TUM Data Innovation Lab

Munich Data Science Institute (MDSI)

Technical University of Munich

&

PwC

Final report of the project:

Synthetic Data Generation with Generative Artificial Intelligence

Authors	Valentin Gölz, Eva Resch, Ju-Shan Chao, Henri Petuker, Mazen Ba Shammakh
Mentors	Oliver Kobsik (PwC), Sophie Mutze (PwC), Jan-Patrick Schulz (PwC)
TUM Mentor	Prof. Dr. Massimo Fornasier (MDSI)
Project lead	Dr. Ricardo Acevedo Cabra (MDSI)
Supervisor	Prof. Dr. Massimo Fornasier (MDSI)

July 2024

Abstract

Institutions such as banks and insurance companies invest significant resources in constructing data to meet business requirements, often facing challenges related to data quality and availability. Synthetic data generation offers a promising alternative, enabling the creation of high-quality data sets while maintaining privacy and adhering to data regulations. This project aims to address data scarcity by developing innovative methods to generate synthetic data that closely mirrors real-world scenarios. More specifically, we generate accurate synthetic data sets using only statistical information such as mean, distribution type, or correlation factors, without relying on other input information.
The project is split into two approaches: the first approach employs methods from mathematical statistics such as Bayesian networks and copulas. The second approach leverages recently developed Large Language Models such as GPT-4o or Gemini-flash and different prompt-techniques for the generation. We create synthetic data sets of high quality with both approaches. Our comparative analysis highlights the strengths and differences between the statistical and Generative Artificial Intelligence approaches, providing a comprehensive understanding of their capabilities and outcomes. Our findings demonstrate that both, the statistical and the Generative Artificial Intelligence approach, can effectively generate synthetic data that preserves privacy and maintain statistical accuracy. These methodologies offer powerful tools for various applications, including privacy-preserving data sharing, advanced model training, and scenario testing.

Acknowledgements

We thank Oliver Kobsik, Sophie Mutze and Jan-Patrick Schulz (all PwC) for their great support during the project phase. Furthermore, we thank Dr. Ricardo Acevedo Cabra and Prof. Dr. Massimo Fornasier for setting up the TUM Data Innovation Lab and for letting us be part of this project. We also thank everyone else at MDSI and TUM responsible for making this project happen. We thank Prof. Dr. Massimo Fornasier for giving us helpful guidance during the project, especially during the milestone meetings. Last but not least, we thank PwC Germany for sponsoring this project.

Contents

1 Introduction

TUM Data Innovation Lab (TUM-DI-Lab) is a summer (or winter) educational research experience offered by the Munich Data Science Institute (MDSI) [21]. The Lab welcomes master students from TUM, other German universities and Erasmus students interested in exploring AI and new data-driven approaches to interdisciplinary challenges.
The project *Synthetic Data Generation with Generative Artificial Intelligence* was conducted in corporation with PwC Germany and took place between April and July 2024. The overarching goal of this research project was the generation of synthetic data that is solely based on a parametric description of the underlying distribution, using statistical as well as Generative Artificial Intelligence (GenAI) methods and comparing them.

We begin this report by defining the scope of the project in section 2 and presenting some related work in section 3. Then, we introduce our code base in section 4 and present the statistical approaches to our problem in section 5 and the approach via GenAI in section 6. Lastly, we evaluate the results from this project in section 7, suggest next steps in section 8 and come to a conclusion in section 9.

2 Project Goal

The objective of this project is to build a tool for PwC to help optimise the use of all accessible information. Due to data protection requirements and legal restrictions, PwC often has limited access to real-world data. For this reason, we are focusing and limiting ourselves in this project to statistical information such as univariate and multivariate measures. In particular, this means to have insights into univariate measures for each variable, like the median, the quartiles and an expert opinion of the underlying distribution, and some multivariate measures, most importantly the empirical correlation matrix. Given this information, we want to generate synthetic data that mimics the data described by the univariate and multivariate measures. This synthetic data does not pose any problems regarding privacy and can therefore be used internally for in-depth market analyses, portfolio modeling, benchmarking, and even help for publications.
The twist of this project is that we want to follow both a modern approach with GenAI and a classical approach leveraging statistical methods. We are able to compare these approaches, exploit their strengths and weaknesses, and present our findings.

3 Related Work

The landscape of synthetic data generation has recently seen substantial advancements, with significant contributions from various fields and with many applications. Here, we provide an overview of notable research in this domain.
Guo and Chen [9] explore the use of GenAI, particularly Large Language Models (LLMs), for generating synthetic data. They highlight the ability of these models to generate data that rivals real-world data in scenarios with limited availability. Patel et al. [15] introduce DataDreamer, an open-source tool for implementing LLM workflows, including synthetic data generation. They address challenges related to the scale, closed-source nature, and

reproducibility of LLM-based workflows. Yu et al. [24] investigate the use of LLMs as training data generators with diverse attributed prompts. Their study highlights the importance of attribute diversity in enhancing model performance and mitigating biases in synthetic data sets. In [4], Van Breugel and Van der Schaar advocate for prioritizing research on large tabular models (LTMs) to revolutionize the use of tabular data in machine learning. They argue that LTMs could transform data science by enabling few-shot learning, automating data analysis, and improving out-of-distribution data generation.
The exploration of LLMs' ability to generate random numbers and sample probability distributions has also been a critical area of research. Studies such as *Can LLMs Generate Random Numbers?* [11] evaluate LLMs' performance as distribution samplers, revealing challenges in inducing reasonable distributions over generated elements, which suggests a need for careful consideration of sampling methodologies. Additionally, the Stochastic Interpolants framework introduces a unifying approach for flow-based and diffusion-based generative models [3]. This framework leverages stochastic processes to bridge arbitrary probability density functions, offering a robust method for constructing generative models with adjustable noise levels and control over likelihood, highlighting its potential applicability in probabilistic tasks and generative modeling.
While there exist rich resources in the area of synthetic data generation, we haven't found related work that generates new data directly from statistical parameters, without additional training data or multi-shot techniques. This leads to a new research field that has yet to be developed.

4 Code Base

In this section, we give an introduction to our code base. The repository is currently private as it belongs to our project partner PwC. We focus on explaining the general structure and show which models can be used. We implement all models mentioned here in sections 5 and 6.

4.1 Architecture Overview

Our overarching goal is the creation of a fully functioning application, which can be used to generate custom synthetic data that fits the operator's needs. Thus, our application has two parts: a frontend, which implements a well-rounded UI / UX, and a backend, which is used for the generation of data. Because the focus of this project lies on the research of new approaches, we choose the open-source framework `streamlit` to implement the frontend. This framework suits all our needs, and is extremely easy to use.
The project requirements are to facilitate uploading statistical information such as the distribution type or the median, as well as a correlation matrix. The user should then be able to choose a generation method, e.g. copula or GPT-4o, and then generate synthetic data based on that approach. We now give an overview of the front- and backend to explain how the application works.

Frontend

The first thing a user sees when opening the application is the screen in Figure 4 (the figures are in appendix C). On the left side in the side-bar, the user has the possibility to choose how many samples they want to generate, which model and which csv separator should be used. In the center, a short guide introduces the functionality of the application. After scrolling down, the user can choose which type of statistical data they want to provide. Currently, there are two options: *statistical information*, which includes values such as median or distribution type, and *correlation matrix*, which should be a csv file containing the correlation matrix of the input features.
After uploading the files as seen in Figure 5, the user only needs to press *generate synthetic data* and wait until the generation is completed. The generation time depends on the used method and on the sample size. Generally, the statistical approaches are faster as they rely on simpler algorithms than LLMs. The generated data then looks like the one in Figure 7. The user can always choose a different method or sample size and rerun the generation. One limitation of LLMs is their performance with varying sample sizes. Specifically, the application encounters difficulties in producing optimal output when the sample size exceeds 50. This issue does not exist for the statistical approaches.

Backend

We now cover the structure of the backend. For the file structure, see Figure 6. The centerpiece is the file `main.py`, which contains the main function. This function represents the pipeline of our project and is called for each generation. It receives statistical features from the user, calls the correct model and its synthetic data generation function and in the end returns the generated data to the user. Because of this pipeline structure, we are also able to run experiments directly from the backend. This allows us to run large-scale experiments, e.g. to facilitate a grid-search using all available methods and different sample sizes.
Besides this function, the most important module is the directory called `synthdata`. Here all the models are stored, the prompt templates are saved and also where the type-checking happens. To add a new method in the future, users can use a predefined template which is also part of the repository. The full process in documented in our `README` file.

4.2 Model Overview

We continue by giving a quick overview of the different models and approaches that we have implemented, the idea behind each approach will be explained in detail in the following chapters. For a summary see Table 1.
As already mentioned, our models can be categorized in two general approaches:

- A statistical approach, using standard methods from mathematical statistics
- A GenAI approach, leveraging recently published LLMs.

For the statistical part, we use two different approaches. Our baseline, the parametric approach, uses the random number generators provided by the library `numpy` [10] which can generate univariate and multivariate samples from various distributions, but is not fit

Model Name	Type	#Params
Parametric Method	Statistical	–
Copula	Statistical	–
Gemini Flash	GenAI	unknown
Gemini Pro	GenAI	>100 bn
GPT-3.5-turbo	GenAI	>175 bn
GPT-4-turbo	GenAI	>1.7 tn (est.)
GPT-4o	GenAI	>1.7 tn (est.)

