reconstruction process according to the data type of each feature, allowing for a more accurate modeling of mixed data.

During training, the TVAE minimizes a modified Evidence Lower Bound (ELBO):

$$\mathcal{L}_{TVAE} = -\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + D_{KL}(q_\phi(z|x)\|p(z))$$

where $q_\phi(z|x)$ denotes the encoder distribution, $p_\theta(x|z)$ is the decoder likelihood, and $p(z) \sim \mathcal{N}(0, I)$ represents the prior over the latent variables.

The first term, $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$, corresponds to the *reconstruction component*, whose expression depends on the type of variable being modeled:

- For continuous variables, a Gaussian likelihood is used, producing a mean-squared reconstruction penalty:

$$\log p_\theta(x_j|z) = -\frac{1}{2\sigma_j^2}(x_j - \hat{x}_j)^2$$

- For categorical variables, a cross-entropy term is employed to capture the discrete probability distribution:

$$\log p_\theta(x_j|z) = \sum_k x_{jk} \log \hat{x}_{jk}$$

By combining both terms, the overall objective becomes:

$$\mathcal{L}_{TVAE} = -\sum_{j\in\text{cont}} \mathbb{E}_{q_\phi(z|x)}\left[\frac{(x_j - \hat{x}_j)^2}{2\sigma_j^2}\right] - \sum_{j\in\text{cat}} \mathbb{E}_{q_\phi(z|x)}\left[\sum_k x_{jk}\log \hat{x}_{jk}\right] + D_{KL}(q_\phi(z|x)\|p(z))$$

The tabular data generative model maintain smooth latent representations while accurately reconstructing both numerical and categorical features. In this way, there is an improvement of the generation of realistic synthetic tabular data.

2.3 Diffusion Models

Diffusion models are Generative AI models that have been introduced the first time in the paper [4]. For some years, despite its potential, the idea was mostly ignored for years in favor of GANs and VAEs.
Diffusion models have regained the attention of the scientific community with the pubblication of [5], which defines a fundamental framework for this thesis: DDPM (Denoising Diffusion Probabilistic Models). In this section I will describe the mathematical structure, that will be useful in the next chapter in the case of tabular data.
Mathematical Structure: The most important feature of DDPM is that the generative process is defined through two complementary Markov chains: a forward process and a reverse process.

- **Forward process (diffusion process)**: in this first step, a sample x_0 that comes from the data distribution $\omega = \omega(x_0)$ is modified adding noise, implementing the following process:

$$\omega(x_{1:T} \mid x_0) = \prod_{t=1}^{T} \omega(x_t \mid x_{t-1})$$

where x_0 is an initial sample taken from the data distribution $\omega(x_0)$.
The numbers $\gamma_t, t = 1, ..., T$ are the variances of the phase $x_{t-1} \to x_t$, which means mathematically that

$$Var(\omega(x_t|x_{t-1})) = \gamma_t I$$

The interpretation of these variances is as follows: The parameters γ_t control the amount of noise added at each step of the forward process. Smaller values of γ_t make the transition slower and more stable, whereas larger values lead to a noisier process but require fewer steps.

- **Reverse process**: there is a gradual denoising process

$$p_\lambda(x_{0:T}) = \prod_{t=1}^{T} p_\lambda(x_{t-1}|x_t)$$

where the dependency on a variable λ is due to the fact that the distributions $p_\lambda(x_{t-1}|x_t)$ is not known and therefore it is approximated by a neural network. The purpose of the model is to optimize the bound on log likelihood, that means

$$\mathbb{E}_{\omega(x_0)}[\mathcal{L}_0 - \mathcal{L}_T - \sum_{t=2}^{T} \mathcal{L}_t] \leq log[\omega(x_0)] \tag{2.1}$$

$$\mathcal{L}_0 = log[p_\lambda(x_0|x_1)] \tag{2.2}$$

$$\mathcal{L}_t = K\mathcal{L}[\omega(x_T|x_0)|\omega(x_T)] \tag{2.3}$$

$$\mathcal{L}_T = K\mathcal{L}[\omega(x_{t-1}|x_t x_0)|p_\lambda(x_{t-1}|x_t)] \tag{2.4}$$

where L is the loss function. The term $\mathcal{L}_T$ represents the Kullback–Leibler divergence between the true posterior distribution $\omega(x_{t-1}|x_t, x_0)$ and its neural approximation $p_\lambda(x_{t-1}|x_t)$. Since $\omega(x_{t-1}|x_t, x_0)$ is analytically tractable and depends on a known Gaussian noise ϵ, the optimization can be simplified by minimizing the mean squared error between the real noise and the one predicted by the model:

$$\mathcal{L}_{simple} = \mathbb{E}_{t,\, x_0,\, \epsilon}\big[\|\epsilon - \epsilon_\lambda(x_t, t)\|^2\big],$$

where $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon$ and $\bar{\alpha}_t = \prod_{i=1}^{t}(1 - \gamma_i)$. This simplified form is equivalent to maximizing the variational bound on the log-likelihood and is commonly used for training DDPMs in practice.

Therefore, this is a description of the foundationd of diffusion models. Other information about the mathematical structure can be found in [5]. Instead, in the next chapter, that will be completely dedicated to diffusion models for tabular data, there will be a complete description of the structure of two diffusion models: TabDDPM and FinDiff.

2.4 Copula-based Generative models

In this section I am going to provide a short review of copula-based models, since those have been used in the computational cases for some reason that I am going to underline in the Chapter 4. Therefore, for the sake of completeness, I have summarized the fundamental theoretical aspects of these generative models. The main reference for copula-based models is [10].

Firstly, it is important to provide a formal definition of what a copula is.
Let $U_1, ..., U_d \sim \mathcal{U}(0,1)$ be uniformly distributed random variables

Definition 2.4.1 (Copula)**.** *A Copula is a joint distribution*

$$C(u_1, \dots, u_d) = \mathbb{P}(U_1 \leq u_1, \dots, U_d \leq u_d), \quad (u_1, \dots, u_d) \in [0,1]^d$$

which has the following properties
(i) For each $i \in \{1, ..., d\}$ *, the function*

$$u_i \;\mapsto\; C(u_1, \dots, u_i, \dots, u_d)$$

is non-decreasing.
(ii) 2. The ith marginal distribution is obtained by setting $u_j = 1$ *for* $j \neq i$ *and since it is uniformly distributed*

$$C(1, \dots, 1, u_i, 1, \dots, 1) = u_i$$

(iii) For $a_i \leq b_i$*,*

$$\mathbb{P}(U_1 \in [a_1, b_1], \dots, U_d \in [a_d, b_d])$$

must be non-negative.