Table 1: Models, types, and number of parameters

to generate from a joint distribution of different distribution types. Furthermore, we have developed an approach using copulas, which can generate samples from mixed multivariate distributions. In section 5.2 we cover an approach using Bayesian networks, however, we were not able to create a working implementation, which is why that approach is not covered in the model overview.
For the GenAI approach, we use five different state-of-the-art LLMs. We use the GPT models from OpenAI, as well as the Gemini models from Google.

5 Statistical Approaches

In our preliminary research on employing statistical methods in synthetic data generation, we found many different preexisting implementations. There are packages like `pyvine` and `copula` using copulas and `BaysianNetwork` from `pgmpy.models` to build a corresponding Bayesian network, all of them with the goal to model the inherent dependence structure of the input data. The big challenge we encountered was that every implementation relies on seeing the real data which we do not want to use in this project. This means, we extracted the ideas from the implementations and tried to find approximations and workarounds to adapt them to our needs.

5.1 Copulas

The idea in copula theory is to be able to represent a joint distribution by its marginal distributions and some type of *linking* function to model the dependence structure. In the words of Roger Nelsen in [14] that means those functions "couple" multivariate distribution functions to their one-dimensional marginal distributions.
The word *copula* stems from Latin meaning link and was first introduced by Sklar in 1959 [20]. Since then, this theory has gained a lot of traction and led to several conferences on copulas and their applications.

What is a copula?

In this section we give a brief overview about copula theory, without going into too much detail. For a more rigorous introduction and the probability-theoretic background, please refer to the works of R. Nelsen [14], F. Durante and C. Sempi [7], and C. Czado [6].

We begin by introducing some notation: we denote a d-dimensional random variable (r.v.) by $\mathbf{X}$ and its realisation by $\mathbf{x} = (x_1, \dots, x_d)$ and say $\mathbf{X} \sim F$ if $\mathbf{X}$ follows a distribution with distribution function F. The distribution functions can be absolutely continuous or discrete and therefore have corresponding densities or probability mass functions (pmf), both of which we denote by f. Furthermore, we denote the marginal distribution functions by $F_1, \dots, F_d$ and the marginal densities or pmf by $f_1, \dots, f_d$.
Following [6], we define the d-dimensional **copula** C.

Definition 1. *A d-dimensional **copula** C is a multivariate distribution function on the d-dimensional hypercube $[0,1]^d$ with uniformly distributed marginals.*
If C is absolutely continuous, we can define the corresponding copula density by $c(u_1, \dots, u_d) = \frac{\partial^d}{\partial u_1 \dots \partial u_d} C(u_1, \dots, u_d)$ for all $\mathbf{u}$ in $[0,1]^d$.

The fundamental presentation theorem for multivariate distributions was given by Sklar in [20] and is the cornerstone of the copula theory.

Theorem 2 (Sklar's Theorem)**.** *Let $\mathbf{X}$ be a d-dimensional r.v. with joint distribution function F and marginals $F_1, \dots, F_d$, then F can be expressed as*

$$F(x_1, \dots, x_d) = C\left(F_1(x_1), \dots, F_d(x_d)\right) \tag{1}$$

with associated density or pmf

$$f(x_1, \dots, x_d) = c\left(F_1(x_1), \dots, F_d(x_d)\right) f_1(x_1) \dots f_d(x_d) \tag{2}$$

for some d-dimensional copula C with copula density c.

How does this help us?

When thinking about simulating from some d-dimensional distribution function F, a rather straight forward approach would be via the Rosenblatt transform as described in [17]. This takes into account the conditional distributions $F_{j|1,\dots,j-1}(\cdot \mid x_1, \dots, x_{j-1})$ and their inverse $F^{-1}_{j|1,\dots,j-1}(\cdot \mid x_1, \dots, x_{j-1})$ and produces a sample $x_1, \dots, x_d$, see Algorithm 1.
Finding the conditional distributions $F_{j|1,\dots,j-1}(\cdot \mid x_1, \dots, x_{j-1})$ poses a challenge, especially since we only have information on the marginal distributions and the correlation matrix. But this means, we already have the inner part of the right-hand side of equation (1). By deriving a fitting copula function C from the correlation matrix that models the dependency between the variables well, we can adapt the Algorithm 1 to sample from the copula (cf. Algorithm 2) and utilize the respective marginal quantile functions of each variable to revert the samples back to the original scale.

Algorithm 1 Rosenblatt transform

$w_j \overset{\text{iid}}{\sim} Unif(0,1)$ for $j = 1, \dots, d$
for $j = 1, \dots, d$ **do**
  $x_j \leftarrow F^{-1}_{j|1,\dots,j-1}(w_j \mid x_1, \dots, x_{j-1})$
end for
return $\mathbf{x} = (x_1, \dots, x_d)$

Algorithm 2 Adapted Rosenblatt

$w_j \overset{\text{iid}}{\sim} Unif(0,1)$ for $j = 1, \dots, d$
for $j = 1, \dots, d$ **do**
  $u_j \leftarrow C^{-1}_{j|1,\dots,j-1}(w_j \mid u_1, \dots, u_{j-1})$
end for
return $\mathbf{u} = (u_1, \dots, u_d)$

Unfortunately, we're still facing the challenge of finding the copula function C.
In the following sections, we discuss a method to simplify the d-dimensional copula function C to a construct of several bivariate copula functions by conditioning. We then elaborate on how we can sample from the copula we constructed before and talk about some practical choices we made along the way.
Remark. We will not be able to model the dependence structure perfectly, but as shown in section 7, our approach produces strong results.

Pair-copula constructions

The main step of this approach is constructing a multivariate distribution, in particular the sought-after copula distribution C, using bivariate building blocks. This depends on conditioning, so we first introduce some theoretic background and notation on conditional densities and distribution functions in terms of bivariate copulas. This section follows the notations and definitions introduced in [6].

Using the definition of a conditional density

$$f_{i|j}(x_i \mid x_j) = \frac{f_{ij}(x_i, x_j)}{f_j(x_j)}$$

and the representation of a density f via the associated copula density as shown in equation (2), we can express conditional densities $f_{i|j}$ and distribution functions $F_{i|j}$ of bivariate distributions in terms of their copula.

Lemma 3. *Let $(X_i, X_j) \sim F$ be a bivariate distribution with associated copula C, then*

$$f_{i|j}(x_i \mid x_j) = c_{ij}(F_i(x_i), F_j(x_j)) f_j(x_j),$$
$$F_{i|j}(x_i \mid x_j) = \frac{\partial}{\partial u_j} C_{ij}(F_i(x_i), u_j) \mid_{u_j = F_j(x_j)} .$$

The proof of this lemma can be found in [6].
Traditionally, the conditional copula distribution is defined as

$$C_{i|j}(u_i \mid u_j) = \frac{\partial}{\partial u_j} C_{ij}(u_i, u_j)$$

and inserting this in the Lemma 3, we see that the following relationship holds

$$F_{i|j}(x_i \mid x_j) = \frac{\partial}{\partial u_j} C_{ij}(F_i(x_i), u_j) \mid_{u_j = F_j(x_j)} = C_{i|j}(F_i(x_i) \mid F_j(x_j)).$$

For a clearer notation we introduce the so-called *h-function.*

Definition 4. *We define the h-function corresponding to a bivarite copula C_{ij} as*

$$h_{i|j}(u_i \mid u_j) := \frac{\partial}{\partial u_j} C_{ij}(u_i, u_j) \quad \text{for all } (u_i, u_j) \in [0,1].$$

This concludes the definition of the conditional copula distribution $C_{i|j}$ and the representation of conditional densities $f_{i|j}$ and distribution functions $F_{i|j}$ in terms of their associated copulas.

Now, we want to motivate the usage of bivariate copula functions as building blocks by illustrating that for three dimensions. The goal is to write the joint density f just in terms of the marginal distributions F_i with densities f_i and bivariate copula densities.

Definition 5. *For the three r.v. X_1, X_2, and X_3 with joint density f, we define a pair copula decomposition of f as*

$$\begin{aligned} f(x_1, x_2, x_3) = {} & c_{13;2}(F_{1|2}(x_1 \mid x_2), F_{3|2}(x_3 \mid x_2); x_2) \\ & \cdot c_{23}(F_2(x_2), F_3(x_3)) c_{12}(F_1(x_1), F_2(x_2)) f_1(x_1) f_2(x_2) f_3(x_3). \end{aligned}$$

The derivation of this can be found in the appendix A. $c_{13;2}(\cdot, \cdot; x_2)$ denotes the density of the copula associated with the bivariate conditional distribution $(X_1, X_3) \mid X_2 = x_2$. We can generalize this notation for indices i, j and a set $D \subseteq \{1, \ldots, d\} \setminus \{i, j\}$, where $C_{ij;D}(\cdot, \cdot; \mathbf{x}_D)$ is the copula associated with the bivariate conditional distribution $(X_i, X_j) \mid \mathbf{X}_D = \mathbf{x}_D$. This is in general not the same as the conditional copula distribution $C_{ij|D}(\cdot, \cdot; \mathbf{u}_D)$.

So far, we have shown that by representing conditional densities $f_{i|j}$ by their associated copula, as seen in Lemma 3, and conditioning in a clever way, we find a representation of a three-dimensional density f just in terms of bivariate copulas and the marginal distributions.
The last step in this process of introducing the pair-copula constructions is to generalize what we just did to any d-dimensional setting. For this, we start again with the decomposition of the density f by recursive conditioning which yields

$$f(x_1, \ldots, x_d) = \left[\prod_{t=2}^{d} f_{t|1,\ldots,t-1}(x_t \mid x_1, \ldots, x_{t-1}) \right] f_1(x_1).$$

With this we can construct the canonical vine (C-vine) density.

Theorem 6 (Canonical vine density)**.** *Decompose the joint density by*

$$f(x_1, \dots, x_d) = \left[\prod_{j=1}^{d-1} \prod_{i=1}^{d-j} c_{j(j+i);1,\dots,j-1}\right] \left[\prod_{k=1}^{d} f_k(x_k)\right].$$

This is the result of recursively applying Lemma 3 to express $f_{t|1,\dots,t-1}$ in terms of the conditional distribution of (X_{t-1}, X_t) given $X_1, \dots, X_{t-2}$. We can see that the structure built by this formula starts at some index j and takes into account all the copulas associated with the conditional distribution of (X_j, X_i), for an index i ranging from $j+1$ to d, conditioned on the previous variables $X_1, \dots, X_{j-1}$. To make this clearer we can arrange all the bivariate copulas in a scheme:

$$\begin{array}{ccccc} c_{12} & c_{13} & c_{14} & c_{15} & \dots \\ & c_{23;1} & c_{24;1} & c_{25;1} & \dots \\ & & c_{34;12} & c_{35;12} & \dots \\ & & & c_{45;123} & \dots \\ & & & & \ddots \end{array}$$

and each row corresponds to an index j while iterating over the indices i.

Sampling with the copula

In this section we discuss, how this helps our goal in sampling from the copula and in turn from the joint distribution F. We already have all the pieces, we just need to put them together.
If we had samples $\mathbf{x} = (x_1, \dots, x_d)$ of a r.v. $\mathbf{X} \sim F$ and transformed them using the distribution function F, we would get samples $\mathbf{u} := F(\mathbf{x}) = (F_1(x_1), \dots, F_d(x_d))$ in $[0,1]^d$ that follow the copula distribution C associated with F, meaning equation (1) would read as

$$F(x_1, \dots, x_d) = C\left(F_1(x_1), \dots, F_d(x_d)\right) = C\left(u_1, \dots, u_d\right).$$

So, if we do this in reverse, sample $\mathbf{u}$ from the copula C and set $\mathbf{x} = (F_1^{-1}(u_1), \dots, F_d^{-1}(u_d))$, then this sample follows the joint distribution F. [6] offers a nice algorithm, using the C-vine structure described in Theorem 6, to sample from the copula C:

Algorithm 3 Sampling from a C-vine copula

$w_i \overset{\text{iid}}{\sim} Unif(0,1)$ for $i = 1, \dots, d$
$v_{1,1} \leftarrow w_1$
for $i = 2, \dots, d$ **do**
  $v_{i,i} \leftarrow w_i$
  for $k = i-1, \dots, 1$ **do**
    $v_{k,i} \leftarrow h^{-1}_{i|k;1,\dots,(k-1)}(v_{k+1,i} \mid v_{k,k}, \eta_{k,i})$
  end for
end for
return $u_i \leftarrow v_i$ for $i = 1, \dots, d$

This algorithm uses the h-function we have already defined in 4 which describes the conditional distribution of a copula C. The parameter $\eta_{k,i}$ is called a *copula parameter* that is used in the definition of the underlying copula. Implicitly, we had a parameter like this throughout all sections, here we denote it explicitly to highlight that it only depends on the indices k and i. In the following section, we discuss how this parameter looks like in practice.
We mentioned before that, in general, $C_{ij;D}(\cdot,\cdot;\mathbf{x}_D)$ and $C_{ij|D}(\cdot,\cdot;\mathbf{u}_D)$ are not the same, but now we defined the C-vine copula using the first and in the algorithm we use the conditioned second quantity. This is no problem since in the case of C-vines, the following relationship between the two quantities holds

$$\begin{aligned} &C_{i|1,\dots,i-k}(u_i \mid \mathbf{u}_{1,\dots,i-k}) \\ &\quad = \frac{\partial C_{i,i-k;1,\dots,i-k-1}(C_{i|1,\dots,i-k-1}(u_i \mid \mathbf{u}_{1,\dots,i-k-1}), C_{i-k|1,\dots,i-k-1}(u_{i-k} \mid \mathbf{u}_{1,\dots,i-k-1}))}{\partial C_{i-k|1,\dots,i-k-1}(u_{i-k} \mid \mathbf{u}_{1,\dots,i-k-1})}. \end{aligned}$$

Approximations and practical choices

As motivated in the beginning of this section, we want the copula - better the C-vine construction - to represent the dependence structure of the joint distribution. In our case, this is given by the correlation matrix, where each entry describes the so called *Pearson product-moment correlation* ρ_{ij} between variables i and j, a measure of linear dependence. Its definition as well as an estimation technique can be found in appendix A.
For the actual choice of bivariate copulas, we considered the *Gaussian* and *Clayton* copula. The Gaussian copula function is symmetric, whereas the Clayton copula function exhibits a strong tail dependency, as you can see in Figure 3, meaning they could represent different types of dependence well. In appendix A, we introduce both copulas with their specifications.
We decided to focus only on the Gaussian copula, since we encountered two main challenges working with the Clayton copula. Firstly, in order to utilize the function optimally, we would need to apply a rotation to adjust the tail dependency. Secondly, the function needs Kendall's τ, another measure of correlation between two variables, as a parameter which wasn't provided in the scope of this project. These restrictions pose starting points to improve and further develop this method, we discuss this in section 8.
In the end, the implemented workflow of this approach follows these steps:

1. Take as input the statistical information containing the marginal distributions and correlation matrix, as well as the number m of samples we want to generate.

2. Generate m samples $\mathbf{u} \in [0,1]^d$ according to the Algorithm 3 using the inverse h-function of the Gaussian copula with parameter $\eta_{k,i} = \rho_{k,i}$, the correlation.

3. Transform each of the m samples using the quantile function of the marginal distribution: $\mathbf{x} = (F_1^{-1}(u_1), \dots, F_d^{-1}(u_d))$.

Theoretically, the inverse h-function is conditioned on variables $1, \dots, k-1$ which influences the choice of copula. Since we decided to solely use the Gaussian copula and since, with the information we are given, we cannot make any assumptions on the conditional distribution of variables X_i and X_k, we cannot expect to find the true copula to use in Algorithm 3.

Therefore, we approximate the true copula by using a Gaussian copula and our choice of copula parameter is the correlation $\rho_{k,i}$ between X_i and X_k. With this approximation, we are able to produce strong results which are discussed in section 7.

5.2 Bayesian Networks

After having discussed copulas in section 5.1, we now focus on our second statistical approach, which is using *Bayesian networks*.
Bayesian networks have become a key tool in AI and statistical modeling due to their efficiency in representing and reasoning with probabilistic information. Introduced by Judea Pearl in 1982, Bayesian networks provide a structured way to manage complex probabilistic distributions via *directed acyclic graphs* (DAGs) [16].

What are Bayesian networks?

Bayesian networks are graphical models that represent the decomposition of probability distributions by DAGs and are used to estimate the joint distribution of variables. The possibility to encode conditional dependence between the variables in these networks makes them crucial in applications such as probabilistic and causal inference.
The paper "A Review of Bayesian Networks and Structure Learning"[22] guides the literature review. It provides an in-depth understanding of Bayesian networks, including the underlying theory, inference problems, and learning algorithms.
Key concepts are the decomposition of probability distributions, d-separation for determining conditional independence in DAGs, and a variety of inference problems such as estimating parameters and learning network structures from data.
The decomposition of the joint probability distribution $p(X_1, \ldots, X_d)$ according to the DAG is given as follows:

$$p(X_1, \ldots, X_d) = \prod_{j=1}^{d} p(X_j \mid \mathrm{pa}(X_j))$$

where $\mathrm{pa}(X_j)$ represents the parents of X_j and is the smallest subset for which the decomposition holds.
The second concept, d-separation, is a graphical criterion for determining conditional independence in a DAG. Two nodes X and Y are d-separated by the set S if the set S blocks all paths between X and Y. The conditional independence is formally denoted as $X \perp Y \mid S$.
Another concept that we use is inference, which is the computation of the posterior distribution of variables, given some evidence. There are two main types of inference: predictive inference, which predicts the probability distribution of a future or unobserved variable based on current evidence, and diagnostic inference, which determines the probability of some cause or past event based on observed evidence.
Our project focuses on predictive inference in the Bayesian network. The predictive inference process begins with collecting evidence, which updates the probabilities of unobserved variables in the network. A variable elimination prediction technique is an accurate inference algorithm that calculates the marginal distribution of a query variable by summing

over non-evidence variables. The network can provide accurate predictions by applying variable elimination to adjust the probability distribution based on the evidence. This approach exploits the structural and conditional dependencies within the network.
We use the Hill Climbing Algorithm to help us in the development process. The hill-climbing algorithm [2] is a heuristic optimization algorithm used in the Bayesian network's structure learning. It starts with an initial solution, an initial network, and then iteratively makes small changes to the network. Each time, it chooses the one that brings the most improvement according to a predetermined scoring function. This process continues until no further progress can be made, resulting in finding a local optimum. The method is particularly effective when dealing with large and complex data sets, as it is computationally infeasible to fully explore all possible network configurations.

How does the implementation look?