Teorema 2.4.1 (Sklar). *Let F be a joint cumulative distribution function on $\mathbb{R}^d$ with marginal distribution functions $F_1, \dots, F_d$. Then there exists a copula $C : [0,1]^d \to [0,1]$ such that, for all $(x_1, \dots, x_d) \in \mathbb{R}^d$,*

$$F(x_1, \dots, x_d) = C\big(F_1(x_1), \dots, F_d(x_d)\big).$$

If the marginals $F_1, \dots, F_d$ are continuous, then C is unique; otherwise, C is uniquely determined only on the set $\mathrm{Ran}(F_1) \times \cdots \times \mathrm{Ran}(F_d)$*. Conversely, if C is a copula and $F_1, \dots, F_d$ are distribution functions, then the function F defined as above is a multivariate distribution function with marginals $F_1, \dots, F_d$.*

Proof. **[Existence]**
Let F be a joint cdf on $\mathbb{R}^d$ with marginals $F_1, \dots, F_d$. For $u = (u_1, \dots, u_d) \in [0,1]^d$, define the (left–continuous) generalized inverse

$$F_i^{-1}(u_i) \;:=\; \inf\{x \in \mathbb{R} :\, F_i(x) \geq u_i\}, \qquad i = 1, \dots, d,$$

and define

$$C(u_1, \dots, u_d) \;:=\; F\big(F_1^{-1}(u_1), \dots, F_d^{-1}(u_d)\big).$$

Then $C : [0,1]^d \to [0,1]$ is grounded and d-increasing (being the composition of a cdf with nondecreasing arguments), hence a cdf on $[0,1]^d$. Moreover, for each i and $u \in [0,1]$,

$$C(1, \dots ,1, u, 1, \dots ,1) = F\big(\infty, \dots, \infty, F_i^{-1}(u), \infty, \dots, \infty\big) = F_i\big(F_i^{-1}(u)\big) \geq u,$$

while right–continuity and monotonicity of F_i imply also $C(1, \dots ,1, u, 1, \dots ,1) \leq u$; hence the i-th margin of C is u. Therefore C is a copula. By construction we have, for every $x = (x_1, \dots, x_d) \in \mathbb{R}^d$,

$$F(x_1, \dots, x_d) = C\big(F_1(x_1), \dots, F_d(x_d)\big),$$

because $F_i^{-1}\big(F_i(x_i)\big) \leq x_i$ and F is increasing in each coordinate, while taking limits from the right gives the reverse inequality.
[Uniqueness]

If all marginals $F_1, \dots, F_d$ are continuous, then $F_i^{-1}(F_i(x)) = x$ and $F_i(F_i^{-1}(u)) = u$ for all x, u; hence the above C is the unique copula satisfying $F(x) = C(F_1(x_1), \dots, F_d(x_d))$. If some marginal is not continuous, any copula representation is uniquely determined on $\mathrm{Ran}(F_1) \times \cdots \times \mathrm{Ran}(F_d)$, and possibly differs elsewhere.

[Converse]

Conversely, let C be a copula and $F_1, \dots, F_d$ cdfs on $\mathbb{R}$. Define

$$F(x_1, \dots, x_d) \; := \; C(F_1(x_1), \dots, F_d(x_d)).$$

Then F is a cdf on $\mathbb{R}^d$ whose marginals are exactly $F_1, \dots, F_d$ (obtained by setting all but one coordinate to $+\infty$), completing the proof.

$\square$

Now that we have introduced the definition of a copula and the main theoretical result concerning copulas, we turn to their role in generative AI. Specifically, we discuss how copulas can be used within generative algorithms to model complex dependencies among variables and to construct flexible synthetic data generators.

Once the theoretical foundation has been established, it is possible to highlight the role of copulas in generative algorithms. The key idea is that copulas provide a principled way to separate marginal distributions from the dependence structure among variables. In practice, this means that each marginal distribution F_i can be estimated independently, using either parametric or non-parametric methods, while the copula C is used to reconstruct the joint distribution. This decomposition is particularly valuable in tabular generative modeling, since real-world datasets often exhibit heterogeneous distributions across features: some variables are heavy-tailed, others are approximately Gaussian, and categorical variables can be embedded into discrete distributions.

Generative algorithms use this property by first transforming observed data into uniformly distributed random variables on [0,1] via the probability integral transform. A copula function is then fitted to capture the dependencies between these transformed variables. Finally, synthetic samples are generated by simulating from the copula and applying the inverse transforms of the estimated marginals. This process ensures that both the univariate properties of each variable and the multivariate dependence structure are preserved in the generated data.

In recent years, copula-based models have also been combined with modern machine learning approaches, such as normalizing flows and neural networks, to enhance their expressive power. These hybrid frameworks leverage the interpretability and flexibility of copulas while benefiting from the scalability and representational capacity of deep learning methods. As such, copulas offer a robust and versatile tool for constructing synthetic datasets, making them an important reference point when evaluating more recent AI-based generative models, such as diffusion models or GAN variants, in the tabular domain.

Chapter 3

Diffusion Models for tabular data

3.1 Tabular data

Tabular data represent one of the most common and fundamental forms of structured data. They are organized into rows, corresponding to individual observations or records, and columns, which represent variables or attributes.
This simple yet powerful format enables consistent storage, processing, and interpretation of information.

Tabular data are very important for business applications and, in particular, they are widely used by companies, banks, and institutions. Indeed, these kind of data are easier to manage and analize, if compared to other kind of data (for instance images, videos).
Since the topic of this thesis is concerning diffusion models for tabular data, I am introducing now the main aspects of tabular data. First, I will describe the general aspects of tabular data. Then, in the following section, I will address the specific case of how tabular data are handled within diffusion models.
For a more solid foundation on machine learning applied to tabular data, the book [6] can

be consulted.

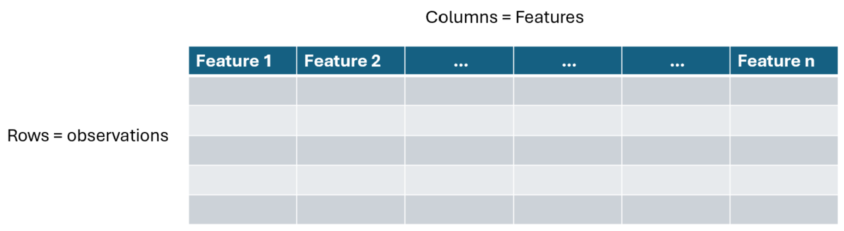


Figure 3.1: Tabular data

As it can be seen from the image, a general tabular data is composed by rows and columns:

- Columns represent the variables (also called features or attributes) that describe certain characteristics of the observations.
- Rows correspond to the individual observations (also called records or instances), each one providing a specific value for every variable.

This format of data is the most used since it allows an efficient storage, querying and application of Machine Learning algorithm.

One of the main issues to deal with is the missing and noisy data: for a comprehensive discussion on how to address these challenges, [6] provides a valuable reference. In this work, however, the focus will be placed specifically on the treatment of such issues within the framework of diffusion models.