This section describes how we wanted to generate synthetic data from Bayesian networks. In developing this, we followed an example outlined by Martins et al. [13], which emphasizes preserving the statistical properties of the original data set through the use of Bayesian networks and DAGs. Our model begins with a correlation matrix and detailed statistical information that form the foundation of our approach. We first use the correlation matrix to group the variables into smaller clusters that exhibit strong correlations between its variables but not to the other clusters. Then, we create smaller DAGs for each cluster using conditional probability distribution (CPD) based on the statistical information provided. However, we encountered challenges with loops within the DAGs, which Bayesian networks cannot deal with. We try to resolve this by replacing CPD with the Hill Climb Search algorithm to ensure the necessary acyclic structures. Furthermore, the generated synthetic data shows identical values across all samples. We found that this is due to the deterministic predictions of the Bayesian network which lead to a lack of variability in the data.
The literature review also reflects several key challenges that we have encountered and potential solutions. The literature suggests that a synthesized full Bayesian approach combined with posterior predictive distributions is essential to introduce the necessary variability and properly quantify uncertainty [13]. Our initial implementation may not adequately capture the statistical variability required to generate robust synthetic data.
In order to improve on this approach, we need to compare it with those proposed in the literature and analyze the differences and similarities in the data sets used in a full Bayesian approach. We should also investigate using posterior predictive distributions and penalizing priors to identify specific adjustments and improvements that need to be made to our approach.

5.3 Parametric Approach

Given the challenges we faced using Bayesian networks, we explore parametric methods based on the `numpy` library as an alternative. The parametric approach ensures that the data maintains the statistical properties required to construct accurate models.
First, we initialize a structure to store the generated data. Sequentially, we include specific operations like traversing each variable in the statistical data set and then extracting its basic statistical attributes such as median, quartiles, minimum and maximum values, and

data type. Additionally, we want to make sure that the generated data contains a realistic percentage of missing entries by taking into account the missing percentage of the data. The synthetic data is generated based on normal, binary, or exponential distributions. We use the mean and standard deviation from the statistical data for the normally distributed variables and in case the standard deviation is not given we use the quartiles to approximate it. The correlation matrix is then transformed into a covariance matrix to adjust it to the method. We input the covariance matrix, standard deviation, and mean into the `numpy.random.Generator.multivariate_normal` method to generate samples of multivariate normal distributions. For binary distributions, we use the median as the binomial distribution probability and for exponential distributions, we use the inverse of the median as the rate parameter. If no specific distribution is specified, a normal distribution with a mean of the median and a standard deviation of 1 is used by default. We constrain the generated data to the specified minimum and maximum values and convert them to the appropriate data type, either integer or float. Afterwards, missing values are inserted and shuffled into the data to achieve the given rate of missing values for the variables.
By maintaining the statistical properties of the original data set and introducing variability through random sampling, the approach effectively generates synthetic data. While the generation is robust and flexible, it is limited by the functionality provided by the `numpy` library.

6 Generative Artificial Intelligence Approaches

In recent years, transformer-based LLMs have fundamentally changed the fields of GenAI and Natural Language Processing (NLP). By massively scaling model size and data, these models continued to achieve new state-of-the-art performances on various NLP tasks [5]. Motivated by these developments, we have explored the capabilities of GenAI models on the project's task of synthetic numerical data generation solely from statistical descriptions. As mentioned before, this marks a novel application of GenAI models as they are usually applied in settings where real data is available and is the starting point for synthetic data generation [8, 9].
One major concern that has been raised from the start is the mismatch between the fundamental nature of transformer-based LLMs with the task of generating truly random numbers and sequences of numbers. Transformers like the GPT family have been trained on Next Token Prediction, meaning they predict the next token, a representation of a word or part of a word, based on the previous context. In our use case, however, we are interested in sampling data points from a distribution and such data should be independent and identically distributed (i.i.d.). Furthermore, LLMs are challenged by tasks that involve numbers and mathematical understanding [19]. This can lead to situations where using large models like GPT-4o the model tends to use external tools like a random number generator instead of producing the output fully by itself. Overall, we expect the current generation of GenAI models to not be a perfect match for our use case yet, but considering the rapid developments in this field, future innovations might address these points.
In the following we will discuss the different approaches and models we have tested as well as issues that we have encountered.

6.1 Methodology

Given the high resource requirements of using GenAI models, we have opted to only investigate prompting as a tool to apply the models to our task. In contrast to the previously common approach of fine-tuning a model to a specific task, prompting does not require us to modify the underlying model by updating its parameters nor to have a data set of task examples to fine-tune the model on.
Other ideas that we have considered during the research stage of the project included Retrieval Augmented Generation (RAG). While it can improve performance and reliability on knowledge-intensive tasks by retrieving input-relevant information and adding it to the model input, it does not benefit our use case. This is particularly true since we do not know what kind of information could assist with our task and therefore do not have a data set from which to retrieve information. We have also discarded a Neurosymbolic approach to the problem, where we intended to use a LLM to translate the statistical input data to executable Python code to generate synthetic data. This approach is limited by the possibilities of using Python and e.g. the `numpy` library for generation and ultimately performs the same steps as the Parametric Approach in section 5.3 with additional LLM overhead. Thus, we focused on prompting LLMs for the task.

Standard prompting approach

We can view prompts as the glue between the statistical input data and the models. Since LLMs have been mainly trained to understand and continue natural language sentences, interpreting raw statistical information in table format is a challenging task and might even fail. Therefore, we have defined handwritten prompt templates which enclose the statistical information and numbers in natural language sentences, differentiating between prompt templates that include multivariate information like a correlation matrix and templates that do not. In addition to the statistical information a prompt for conversational style LLMs needs an instruction about the task that it should perform. Examples of prompt templates for the univariate and multivariate case can be found in Table 2.

`Sample {n_samples} data points from a normal distribution with mean {mean} and standard deviation {std}.`
`Sample {n_samples} data points from the following multivariate {n_variables}-dimensional joint distribution. The joint distribution is given by the marginal distribution for each variable and their correlation matrix at the end. Variable {number_var} follows a normal distribution with mean {mean} and standard deviation {std}. [...] The correlation matrix is given by {corr_matrix}.`

Table 2: Examples of uni- and multivariate prompt templates.

One significant milestone in NLP was the discovery of Emergent Abilities in LLMs which were not present in previous models [23]. Most notably, few-shot prompting enabled models to be able to perform previously unseen tasks by having a few examples of the task inside the model prompt [5].
As a natural step, we have also added few-shot examples to our handwritten prompts, as few-shot prompting can improve the models ability to understand the task and therefore its performance. In addition, defining concrete examples can lead the model to produce the output in the desired format, in our case arrays containing data samples. The few-shot

examples have been handcrafted using the Python library `numpy`, similar to the parametric method in section 5.3. Examples of full prompts can be found in appendix B.
Lastly, with few-shot prompting becoming a popular approach for using LLMs, the sensitivity of the models to the prompts has become a major topic. Specific wording, the structure of the prompts as well as the choice of few-shot examples can have a significant effect on the models performance for a task [18]. This has lead to the development of the field of Prompt Engineering where the input space of possible prompts is systematically explored. Although we have not actively applied these techniques to the prompts used in this approach, we explored Prompt Engineering in the following chapter on LLM frameworks.

DSPy: A modern framework for LLM programs

As LLMs are being used more widely, new tools and frameworks have been developed that systemize and simplify typical workflows with LLMs. For this project, we have decided to utilize the framework called DSPy [12] developed and maintained by the NLP group at Stanford University.
On the one hand, this framework allows us to define a "LLM program" for the task by breaking it down into a workflow of subtasks, for which we each prompt the model separately, visualized in Figure 1. Compared with our handwritten prompts, we let the model transform the statistical input data to textual descriptions by itself and we added an option for asking the model to correct its generated synthetic data, which is turned off by default.

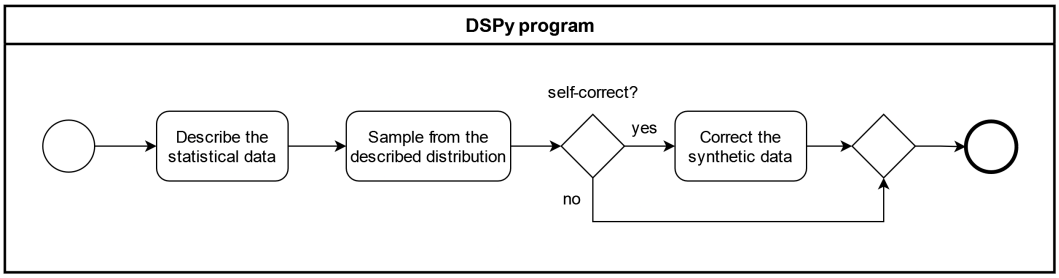

Figure 1: Workflow of data sampling DSPy program in BPMN.

On the other hand, the framework provides tools to automatically optimize few-shot example selection and prompt instruction formulation for each of the defined subtasks. With our handcrafted few-shot examples from the previous section used as train and development sets as well as an evaluation metric for the synthetic data following the evaluation method in section 7, we let the framework *compile* optimized DSPy programs for the models GPT-3.5-turbo and GPT-4o for uni- and multivariate style prompts.
For instruction wording optimization we investigated the COPRO teleprompter from the framework, which internally prompts the LLM to generate diverse variations of the signatures of DSPy modules, parts of prompts, and scores their performance on a development set. Examples for DSPy definitions and prompts can be found in appendix B.

Model choice

For this project, we have only investigated closed-source solutions including OpenAI's GPT family models and Google's Gemini family models over API access with their default parameters including model temperature. By providing this report and our code base, we hope to achieve good reproducibility of the experiments and results.
Initially, we have planned to also include open-source models like Llama 3 and DeepSeek-Math. We hypothesized that models exposed to more math and numerical data during pretraining would outperform models of similar size on our task, because of their superior performance on other math related tasks. However, during this project were unable to test these models and we cannot confirm this hypothesis.

6.2 Limitations of Language Models

During the development of the GenAI solutions, we have encountered several limitations specific to a Language Model approach. These limitations have been more significant in our use case, since we do not apply LLMs on typical language related NLP tasks but on a statistical data sampling task requiring numeric understanding and precision of input and output formats. In the following, we give a brief overview of the limitations and how we addressed them.

Output token limit

Typically when discussing token limits of LLMs, the focus lies on the available context length of the models - the amount of tokens you can feed into the model for one prompt. However, in our case the maximum number of output tokens posed a challenge as embedding numbers using tokenizers is rather costly, requiring multiple tokens for a decimal point number. This resulted in a limited amount of data points that can be generated with a single prompt.
We addressed this issue by reducing the amount of samples generated per prompt. Figure 2 shows the process for adjusting the sample size n based on an output token estimation procedure, currently only implemented for the OpenAI models. Given a sample size n, we estimate the total generated output tokens by utilizing the tokenizer used by the OpenAI models. If the amount of estimated output tokens exceeds the maximum output token limit, which is 4096 for these models, we reduce the sample size n such that it only generates within the maximum token limit. After generation we check if the number of generated samples matches the requested amount by the user. If the target sample size is not reached, the sample size n will be updated and the process loops back to the estimation step. This procedure will be continued until the original target sample size has been reached and since we are in an i.i.d. data setting it does not cause issues.

Unexpected artifacts in generation

LLMs can tend to generate unwanted artifacts or explanatory text in its output, which causes the output parsing to fail. Certain artifacts like omitting parts of the generated data with "..." or marking json code with quotations can be model specific, especially with models from the GPT family.

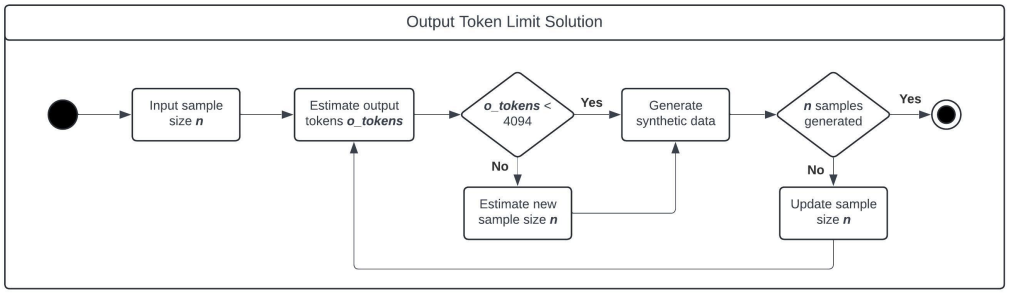

Figure 2: Process for prompting with different sample sizes

By using regular expressions to filter out unwanted text or artifacts, we tried to reduce the number of errors that were caused by this LLM property.

Wrong output format

To parse the generated synthetic data points and to evaluate them, we require a specific json format of the model completion. In spite of instructing the model to generate the desired array format as well as providing few-shot examples in the prompt, it sometimes fails to produce the format causing the output parsing to fail. As an example, we want all of the n-dimensional data points to be inside of a single array. However, at large n it can happen that the models output each point individually and not inside a single array.
As a workaround measure against these errors, we instruct the models to try up to $k = 5$ times if parsing errors occur.

Semantic errors

Lastly, due to the difficulty of the task for LLMs, we observed semantical errors of the models. This included outputting data points of the wrong dimensionality or failing the first DSPy subtask of describing the statistical input data causing the entire DSPy models to essentially generate random outputs. These issues highlight again the limits of current GenAI tools and their unreliability in performing this task.

7 Results

In this section we present and discuss the findings of our synthetic data generation approaches. First, we provide a comprehensive overview of the evaluation metrics using the supported univariate and multivariate parameters, followed by detailed results of the implemented solutions for both statistical and GenAI approaches. Finally, we compare the efficacy of these approaches to determine the most effective method for our synthetic data generation use case.

7.1 Evaluation

Methodology

The application currently supports a predefined list of univariate and multivariate statistical parameters which the user can provide. The univariate parameters include the following: *mean, median, q25, q75, min, max, standard deviation, distribution, datatype*, and *missing percentage*. The multivariate input is given in the form of a correlation matrix for the whole input data set. The distributions that are currently supported by the application include *Gaussian*, *binomial*, and *exponential*.

We differentiate between four types of evaluation metrics for both univariate and multivariate parameters:

1. The first evaluation type is calculating the deltas of the numerical univariate parameters, such as mean and standard deviation, for each feature, as well as calculating the deltas of the input correlation matrix and the correlation matrix of the generated synthetic data. This delta metric provides an insight about how far or close the synthetic data is from the data described in the input.

2. The second evaluation type is checking if non-numerical input and synthetic univariate parameters like distribution type match or not. Since deltas calculation only works for numerical parameters, the matching flag helps to determine the non-numerical parameter differences between the input statistics and the synthetic data statistics.

3. The third evaluation type is calculating the distance between the input correlation matrix and the one corresponding to the output data, using the Frobenius norm. The Frobenius norm provides basic insights about the magnitude of the delta of the entries of the input and synthetic correlation matrix. The lower the Frobenius norm is, the more similar the synthetic data is in terms of correlation to the original data.

4. The fourth evaluation type is calculating the relative scores for a subset of the statistical parameters, namely the mean, standard deviation, and Frobenius norm of the correlation matrices. Calculating the relative scores of the statistical parameters indicates how well each parameter is modeled in the synthetic data. Note that the relative score can be any real number, and is not limited to the range $[0, 1]$, but the overall performance measurement is based on how close the relative score is to 1. For instance, a relative mean of 1 indicates that the synthetic data and the original data has relatively the same mean. A relative mean that is less than 1 indicates that the synthetic data's mean is relatively lower than the origignal data's mean, and a relative mean that is greater than 1 indicates that the synthetic data's mean is relatively larger than the original data's mean. The following formula defines how the relative mean, standard deviation, and Frobenius norm are calculated. We first define the following parameters: n = number of features, s_i = synthetic measure$_i$, k_i = input measure$_i$. The formula is then given as:

$$\text{relative measure} = \frac{1}{n} \sum_{i=1}^{n} \frac{s_i}{k_i}$$

Data set

The test data set is provided by PwC, and it describes different data features for various risk drivers associated with borrowers, focusing on features such as the borrower's unique identifier (ID), the specific month and year of the recorded data (Time Series), and a trust score indicating the borrower's reliability (Business Relation Client). It also includes the Cash Flow Ratio, which measures how many times the borrower can cover their short-term obligations with their cash flow, and the geographical location of the borrower aggregated at the country level (CMS Country). This data aids in assessing and managing the risk associated with lending to different borrowers based on financial and geographic indicators. The data set includes 28 features in total. We focus on features that are distributed across one of the following distributions: normal, binary, or exponential. For our testing procedures, we generate data using every method with three variations of the given data set: all features variation, 10 features variation, and 5 features variation. For each variation, we generate 6 different synthetic data sets with 10, 20, 50, 100, 500, and 1000 samples. In the following sections, all evaluations are done using the full set of features, which is 28. Only the first and last evaluation types, namely the deltas and the relative scores, are presented in the following sections.

7.2 Statistical Approaches Results

The results of both the copula method and the parametric method for synthetic data generation demonstrated their robustness and reliability in generating statistically correct synthetic data. One advantage of these statistical approaches is that they had no constraints on generating large sample sizes. This indicates their suitability for a wide range of applications where large data sets are involved, ensuring accurate and robust synthetic data generation. Table 3 shows the relative scores of the different statistical methods for two sample sizes: 100 and 1000. The relative score for the copula and parametric approaches are close to 1, which means that both methods are performing well for synthetic data generation. For small and large sample sizes, the copula method seems to generate data that is statistically more similar to the original statistical information than the parametric method. Also, it is noticeable that the parametric method seems to always generate synthetic data that has relatively smaller univariate statistics than the copula method.