3.2 TabDDPM

The main reference for this section is [1].
The theoretical foundations of TabDDPM are primarily based on the mathematical framework introduced in Chapter 2 for DDPM. In particular, TabDDPM inherits the general diffusion-based structure, while specializing to the case where the underlying distribution

is assumed to be Gaussian in the case of numerical features, and Multinomial diffusion in the case of categorical and binary features.

3.2.1 Mathematical Structure:

In order to deal with both types of features for the datasets (numerical and categorical), there needs to choose a vector of variance $(\gamma_1, ..., \gamma_T)$, and then the probability distribution are the following.

Multinomial diffusion

When we have to deal with categorical of binary data, i.e. $x_t \in \{0{,}1\}^K \quad \forall t$, and the distribution are

$$\omega(x_t \mid x_{t-1}) \sim Cat(x_t; (1-\gamma_t)x_{t-1} + \gamma_t/K) \tag{3.1}$$

$$\omega(x_T) \sim Cat(x_T; 1/K) \tag{3.2}$$

$$\omega_\theta(x_t \mid x_0) \sim Cat(x_{t-1}; \bar{\alpha}_t x_0 + (1-\bar{\alpha}_t)/K)) \tag{3.3}$$

$$\alpha_t := 1 - \gamma_t, \quad \bar{\alpha}_t := \prod_{i \le t} \gamma_i \tag{3.4}$$

Gaussian diffusion

In this case, the data $x_t \in \mathbb{R}^n \quad \forall t$, and the distribution are

$$x_t = \sqrt{1-\gamma_t}\, x_{t-1} + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \gamma_t I) \tag{3.5}$$

$$\omega(x_t \mid x_{t-1}) \sim \mathcal{N}\Big(x_t; \sqrt{1-\gamma_t}\, x_{t-1},\, \gamma_t I\Big) \tag{3.6}$$

$$\omega(x_T) \sim \mathcal{N}(x_T; 0, I) \tag{3.7}$$

$$p_\lambda(x_{t-1} \mid x_t) \sim \mathcal{N}(x_{t-1}; \mu_\lambda(x_t, t), \Sigma_\lambda(x_t, t)) \tag{3.8}$$

The working of the diffusion and reverse process have already been described accurately in the Chapter 2, section 3. Furthermore , it is important to underline that the parameter λ is the parameter that the neural networks need to learn, and this idea is the same for TabDDPM and FinDiff.

Loss function The reverse process in TabDDPM is composed by a multi-layer neural network, and the model has the purpose to minimize the total loss function

$$\mathcal{L}_t^{\text{Total}} = \mathcal{L}_t^{\text{Gaussian}} + \frac{\sum_{i \le C} \mathcal{L}_t^i}{C}$$

where $\mathcal{L}_t^{\text{Gaussian}}$ denotes the mean-squared error minimization term for the Gaussian component, and $\mathcal{L}_t^i$ represents the loss defined in the previous formula.

The Gaussian loss term corresponds to the denoising objective used in standard diffusion models. In particular, for each timestep t, the model predicts the noise component $\epsilon_\lambda(x_t, t)$ added to the data sample, and the Gaussian loss is defined as

$$\mathcal{L}_t^{\text{Gaussian}} = \mathbb{E}_{x_0,\epsilon,t}\Big[\big\|\epsilon - \epsilon_\lambda(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon, t)\big\|_2^2\Big],$$

where $\epsilon \sim \mathcal{N}(0, I)$ and $\bar{\alpha}_t = \prod_{i \le t}(1-\gamma_i)$. This term corresponds to a mean squared error between the true noise ϵ and the noise predicted by the neural network $\epsilon_\lambda(\cdot, t)$. Minimizing this loss encourages the model to accurately learn the reverse diffusion process for continuous (Gaussian) features.

Neural network architecture. The reverse process $p_\lambda(x_{t-1} \mid x_t)$ is parameterized by a multi-layer perceptron (MLP), which approximates the denoising mapping

$$f_\lambda(x_t, t) = W_L\sigma(W_{L-1}\sigma(\dots\sigma(W_1[x_t \,\|\, e_t] + b_1)\dots) + b_{L-1}) + b_L,$$

where $\sigma(\cdot)$ denotes a non-linear activation (e.g. ReLU or GELU), e_t is a time embedding, and $\|$ indicates concatenation. The output represents either the estimated noise $\epsilon_\lambda(x_t, t)$ for continuous variables or the logits for categorical features.

Optimization. The parameters λ of the MLP are optimized via gradient descent:

$$\lambda \leftarrow \lambda - \eta\,\nabla_\lambda \mathcal{L}_t^{\text{Total}},$$

with learning rate $\eta > 0$. In practice, the expectation is approximated through minibatch stochastic optimization (Adam or AdamW).

Sampling. After training, new samples are obtained by iteratively sampling

$$x_{t-1} \sim p_\lambda(x_{t-1} \mid x_t), \quad t = T, T-1, \dots, 1,$$

starting from $x_T \sim \mathcal{N}(0, I)$ for continuous variables and a uniform categorical distribution for discrete ones.

3.3 FinDiff

The main reference for this section is [2]. FinDiff (Financial Diffusion) is a generative modeling framework specifically designed for financial synthetic tabular data generation. While diffusion-based generative models have recently demonstrated remarkable performance in image, text, and multimodal tasks, their direct application to tabular data is still a field with continuous development and deep research due to the heterogeneous nature of financial datasets.

In order to deal with these challenges, FinDiff extends the diffusion paradigm by incorporating domain-aware adaptations tailored to financial data distributions. The model is designed to preserve statistical fidelity, ensuring that the synthetic data reflects the marginal and joint distributions of real datasets, while also providing privacy guarantees by reducing the risk of sensitive information leakage.

Synthetic data generated by FinDiff enables a range of remarkable applications, including credit risk modeling, fraud detection, customer segmentation, and stress testing, where access to large-scale high-quality data is often restricted by confidentiality and regulatory concerns. Overall, FinDiff contributes to both practical utility and responsible data sharing in financial research and industry applications.

3.3.1 Mathematical structure

The model handles a mixture of numerical and categorical variables. Figure 4.11 outlines the overall FinDiff architecture.

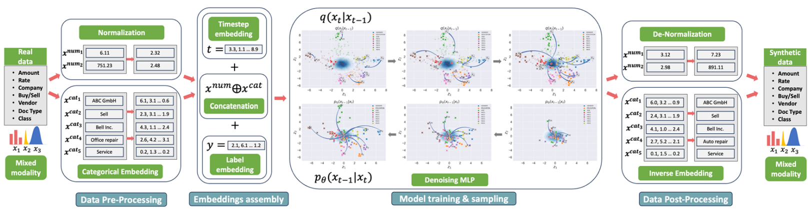


Figure 3.2: FinDiff architecture (Source: [2])

The main steps of the generative process are as follows:

- **Data pre-processing phase:** Categorical variables are converted into learnable embeddings, while numerical ones are standardized or normalized to zero mean and unit variance.
- **Embedding assembly phase:** The pre-processed numerical and categorical representations are concatenated and combined with time and label embeddings, forming the complete input vector for the diffusion model.
- **Training phase:** The model learns to reverse the diffusion process by minimizing a loss function defined over both numerical and categorical components.
- **Post-processing phase:** Generated categorical outputs are mapped to their closest valid category, while numerical values are denormalized back to the original scale.

Mathematical formulation

Let the tabular sample at step t be denoted as $\mathbf{x}_t = (\mathbf{x}_t^{\text{num}}, \mathbf{x}_t^{\text{cat}})$, where $\mathbf{x}_t^{\text{num}}$ and $\mathbf{x}_t^{\text{cat}}$ represent numerical and categorical components, respectively. The diffusion process is applied separately to these two types of data, following different noise dynamics.

Numerical features. For the continuous attributes, the forward diffusion adds Gaussian noise according to a variance schedule $\{\beta_t\}_{t=1}^{T}$:

$$q(\mathbf{x}_t^{\text{num}} \mid \mathbf{x}_0^{\text{num}}) = \sqrt{\bar{\alpha}_t}\, \mathbf{x}_0^{\text{num}} + \sqrt{1-\bar{\alpha}_t}\, \boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \sim \mathcal{N}(0, I), \tag{3.9}$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.

During training, the neural network $\boldsymbol{\varepsilon}_\theta$ learns to predict the added noise $\boldsymbol{\varepsilon}$, minimizing the mean squared error (MSE):

$$\mathcal{L}_{\text{num}} = \mathbb{E}_{t,\mathbf{x}_0^{\text{num}},\boldsymbol{\varepsilon}}\Big[\big\|\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_\lambda(\mathbf{x}_t^{\text{num}}, t, \mathbf{c})\big\|_2^2\Big], \tag{3.10}$$

where $\mathbf{c}$ denotes optional conditioning information such as labels or other metadata.

Categorical features. In this case,the diffusion process is defined over categorical states. Let $z_0 \in \{1, \dots, K\}$ be a categorical variable with K possible classes. At each step t, a transition matrix gradually corrupts the original state toward a nearly uniform distribution:

$$q(z_t = k \mid z_{t-1} = j) = (1 - \beta_t)\, \mathbb{1}[k = j] + \beta_t\, \pi_k, \qquad \pi_k \approx \tfrac{1}{K}. \tag{3.11}$$

The model learns to recover the original (clean) category from the noised version z_t by predicting a categorical distribution:

$$p_\lambda(z_0 \mid z_t, t, \mathbf{c}) = \mathrm{Cat}\big(\mathrm{softmax}(\mathbf{g}_\lambda(\mathbf{e}(z_t), t, \mathbf{c}))\big)\,, \tag{3.12}$$

where $\mathbf{e}(z_t)$ is the embedding of the discrete variable at time t. The corresponding training loss is the negative log-likelihood (equivalently, a cross-entropy loss):

$$\mathcal{L}_{\text{cat}} = \mathbb{E}_{t, z_0, z_t \sim q(\cdot \mid z_0)}\big[-\log p_\lambda(z_0 \mid z_t, t, \mathbf{c})\big]. \tag{3.13}$$

Joint training objective. Since the model handles both data types simultaneously, the total loss combines the numerical and categorical components:

$$\mathcal{L}_{\text{total}} = \lambda_{\text{num}}\, \mathcal{L}_{\text{num}} + \lambda_{\text{cat}}\, \mathcal{L}_{\text{cat}} + \lambda_{\text{reg}}\, \mathcal{R}(\lambda), \tag{3.14}$$

where λ_{num} and λ_{cat} balance the relative contributions of each loss, and $\mathcal{R}(\lambda)$ denotes an optional regularization term (e.g., weight decay).

Reconstruction phase. During sampling, the model iteratively reverses the diffusion process:

- For numerical features, the model progressively removes Gaussian noise using the estimated $\boldsymbol{\varepsilon}_\lambda$.

- For categorical features, the model samples from the predicted categorical distributions $p_\lambda(z_{t-1} \mid z_t, t, \mathbf{c})$ until reaching $t = 0$.

Finally, numerical data are denormalized, and categorical embeddings are mapped back to their most probable discrete values.

3.4 TabDDPM and FinDiff: A theoretical comparison

In this section I am going to underline the main difference between the two models from a theoretical point of view. Indeed, these differences are also reflected in the computational results, which will be analyzed in the next chapter.

The differences between the two diffusion models are clear from the previous sections describing their mathematical properties. The present section is therefore devoted to synthesizing and tabulating these differences, and it provides a concise reference to be kept in mind for the subsequent chapter, where these differences will be

Aspect	**TabDDPM**	**FinDiff**
Application domain	Generic (mixed tabular data)	Specific for mixed financial data
Categorical data	One-hot encoding + multinomial	Embedding encoding
Numerical data	Gaussian diffusion	Gaussian diffusion with advanced scaling
Architecture	Base DDPM model	Feedforward NN with semantic embeddings
Focus	Privacy & validity	Privacy + Utility + Fidelity
Scalability	Limited by one-hot	More scalable with embeddings

Table 3.1: Comparison among TabDDPM and FinDiff

Chapter 4

Computational Results

All the tests have been made on Google Colab, using the T4GPU, since the generative models need high computational power.
In the following sections of this chapter I am describing the datasets and the results, focusing on the computational aspects with a view on the theoretical aspects already described in the Chapter 3 and if the computational results align with what was expected from the mathematical theory.

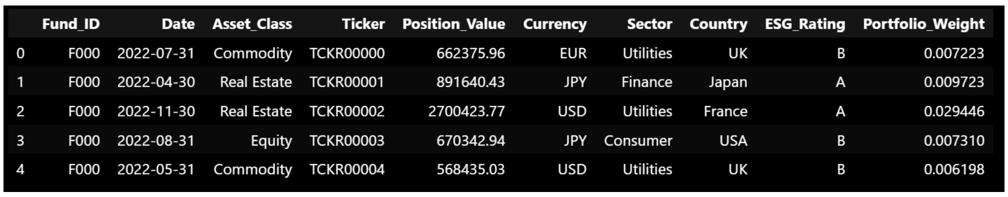
4.1 Datasets

In this section, I am going to describe the main features of the datasets used to evaluate the models.
Two different datasets were employed in this work in order to test the robustness of generative models across heterogeneous data structures. , one completely numerical and one mixed numerical and categoricals.

4.1.1 First Dataset: Financial portfolio

The first dataset employed in this work is a Financial portfolio dataset, randomly generated for this case study with ChatGPT. The dataset consists of 5,000 observations and 10 variables, providing detailed information on the composition of an investment portfolio.