	$n_{samples} = 100$			$n_{samples} = 1000$		
	mean	std	Frobenius	mean	std	Frobenius
Copula	1.128	0.931	0.9857	1.18	0.9288	0.9377
Parametric	0.8425	0.8944	0.978	0.8177	0.8817	0.9846

Table 3: Relative scores of statistical approaches

Table 4 shows the results of generating 1000 samples for five correlated features using the copula method. For features 1-4, the deltas of the median, q25, q75, min, mean, and std are very negligible, which indicates that these univariate parameters of the input and

synthetic data are almost the same. On the contrary, sometimes there is a significant difference between the input and synthetic statistics, as shown for feature 5.

	median	q25	q75	min	max	mean	std
Feature 1	-0.73	0.29	0.25	3.72	-0.73	-0.07	-0.01
Feature 2	-0.01	0.01	0.01	0.20	9999999998	0.02	0.00
Feature 3	0.07	0.03	0.04	0	0	0.00	0.75
Feature 4	0.00	-0.00	-0.00	0.47	-0.05	-0.01	-0.01
Feature 5	-1029984	273163	322561	10959526	9982933055	-171212	7390

Table 4: Generation of 1000 samples using Copula

7.3 GenAI Results

Generally, the results of the GenAI approach heavily depend on the model choice since we are using one predefined prompt. Some of the models like GPT-4o generated statistically correct data for the three relative scores while others like GPT-3.5-turbo generated synthetic data that does not properly resemble the original data set. Table 5 shows the relative scores for the three models that we tried for this task with sample sizes 10 and 50. Note that the GPT-3.5-turbo model could not generate 50 samples for multiple correlated features. We can see that the latest GPT model, namely GPT-4o, outperforms the other models. Although there are token limit restrictions, GPT-4o and Gemini-1.5-flash models were able to generate 50 samples. We also noticed that although the token limit solution was implemented for the GPT models, GPT-3.5-turbo still can not generate correctly formatted and complete output when the sample size exceeds 50. The GPT-4o model outperforms the other two models since all its relative scores are very close to 1. It is also evident that GPT-3.5-turbo and Gemini-1.5-flash generate data that are far away from the original data's statistics. For instance, the relative std scores of the 10 and 50 samples generated by Gemini-1.5-flash show that this model cannot comprehend the notion of standard deviation, unlike GPT-4o.

	$n_{samples} = 10$			$n_{samples} = 50$		
	mean	std	Frobenius	mean	std	Frobenius
GPT-4o	0.8588	1.097	0.9528	1.103	0.9342	0.973
GPT-3.5-turbo	2.113	3.689	0.581	NA	NA	NA
Gemini-1.5-flash	0.913	0.0255	0.961	0.937	0.2569	0.9727

Table 5: Relative scores of GenAI approaches

There are a few aspects of the GenAI approach that should be taken into account for further analysis. First, the effectiveness of using few-shot prompting for synthetic data generation is not examined deeply because our main objective of using such prompting technique is to provide the model with the correct format of the output, and not to improve the performance of the model. It can be the case that the LLM sometimes reuses the same exact numbers that are present in the few-shot examples of the prompt. In addition, we

also need to take into account the possibility that the LLM models accessed over the API use a random number generator. In this case, this approach should be seen as a hybrid between LLM generation as well as programming with random number generators.
Lastly, we have compared the DSPy models with models using our handcrafted prompt templates and noticed the fragility and unreliability of the DSPy models. They often failed the first subtask of describing the statistical input data, resulting in generating essentially random data. In the case of using GPT-4o, the optimized DSPy model did not include any few-shot examples, as seen in the prompt in Table 10, meaning that adding even a single example hurt the performance, which is a rather unintuitive result.
Because of these findings, the DSPy approach is not included in further evaluations. Other examples for optimized DSPy prompts as well as examples for the mixed results achieved with prompt instruction optimization using COPRO can be found in appendix B.

7.4 Discussion

Given the results mentioned in the previous sections, it is evident that the statistical approaches achieve higher scores and are more deterministic than the GenAI approaches. Table 6 shows a summary of the relative scores for all the different generation methods implemented in this report. Note that the sample sizes are 10 and 50 for all the methods. We notice that for small sample sizes, the scores for the statistical approaches are slightly lower than the scores for large sample sizes. On the other hand, the GPT-4o model seems to be better than copula and parametric method in generating data sets with small sample sizes.

	$n_{samples} = 10$			$n_{samples} = 50$		
	mean	std	Frobenius	mean	std	Frobenius
Copula	0.837	0.7509	0.9712	0.9729	0.9244	0.9794
Parametric	0.9582	0.7217	0.9563	0.7907	0.9083	0.973
GPT-4o	0.8588	1.097	0.9528	1.103	0.9342	0.973
GPT-3.5-turbo	2.113	3.689	0.581	NA	NA	NA
Gemini-1.5-flash	0.913	0.0255	0.961	0.937	0.2569	0.9727

Table 6: Comparison of relative scores

Another important aspect that is important during the evaluation phase it to calculate the amount of time each generator takes to generate the synthetic data. Table 7 shows a comparison of the generation time for the different statistical and GenAI methods. Generally, statistical methods takes much less time for the generation of small sample sizes than the GenAI methods. The fastest generator is the parametric method, with nearly 1.31 seconds for generating 1000 samples. In fact, GenAI generation is considered expensive in terms of time. For instance, the time taken to generate only 10 samples using GenAI can generate more than 400 samples using statistical methods. This makes it evident that statistical methods are more practical than the GenAI methods given the importance of the time asset for businesses.

$n_{samples}$	10	20	50	100	500	1000
Copula	1.25	5.80	9.60	16.73	70.40	140.80
Parametric	0.89	2.19	1.30	0.96	0.98	1.31
GPT-4o	68.24	83.33	190.80	351.76	NA	NA
GPT-3.5-turbo	42.32	NA	NA	NA	NA	NA
Gemini-1.5-flash	39.19	33.62	42.30	NA	NA	NA

Table 7: Comparison of generation time in seconds

8 Future Work

In this section, we discuss possible adjustments and next steps that could improve our proposed methods.
As hinted upon in section 5.1, there are a few possible improvements to be made to the copula approach. The first would be to implement further copula functions and see if they perform better. We suggest looking into the Clayton copula since it can model non-symmetric dependencies between variables. Using rotations of the variables could help fitting them to the dependence structure in the data. Further considerations can be found in A. Furthermore, since using just one type of copula might not be sufficient, we also suggest optimizing over the assignment of each copula type. Meaning, one could generate a relatively small sample of data (e.g. $n = 100$) for each possible assignment, evaluate the generated data as described in section 7 and choose the assignment producing the best results.

One possibility to extend on our proposed methods could be to combine the statistical and GenAI approaches into one integrated solution, utilizing the strengths of both through a boosting mechanism. However, we lack a concrete suggestion for what such a solution could look like.

Regarding GenAI, more prompting techniques like Chain-of-Thought could be applied to our handwritten prompt templates. One example could be to formulate the correlation matrix into a Chain-of-Thought-style interpretation of its content, making it more digestible by models than appending a complete correlation matrix to the prompt. Simplifying the correlation matrix by setting small correlation values to 0 and accepting some inaccuracy could further improve the performance of GenAI and also statistical models.

Lastly, the evaluation metrics can also be extended to more qualitative characteristics. For instance, criteria like naturalism, consistency, realism, and usefulness of the synthetic data could provide more insightful analyses about how valuable the generated synthetic data is and how close it resembles the real data. In addition, a better single evaluation score can be defined for reflecting the overall performance of a generator.

9 Conclusion

This report addresses the task of synthetic data generation solely from statistical information of original data sets and explores various methods to produce statistically correct synthetic data. We investigated both statistical and GenAI approaches, including the copula method, parametric method, and prompting techniques using modern LLMs. Based on our evaluation, we concluded that the best overall generator for this task is the copula method as it generated synthetic data whose properties are closest to the input statistical information. Although the parametric method generated data faster, it had worse scores than the copula method. Among the GenAI models, the GPT-4o model from OpenAI performed best and placed third overall, followed by Gemini-1.5 Flash. GenAI models struggled to generate data in the correct format for large sample sizes and more difficult settings with many input variables.

With our developed tools and experiments, we believe that we can support data-driven research and applications, aiming to solve the issue of data scarcity in practice. In cases where enterprises might not be willing to expose their original data for privacy and security reasons, but might be willing to share their data's statistical information, synthetic data could be generated and used. Our findings contribute to the field by offering first practical insights and give an outlook on future developments and possible extensions.

References

[1] Kjersti Aas et al. "Pair-copula constructions of multiple dependence". In: *Insurance: Mathematics and economics* 44.2 (2009), pp. 182–198.

[2] Ria Puan Adhitama and Dewi Retno Sari Saputro. "Hill climbing algorithm for Bayesian network structure". In: *AIP Conference Proceedings*. Vol. 2479. 1. AIP Publishing. 2022.

[3] Michael S. Albergo, Nicholas M. Boffi, and Eric Vanden-Eijnden. *Stochastic Interpolants: A Unifying Framework for Flows and Diffusions*. Nov. 6, 2023. arXiv: 2303.08797[cond-mat]. URL: http://arxiv.org/abs/2303.08797 (visited on 05/13/2024).

[4] Boris van Breugel and Mihaela van der Schaar. "Why Tabular Foundation Models Should Be a Research Priority". In: (2024). DOI: 10.48550/ARXIV.2405.01147. URL: https://arxiv.org/abs/2405.01147 (visited on 05/05/2024).

[5] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[6] Claudia Czado. *Analyzing dependent data with vine copulas*. Vol. 222. Springer, 2019.

[7] Fabrizio Durante, Carlo Sempi, et al. *Principles of copula theory*. Vol. 474. CRC press Boca Raton, FL, 2016.

[8] Peter Eigenschink et al. "Deep generative models for synthetic data: A survey". In: *IEEE Access* 11 (2023), pp. 47304–47320.

[9] Xu Guo and Yiqiang Chen. *Generative AI for Synthetic Data Generation: Methods, Challenges and the Future*. Mar. 6, 2024. arXiv: 2403.04190[cs]. URL: http://arxiv.org/abs/2403.04190 (visited on 04/24/2024).

[10] 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. URL: https://doi.org/10.1038/s41586-020-2649-2.