It includes identifiers such as fund ID, date, asset class, ticker, position value, and currency. In addition, descriptive attributes are provided, including sector, country, and ESG rating, along with the portfolio weight of each position. In the picture below, there can be seen the first 5 rows of the dataset useful to understand the structure of the dataset. Overall, this is a classical instance of tabular financial data and combines both numerical



	Fund_ID	Date	Asset_Class	Ticker	Position_Value	Currency	Sector	Country	ESG_Rating	Portfolio_Weight
0	F000	2022-07-31	Commodity	TCKR00000	662375.96	EUR	Utilities	UK	B	0.007223
1	F000	2022-04-30	Real Estate	TCKR00001	891640.43	JPY	Finance	Japan	A	0.009723
2	F000	2022-11-30	Real Estate	TCKR00002	2700423.77	USD	Utilities	France	A	0.029446
3	F000	2022-08-31	Equity	TCKR00003	670342.94	JPY	Consumer	USA	B	0.007310
4	F000	2022-05-31	Commodity	TCKR00004	568435.03	USD	Utilities	UK	B	0.006198

Figure 4.1: Portfolio Dataset

and categorical variables, making it suitable for portfolio allocation and diversification analysis.

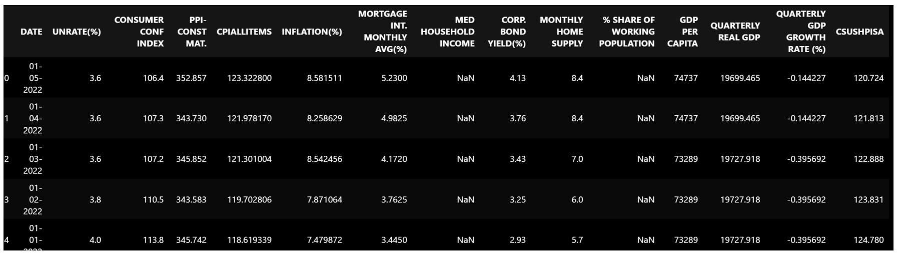
One of the main question someone can ask is what is the purpose of using generative models to create synthetic dataset. These methods are able to capture the dependencies between heterogeneous features while preserving the statistical properties of financial attributes. The generation of synthetic data in this context can be useful for multiple purposes:

- **Data augmentation:** enrich limited datasets by generating additional plausible observations.
- **Privacy preservation:** allow the sharing of realistic financial data without disclosing sensitive information.
- **Stress testing and scenario analysis:** simulate portfolios under extreme or unobserved market conditions.
- **Benchmarking and model evaluation:** provide controlled datasets for validating analytical methods and machine learning models.

Overall, synthetic data generation represents a valuable tool for research, risk management, and practical applications in portfolio analysis.

4.1.2 Second Dataset: Macroeconomics

The second dataset employed for the generation of synthetic data is a macroeconomics dataset about US macroeconomics factors that can be taken from the website Kaggle or from other datasets repository, since macroeconomics data are are publicly available.
This contains main macroeconomis measurements as inflation, mortage interests, bond yields and others. In the next picture, there can be visualized the first 5 rows to see how the dataset is composed.



	DATE	UNRATE(%)	CONSUMER CONF INDEX	PPI-CONST MAT.	CPIALLITEMS	INFLATION(%)	MORTGAGE INT. MONTHLY AVG(%)	MED HOUSEHOLD INCOME	CORP. BOND YIELD(%)	MONTHLY HOME SUPPLY	% SHARE OF WORKING POPULATION	GDP PER CAPITA	QUARTERLY REAL GDP	QUARTERLY GDP GROWTH RATE (%)	CSUSHPISA
0	01-05-2022	3.6	106.4	352.857	123.322800	8.581511	5.2300	NaN	4.13	8.4	NaN	74737	19699.465	-0.144227	120.724
1	01-04-2022	3.6	107.3	343.730	121.978170	8.258629	4.9825	NaN	3.76	8.4	NaN	74737	19699.465	-0.144227	121.813
2	01-03-2022	3.6	107.2	345.852	121.301004	8.542456	4.1720	NaN	3.43	7.0	NaN	73289	19727.918	-0.395692	122.888
3	01-02-2022	3.8	110.5	343.583	119.702806	7.871064	3.7625	NaN	3.25	6.0	NaN	73289	19727.918	-0.395692	123.831
4	01-01-	4.0	113.8	345.742	118.619339	7.479872	3.4450	NaN	2.93	5.7	NaN	73289	19727.918	-0.395692	124.780

Figure 4.2: Macroeconomics dataset

The dataset is composed by 242 rows and 15 columns (i.e. the features).
These variables determine the economic conditions and are useful to analyze trends, they are collected from 2003 to 2022.

4.2 Evaluation metrics

In both TabDDPM and FinDiff, data quality is not defined intrinsically by the models themselves, but rather by standardized metrics for synthetic data evaluation, as implemented in libraries such as `SDMetrics`.
In particular, data quality refers to the extent to which the synthetic data reproduces the statistical properties of the real data.

The **Column Shapes** metric evaluates how well the distribution of each individual column in the synthetic data matches the corresponding distribution in the real dataset. For numerical variables, this comparison is commonly performed using the *Kolmogorov–Smirnov Complement (KSComplement)*, while categorical variables are assessed through the Total Variation Distance. Each column receives a score between 0 and 1, where values

closer to 1 indicate a higher similarity between real and synthetic distributions. From a mathematical point of view, the score is obtained as it follows:
If z_i is the i-th column, the column fidelity is defined as it follows:

$$\lambda_i = \begin{cases} 1 - KS(z_i, s_i) & \text{if } i \text{ is numerical} \\ 1 - \frac{1}{2}TVD(z_i.s_i) & \text{if } d \text{ is categorical} \end{cases} \tag{4.1}$$

where

$$KS(z, s) = \sup_x |F_z(x) - G_s(x)|$$

with F and G are empirical cumulative distribution functions respectively of z (the original data) and s (the synthetic dataset).

The overall *Average Score* is then computed as the mean of all per-column scores, providing a global indicator of how accurately the marginal distributions are reproduced by the generative model. For instance, an average score of 0.90 suggests that, on average, the column-wise distributions in the synthetic data are highly consistent with those of the real dataset.

Another important aspect that needs to be deeply analyzed is the preservation of **Privacy**.

To assess privacy preservation in synthetic data, we compute the **Nearest Neighbor Distance (NND)** between synthetic and real records. In this approach, each synthetic sample is encoded and scaled consistently with the real data, and its distance to the closest real record is measured using a nearest-neighbor search. The distribution of these distances provides an indicator of potential privacy risks:

- **Average Distance:** A higher average distance suggests that synthetic records are sufficiently different from the real ones, thus reducing the likelihood of direct memorization.
- **Minimum Distance:** Very low or zero distances may indicate that a synthetic record exactly reproduces a real one, which represents a potential privacy breach.
- **Suspicious Records:** The count of synthetic samples with distance below a small threshold (e.g., 10^{-6}) highlights possible memorization cases.