[11] Aspen K. Hopkins, Alex Renda, and Michael Carbin. "Can LLMs Generate Random Numbers? Evaluating LLM Sampling in Controlled Domains". In: ICML 2023 Workshop: Sampling and Optimization in Discrete Space. Aug. 2, 2023. URL: https://openreview.net/forum?id=Vhh1K9LjVI (visited on 05/05/2024).

[12] Omar Khattab et al. "Dspy: Compiling declarative language model calls into self-improving pipelines". In: *arXiv preprint arXiv:2310.03714* (2023).

[13] Larissa NA Martins, Flávio B Gonçalves, and Thais P Galletti. "Generation and analysis of synthetic data via Bayesian networks: a robust approach for uncertainty quantification via Bayesian paradigm". In: *arXiv preprint arXiv:2402.17915* (2024).

[14] Roger B Nelsen. *An introduction to copulas*. Springer, 2006.

[15] Ajay Patel, Colin Raffel, and Chris Callison-Burch. *DataDreamer: A Tool for Synthetic Data Generation and Reproducible LLM Workflows*. Feb. 15, 2024. arXiv: 2402.10379[cs]. URL: http://arxiv.org/abs/2402.10379 (visited on 05/03/2024).

[16] J. Pearl. *Bayesian Networks: A Model of Self-activated Memory for Evidential Reasoning*. Report (University of California, Los Angeles. Computer Science Dept.) UCLA Computer Science Department, 1985. URL: https://books.google.de/books?id=1sfMOgAACAAJ.

[17] Ludger Rüschendorf. "Stochastically ordered distributions and monotonicity of the OC-function of sequential probability ratio tests". In: *Statistics: A Journal of Theoretical and Applied Statistics* 12.3 (1981), pp. 327–338.

[18] Melanie Sclar et al. "Quantifying Language Models' Sensitivity to Spurious Features in Prompt Design or: How I learned to start worrying about prompt formatting". In: *arXiv preprint arXiv:2310.11324* (2023).

[19] Zhihong Shao et al. "Deepseekmath: Pushing the limits of mathematical reasoning in open language models". In: *arXiv preprint arXiv:2402.03300* (2024).

[20] M Sklar. "Fonctions de répartition à n dimensions et leurs marges". In: 8.3 (1959), pp. 229–231.

[21] *Startseite - Munich Data Science Institute*. URL: https://www.mdsi.tum.de/di-lab/tum-di-lab/.

[22] John Noble Timo J.T. Koski. "A Review of Bayesian Networks and Structure Learning". eng. In: *Mathematica Applicanda* 40.1 (2012). URL: http://eudml.org/doc/292706.

[23] Jason Wei et al. "Emergent abilities of large language models". In: *arXiv preprint arXiv:2206.07682* (2022).

[24] Yue Yu et al. *Large Language Model as Attributed Training Data Generator: A Tale of Diversity and Bias*. Oct. 17, 2023. arXiv: 2306.15895[cs]. URL: http://arxiv.org/abs/2306.15895 (visited on 05/03/2024).

Appendix

A: Statistical Approaches

Derivation of pair-copula decomposition in 3 dimensions

For the three random variables X_1, X_2, and X_3, we can factorize their joint density into

$$f(x_1, x_2, x_3) = f_{3|12}(x_3 \mid x_1, x_2) f_{2|1}(x_2 \mid x_1) f_1(x_1). \tag{3}$$

Firstly, to determine the conditional density $f_{3|12}(x_3 \mid x_1, x_2)$, we will apply Sklars Theorem 2 to represent the conditional density $f_{13|2}(x_1, x_3 \mid x_2)$ in terms of its associated conditional copula density, denoted by $c_{13;2}(\cdot, \cdot; x_2)$:

$$f_{13|2}(x_1, x_3 \mid x_2) = c_{13;2}(F_{1|2}(x_1 \mid x_2), F_{3|2}(x_3 \mid x_2); x_2) f_{1|2}(x_1 \mid x_2) f_{3|2}(x_3 \mid x_2). \tag{4}$$

Now, by standard rules of conditioning and plugging in equation (4), we find the following representation

$$f_{3|12}(x_3 \mid x_1, x_2) \overset{def}{=} \frac{f_{13|2}(x_1, x_3 \mid x_2)}{f_{1|2}(x_1 \mid x_2)} \overset{(4)}{=} c_{13;2}(F_{1|2}(x_1 \mid x_2), F_{3|2}(x_3 \mid x_2); x_2) f_{3|2}(x_3 \mid x_2). \tag{5}$$

Lastly, we represent the conditional densities $f_{2|1}(x_2 \mid x_1)$ and $f_{3|2}(x_3 \mid x_2)$ by their associated copula functions as shown in Lemma 3

$$f_{2|1}(x_2 \mid x_1) = c_{12}(F_1(x_1), F_2(x_2)) f_2(x_2) \tag{6}$$

$$f_{3|2}(x_3 \mid x_2) = c_{23}(F_2(x_2), F_3(x_3)) f_3(x_3) \tag{7}$$

Combining equations (4), (5), (6), and (7) and inserting this into (3) yields a so-called pair copula decomposition of the joint density.

Definition 7. *We define a pair copula decomposition of a 3-dimensional density f as*

$$\begin{aligned} f(x_1, x_2, x_3) = {} & c_{13;2}(F_{1|2}(x_1 \mid x_2), F_{3|2}(x_3 \mid x_2); x_2) \\ & \cdot c_{23}(F_2(x_2), F_3(x_3)) c_{12}(F_1(x_1), F_2(x_2)) f_1(x_1) f_2(x_2) f_3(x_3). \end{aligned}$$

Correlation measures

The *Pearson product-moment correlation* between two r.v. X_i and X_j is defined as

$$\rho_{i,j} := Cor(X - i, X_j) = \frac{Cov(X_i, X_j)}{\sqrt{Var(X_1)}\sqrt{Var(X_j)}}$$

and can be estimated by the following formula

$$\hat{\rho}_{i,j} = \frac{\sum_{k=1}^{n}(x_{ki} - \overline{x}_i)(x_{kj} - \overline{x}_j)}{\sqrt{\sum_{k=1}^{n}(x_{ki} - \overline{x}_i)^2}\sqrt{\sum_{k=1}^{n}(x_{kj} - \overline{x}_j)^2}}$$

with $\overline{x}_i = \frac{1}{n}\sum_{k=1}^{n} x_{ki}$.

Bivariate copulas

We consider the *Gaussian* and *Clayton* copulas since they can model different types of dependence structures. The Gaussian copula function is symmetric and the Clayton copula function exhibits a strong tail dependency, that means that for very large values in both variables, or very small ones, the density is much larger. Their densities are depicted in Figure 3. To introduce both copulas, we follow [1].

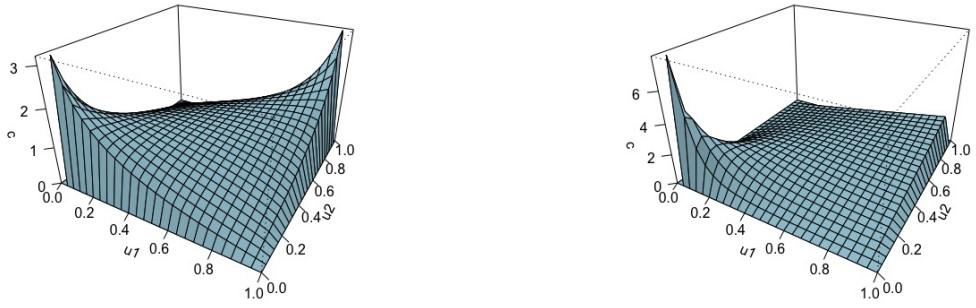

Figure 3: The graphic on the left shows the Gaussian copula density with parameter $\rho = 0.5$ and the graphic on the left the Clayton copula density with parameter $\delta = 1.2$.

The **bivariate Gaussian copula** has distribution function

$$C(u_i, u_j; \rho_{ij}) = \Phi_2(\Phi^{-1}(u_i), \Phi^{-1}(u_j); \rho_{ij}) \tag{8}$$

where ρ_{ij} is the copula parameter, Φ_2 the 2-dimensional standard normal distribution function and Φ^{-1} the quantile function of the 1-dimensional standard normal distribution $\mathcal{N}(0, 1)$. The corresponding h-function is given by

$$h_{i|j}(u_i \mid u_j; \rho_{ij}) = \Phi\left(\frac{\Phi^{-1}(u_i) - \rho_{ij}\Phi^{-1}(u_j)}{\sqrt{1-\rho_{ij}^2}}\right)$$

and its inverse by

$$h_{i|j}^{-1}(u_i \mid u_j; \rho_{ij}) = \Phi\left(\Phi^{-1}(u_i)\sqrt{1-\rho_{ij}^2} - \rho_{ij}\Phi^{-1}(u_j)\right). \tag{9}$$

The copula parameter ρ_{ij} is given by the correlation between the two variables X_i and X_j, so in case of our project, we can pull this information directly from the input data.
The **bivariate Clayton copula** has distribution function

$$C(u_i, u_j; \delta_{i,j}) = \left(u_i^{-\delta_{i,j}} + u_j^{-\delta_{i,j}} - 1\right)^{-1/\delta_{i,j}} \tag{10}$$

with copula parameter $\delta_{i,j}$. For $\delta_{i,j} \to \infty$, perfect dependence is obtained and $\delta_{i,j} \to 0$ corresponds to independence. The corresponding h-function is given by

$$h_{i|j}(u_i \mid u_j; \delta_{ij}) = u_j^{-\delta_{ij}-1}\left(u_i^{-\delta_{ij}} + u_j^{-\delta_{ij}} - 1\right)^{-1-1/\delta_{ij}}$$

and its inverse by

$$h_{i|j}^{-1}(u_i \mid u_j; \delta_{ij}) = \left(\left(u_i \cdot u_j^{\delta_{ij}+1} \right)^{-\frac{\delta_{ij}}{\delta_{ij}+1}} + 1 - u_j^{-\delta_{ij}} \right)^{-1/\delta_{ij}}. \tag{11}$$