4.3 Results and Comparison

4.3.1 Results for first dataset

The results are analyzed considering the metrics defined in the section above. Therefore, I am going to summarize the data quality and privacy concerns for the portfolio dataset.

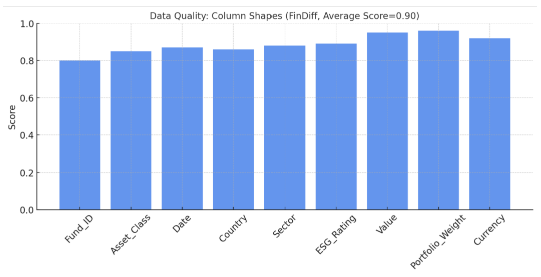


Figure 4.3: FinDiff Quality

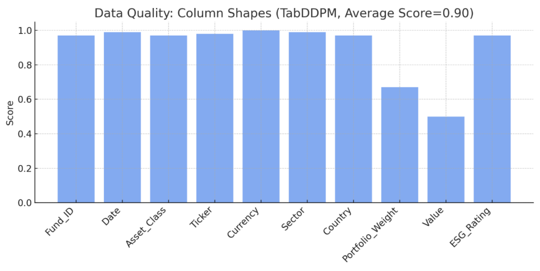


Figure 4.4: TabDDPM Quality

Feature	**FinDiff Score**	**TabDDPM Score**
Fund_ID	0.81	0.98
Asset_class	0.87	0.97
Date	0.9	0.95
Country	0.88	0.94
Sector	0.89	0.92
ESG_rating	0.9	0.96
Value	0.88	0.52
Portfolio_Weight	0.98	0.69
Currency	0.96	0.96
Average Score	**0.9**	**0.9**

Figure 4.5: Column score quality features

These two graphs show the generation quality of each columns of the dataset. As it can be noticed, both diffusion models work very well (0.9 average score). Nevertheless, the most remarkable difference that can be observed is that the feature "Value" is not accurately generated in the TabDDPM case.

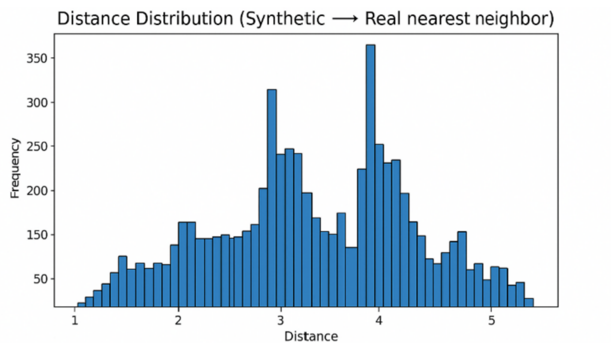
This feature is the only one which is numerical among all the columns, hence this may indicate that such behaviour could be attributed to differences in the architectures of the two generative models.

This is coherent with the theory. The theoretical explanation is the following:

- In TabDDPM, numerical data are handled through a discretization/normalization process, followed by reconstruction via a reverse diffusion procedure. This process involves adding Gaussian noise and then iteratively denoising the data. As a result, numerical precision may degrade due to the repeated noise injection and removal steps. The categorical columns in TabDDPM are often reconstructed accurately, as the multinomial framework is well-suited for handling discrete variables. However, the numerical features tend to appear more coarse-grained or exhibit flatter distributions compared to the real data. This occurs because TabDDPM's numerical component is not specifically optimized, unlike FinDiff, which is explicitly designed to handle continuous numerical structures with higher fidelity.

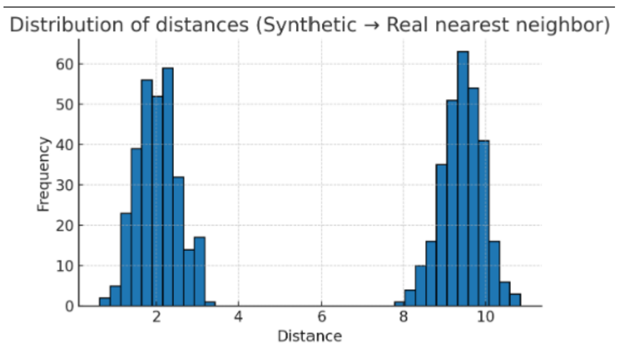
- In contrast, FinDiff is a model specifically designed for mixed numerical-categorical and financial data. It adopts a continuous stochastic differential equation (SDE) formulation, which is more natural for modeling continuous-valued variables. This approach eliminates the need for coarse discretization and better preserves the geometry of the underlying continuous distribution.

After analyzing the quality of the generated data, a study on privacy concerns was conducted.



- Number of synthetic records: 5000
- Average distance to nearest real neighbor: 3.2826
- Minimum distance found: 1.0416
- Suspicious records (distance $<$ 1e-6): 0

Figure 4.6: FinDiff Privacy



- Average distance to nearest real neighbor: 5.49
- Minimum distance found: 0.46
- Suspicious records (distance < 1e-6): 0

Figure 4.7: TabDDPM Privacy

FinDiff is more robust in terms of privacy, with a more uniform distribution and no suspicious samples that are too close to the real ones. TabDDPM performs better in terms of statistical fidelity but risks generating samples identical to real data, making it less secure for privacy. In other words, FinDiff sacrifices some fidelity to ensure privacy, while TabDDPM maximizes fidelity but risks compromising privacy.

		TabDDPM	**FinDiff**
Privacy	Average distance to nearest real neighbor	5.49	3.2826
	Minimum distance found	0.46	1.0416
	Suspicious records (distance $< 10^{-6}$)	0	0
Generation quality	Average score data generation quality	0.9	0.9
	Lower value among features	0.81	0.52
Runtime GPU	Total average time of a run (min)	28.6	24.9

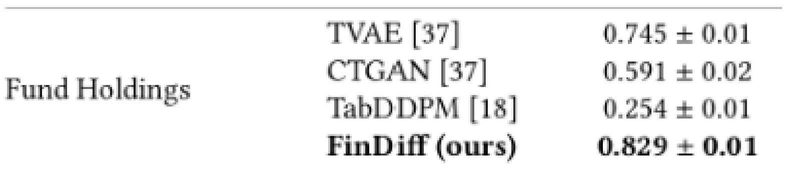
Figure 4.8: Global comparison between TabDDPM and FinDiff

Final considerations for the first dataset: Overall , it can be said that FinDiff is better because it is more robust for privacy concerns and it manages better numerical features.

4.3.2 Results for second dataset

The results for the second dataset were completely different. Indeed, as it was explained in the description of the dataset, it is fully composed by numerical features. For that reason, FinDiff should perform in a better way if compared to TabDDPM, but it does not give optimal performance as in the case of mixed categorical-numerical data.

This can be seen from the picture below taken from the paper [2], where Fund Holdings was the dataset with the largest number of numerical features among the three tested in that work.



Fund Holdings	TVAE [37]	0.745 ± 0.01
	CTGAN [37]	0.591 ± 0.02
	TabDDPM [18]	0.254 ± 0.01
	FinDiff (ours)	**0.829 ± 0.01**

Figure 4.9: FinDiff on the numerical dataset of the paper (Source: [2])

The intuition seems to be that the more the dataset becomes purely numerical, the worse the performance gets.

Since diffusion models have high computational costs for the running, for the case of numerical data the idea is that the use of this can be sobstituted by easier models like Copula-based generative models, already described in Chapter 2.
Using these Gaussian copula models, there can be gained very high performances on the numerical dataset, both from quality generation and privacy concerns.

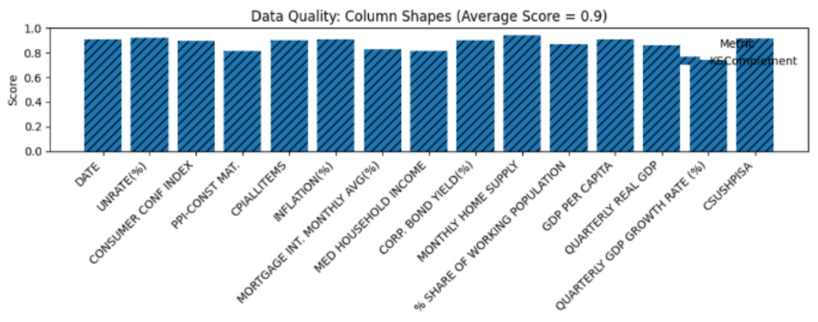


Figure 4.10: Copula model generation quality

In this case, other analysis have been made because of the numerical structure. For instance, one remarkable analysis in these kind of datasets can be the comparison among distributions, and it can be seen that the synthetic data have similar distributions to the real data.

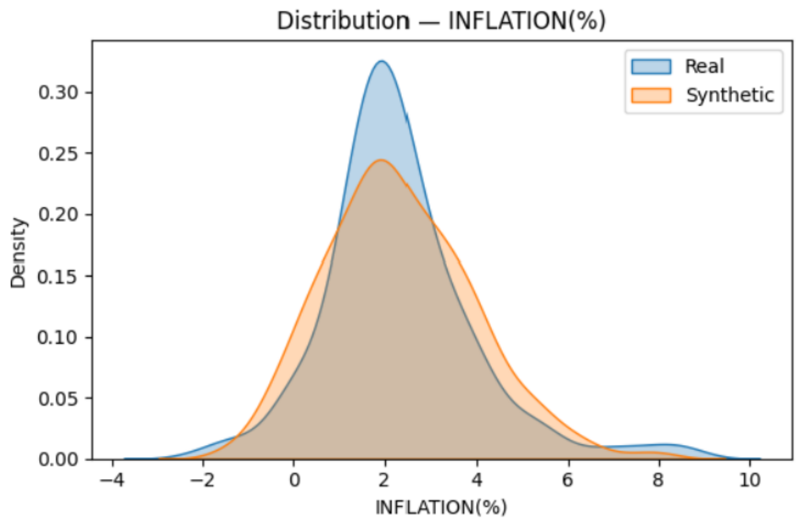
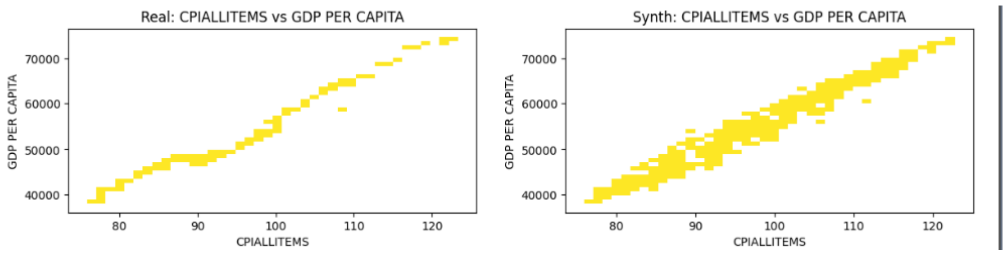


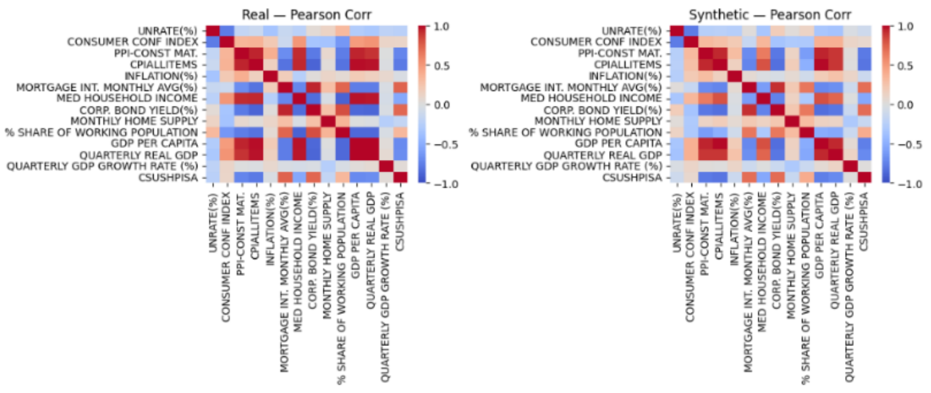
Figure 4.11: Inflation distribution

This is very useful, since for instance inflation samples are limited, and a synthetic dataset can be very useful for forecasting purpose. In the picture below, there is the comparison between the actual inflation distribution and the synthetic generated inflation. It can be noticed that the two distributions are very similar, and inflation is mainly focused in the range 0-4 %, with particular focus on 2%, as it is in the prediction of financial istitutions.

Another analysis which has been conducted is about the correlation between feature. The question that needs to be answered is: Did the synthetic data mantain correlation between variables. The results show that the model is robust enough from this point of view, as it can be seen from the pictures below.



(a) Correlation analysis: a single feature



(b) Correlation Analysis: general

Figure 4.12: Correlation analysis results. (a) Single feature, (b) General analysis.

These are the main results, considering statistical properties and privacy protection robustness. In the next chapter there will be the conclusions.

The following is the main framework of copula-based generative models used for the generation of synthetic dataset.

Algorithm 1 Copula-based Synthetic Data Generator Framework

```
Require: Observed dataset $X \in \mathbb{R}^{n \times d}$
1: Estimate each marginal distribution $F_i$ from $X_{\cdot,i}$
2: Transform data to uniforms: $u_{k,i} \leftarrow F_i(x_{k,i})$
3: Fit a copula $C$ to the transformed data $\{u_k\}_{k=1}^n$
4: for $m = 1$ to $M$ do
5:     Sample $\tilde{u} \sim C$
6:     Compute $\tilde{x}_i \leftarrow F_i^{-1}(\tilde{u}_i)$ for all $i$
7: end for
8: return Synthetic dataset $\{\tilde{x}_m\}_{m=1}^M$
```

Chapter 5

Conclusions

This work analysed the difference between some diffusion models, and made a case study on two different datasets. While for the numerical dataset, it was enough using simpler methods like copula based models, the great results of diffusion models can be observed mainly in the case of mixed data (categorical and numerical). In that case the results are remarkable, and the diffusion models represent one of the most interesting models in the field of tabular data synthetic generation.

5.1 Future works

This work has shown that the performance of generative models for tabular data depends on the nature of the dataset, particularly on whether features are numerical or mixed with categorical variables.

A possible direction for future research is to implement more adaptive architectures that can flexibly accommodate such heterogeneity. For instance, hybrid approaches that combine diffusion-based models with copula mechanisms or normalizing flows appear promising, as they could capture complex dependencies while preserving sampling stability.

Another remarkable topic for new research is how to apply diffusion models to imbalanced tabular data. There are interesting works in this direction, for instance [16] and [17].

One fundamental aspect to deal with is the **scalability** of diffusion models. Indeed, these models require very high computational costs and an interesting direction can be to analyse how reduce these costs in the architecture of the model, working on the training of the algorithms.

The work [14] provides an excellent reference for exploring potential future directions of diffusion models.

Python implementations

The python implementation of TabDDPM is the one in the paper [1], in particular in the official GitHub open-source repository https://github.com/yandex-research/tab-ddpm with MIT License, Copyright (c) 2022 Akim Kotelnikov.
The FinDiff implementation the one in the work [2], in particular in the official GitHub open-source repository https://github.com/sattarov/FinDiff with MIT License, Copyright (c) 2024 Timur Sattarov.
Additionally, I have written the code part for the privacy analysis.

Listing 5.1: Privacy check

```
#         Privacy check FinDiff (same style as TabDDPM)
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt

#         0) Load real/synthetic if not already in RAM



if 'df_real' not in globals():
    df_real = pd.read_csv("portfolio.csv")

# FinDiff: synthetic data start from samples_decoded
if 'df_syn_reordered' not in globals():
    df_syn = samples_decoded.copy()
    # use only common columns and reorder as in the real dataset
    common_cols = [c for c in df_real.columns if c in df_syn.
       columns]
    if not common_cols:
        raise ValueError("No common columns between real and
           synthetic data.")
    df_real = df_real[common_cols].copy()
    df_syn_reordered = df_syn[common_cols].copy()
else:
    # optional: ensure the same column order
    df_syn_reordered = df_syn_reordered[df_real.columns]

#         1) Encode categorical + scale numerical (SAME SCHEMA AS
   TABDDPM)
R = df_real.copy()
S = df_syn_reordered.copy()

R_enc, S_enc = R.copy(), S.copy()
for col in R.columns:
    if R[col].dtype == "object" or str(R[col].dtype).startswith("
       category"):
        le = LabelEncoder()
        both = pd.concat([R[col], S[col]], axis=0).astype(str)
        le.fit(both)
        R_enc[col] = le.transform(R[col].astype(str))
        S_enc[col] = le.transform(S[col].astype(str))

scaler = StandardScaler()
R_enc = pd.DataFrame(scaler.fit_transform(R_enc), columns=R.
   columns)
S_enc = pd.DataFrame(scaler.transform(S_enc), columns=R.columns)
```

```
#        2) Nearest Neighbor Distance (synthetic -> real)


nn = NearestNeighbors(n_neighbors=1, algorithm="auto").fit(R_enc)
distances, _ = nn.kneighbors(S_enc)
distances = distances.flatten()

#        3) Analysis (same print style as TabDDPM)


print(f"       Number of synthetic records: {len(distances)}")
print(f"       Average distance to nearest real: {distances.mean()
   :.4f}")
print(f"       Minimum distance found: {distances.min():.4f}")
print(f"       Suspicious records (distance < 1e-6): {(distances <
    1e-6).sum()}")

#        4) Plot distribution (blue, identical style)


plt.hist(distances, bins=50, color='steelblue', edgecolor='black')
plt.title("Distribution of distances (Synthetic     Real nearest
   neighbor)")
plt.xlabel("Distance")
plt.ylabel("Frequency")
plt.show()
```

Bibliography

[1] Kotelnikov, A., Baranchuk, D., Rubachev, I., Babenko, A., *Tabddpm: Modelling tabular data with diffusion models*, International Conference on Machine Learning, PMLR, 2023. URL: https://proceedings.mlr.press/v202/kotelnikov23a.html

[2] T. Sattarov, M. Schreyer, and D. Borth, *Findiff: Diffusion models for financial tabular data generation*, Proceedings of the Fourth ACM International Conference on AI in Finance, 2023. URL: https://doi.org/10.1145/3604237.3626876

[3] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*, MIT Press, 2nd edition, 2018.

[4] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, e S. Ganguli, *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*, Proceedings of the 32nd International Conference on Machine Learning. URL: https://proceedings.mlr.press/v37/sohl-dickstein15.html

[5] J. Ho, A. Jain, and P. Abbeel, *Denoising Diffusion Probabilistic Models*, Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 6840–6851, 2020.

[6] M. Ryan and L. Massaron, *Machine Learning for Tabular Data: XGBoost, Deep Learning, and AI*, Simon Schuster / Manning Publications, 2025.

[7] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[8] F. Rosenblatt, "The Perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[9] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*, Springer Texts in Statistics, vol. 103, Springer, New York, NY, 2013.

[10] R. Houssou, M.-C. Augustin, E. Rappos, V. Bonvin, and S. Robert-Nicoud, *Generation and Simulation of Synthetic Datasets with Copulas*,

[11] D. P. Kingma and M. Welling, *Auto-Encoding Variational Bayes*, arXiv preprint arXiv:1312.6114, 2014.

[12] Y. Tashiro, J. Song, Y. Song, and S. Ermon, *CSDI: Conditional Score-based Diffusion Models for Probabilistic Time Series Imputation*, Advances in Neural Information Processing Systems (NeurIPS), 2021.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[14] Y. Li, J. Zhang, L. Wang, and K. Chen, *Diffusion Models for Tabular Data: Challenges, Current Progress, and Future Directions*

[15] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, *Modeling Tabular Data Using Conditional GAN*

[16] Y. Qin, Y. Zhang, Y. Li, and J. Zhao, *Class-Balancing Diffusion Models*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 22107–22117, 2023.

[17] P. Panagiotou, A. Roy, and E. Ntoutsi, *Synthetic Tabular Data Generation for Class Imbalance and Fairness: A Comparative Study*

[18] T. Janke, M. Ghanmi, and F. Steinke, *Implicit Generative Copulas*, Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS), 2021.

[19] R. Wen and K. Torkkola, *Deep Generative Quantile-Copula Models for Probabilistic Forecasting*, 2019.

[20] N. Tagasovska, D. Ackerer, and T. Vatter, *Copulas as High-Dimensional Generative Models: Vine Copula Autoencoders*