The copula parameter δ_{ij} is given as a function of Kendall's τ between the variables X_i and X_j:

$$\delta = \frac{2\tau}{1-\tau} \in [0, \infty) \quad \Longleftrightarrow \quad \tau = \frac{\delta}{\delta + 2} \in [0, 1].$$

Kendall's τ is a measure of dependence between two vectors, describing the concordance and discordance in the signs of each entry. Here, this measure is not provided, making the use of the Clayton copula unappealing. But there are possibilities to estimate Kendall's τ. For example, in the case of two normal distributions [6] describes an exact relationship between τ and the correlation:

$$\tau = \frac{2}{\pi} \arcsin(\rho),$$

which could serve as an approximation.
Furthermore, the allowed domain for τ in the case of the Clayton copula is bounded to $[0, 1]$, but in practice it takes values in $[-1, 1]$. To resolve this dilemma, we suggest introducing a rotation of the variables in the bivariate copula density. For further information on this, consult [6].

B: GenAI Prompts

In the following, we give examples of GenAI prompts that are used in the application. Please find the complete list and further details in the project code.

Sample 10 data points from the following multivariate 5-dimensional joint distribution. The joint distribution is given by the marginal distribution for each variable and their correlation matrix at the end. Variable 1 follows a normal distribution with mean 2.67 and standard deviation 1.48. Variable 2 follows a normal distribution with mean 0.26 and standard deviation 0.13. Variable 3 follows a binary distribution with parameter 0.07, so the mean is 0.07 and the variance 0.07. Variable 4 follows a normal distribution with mean 0.28 and standard deviation 0.24. Variable 5 follows a normal distribution with mean 3892822.94 and standard deviation 5126277.04. The correlation matrix is given by [[1.0, 0.21, 0.07, -0.07, -0.07], [0.21, 1.0, 0.1, 0.23, -0.07], [0.07, 0.1, 1.0, -0.06, 0.1], [-0.07, 0.23, -0.06, 1.0, -0.04], [-0.07, -0.07, 0.1, -0.04, 1.0]]. You must follow the format of the following examples. Examples: Sample 20 data points from the following multivariate 2-dimensional joint distribution. The joint distribution is given by the marginal distribution for each variable and their correlation matrix at the end. Variable 1 follows a normal distribution with mean -3.42 and standard deviation 2. Variable 2 follows a normal distribution with mean 38.01 and standard deviation 1.73. The correlation matrix is given by [[1.0, 0.58], [0.58, 1.0]]. [[2.2846, 41.7095], [-1.7491, 36.5811], *...(omitted)...* [-1.3983, 40.7073]] Sample 10 data points from the following multivariate 4-dimensional joint distribution. The joint distribution is given by the marginal distribution for each variable and their correlation matrix at the end. Variable 1 follows a normal distribution with mean 2 and standard deviation 1.41. Variable 2 follows a normal distribution with mean 11.5 and standard deviation 1.22. Variable 3 follows a normal distribution with mean -1.1 and standard deviation 1.34. Variable 4 follows a normal distribution with mean 5.8 and standard deviation 1. The correlation matrix is given by [[1.0, 0.46, 0.32, 0.35], [0.46, 1.0, 0.24, 0.24], [0.32, 0.24, 1.0, 0.15], [0.35, 0.24, 0.15, 1.0]]. [[-0.4329, 10.7886, -2.5406, 5.5736], *...(omitted)...* [0.3122, 9.7824, -1.0206, 6.1308]]

Table 8: Example prompt for multivariate case style 1 (2-shot). Omitted data points in few-shot examples.

Write a short and easy to understand textual description of the single distribution described by the statistical information. Each new line of the statistical information is a new dimension of the distribution.
—
Follow the following format.
Statistical Information: statistical information about a single distribution. each new line is a new dimension of the distribution
Reasoning: Let's think step by step in order to ${produce the description}. We ...
Description: short and easy to understand description of the distribution
—
Statistical Information:
descr type mean std corr
NaN binary 0.94 NaN None
Reasoning: Let's think step by step in order to produce the description. We have a single dimension in the distribution, which is binary. The mean value of this binary dimension is 0.94, and there is no standard deviation or correlation information provided.
Description: This distribution represents a binary variable where the value is 1 most of the time, with an average occurrence of 94%.
—
Statistical Information:
descr type mean std corr
NaN normal -3.2 2.0 None
Reasoning: Let's think step by step in order to produce the description. We have a single dimension described by a normal distribution. The mean of this distribution is -3.2, and the standard deviation is 2.0. There is no correlation information provided, which is expected since there is only one dimension.
Description: This distribution is a normal distribution with a mean of -3.2 and a standard deviation of 2.0.
—
Statistical Information:
descr type mean std corr
NaN exponential 55.0 NaN None
Reasoning: Let's think step by step in order to produce the description. We

Table 9: DSPy prompt for subtask "describe statistical data" (univariate, 2-shot, GPT-3.5-turbo).

Sample data points from the described distribution.
—
Follow the following format.
Description: the description of the distribution
N Samples: the exact number of data points to sample
Reasoning: Let's think step by step in order to ${produce the sampled_data}. We ...
Sampled Data: no text or explanations. must only contain an array filled with the sampled data points. starts with "[" and must end with "]" like [3.1, -1.9] for two 1-dimensional points and [[0.27, 3.11], [-1.21, 8.31]] for two 2-dimensional points.
—
Description: This distribution consists of four dimensions. The first dimension is a normal distribution centered around -80.23 with some variability. The second dimension is another normal distribution centered around 49.24. The third dimension is a binary distribution with an average value of 0.40. The fourth dimension is an exponential distribution with an average value of 3.30. These dimensions are somewhat correlated with each other, meaning changes in one dimension might be related to changes in another.
N Samples: 10
Reasoning: Let's think step by step in order to produce the sampled_data. We

Table 10: DSPy prompt for subtask "sample data points" (multivariate, zero-shot, GPT-4o).

Positive example:
Original: Sample data points from the described distribution.
Optimized: Use probabilistic sampling to select data points based on the distribution described.

Mixed example:
Original: Write a short and easy to understand textual description of the given statistical information.
Optimized: Instruction #11: Infuse creativity and storytelling elements into the textual description of the statistical information, enhancing engagement and memorability.

Faulty example:
Original: Evaluate whether the given data points follow the described distribution. Make corrections if necessary. Also check if the number of data points match the defined number and if not add or remove points.
Optimized: The improved instructions for the language model would be to ———

Table 11: Examples of instruction optimization with DSPy COPRO on parts of the prompts (GPT-3.5-turbo).

C: User Interface of the Web Application

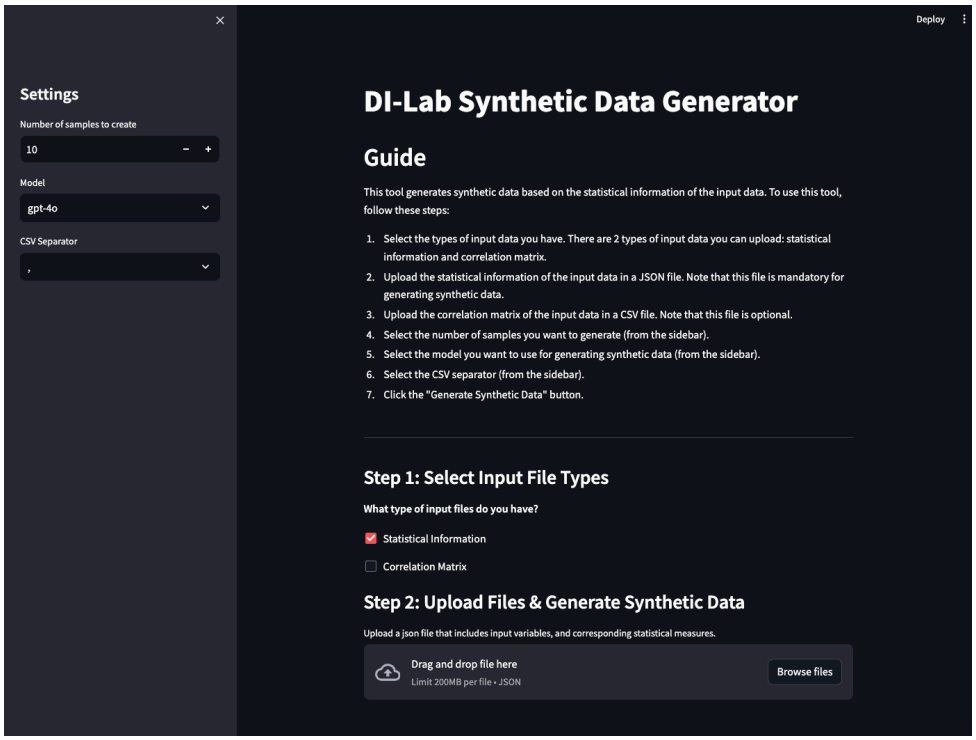

Figure 4: The initial welcome page.

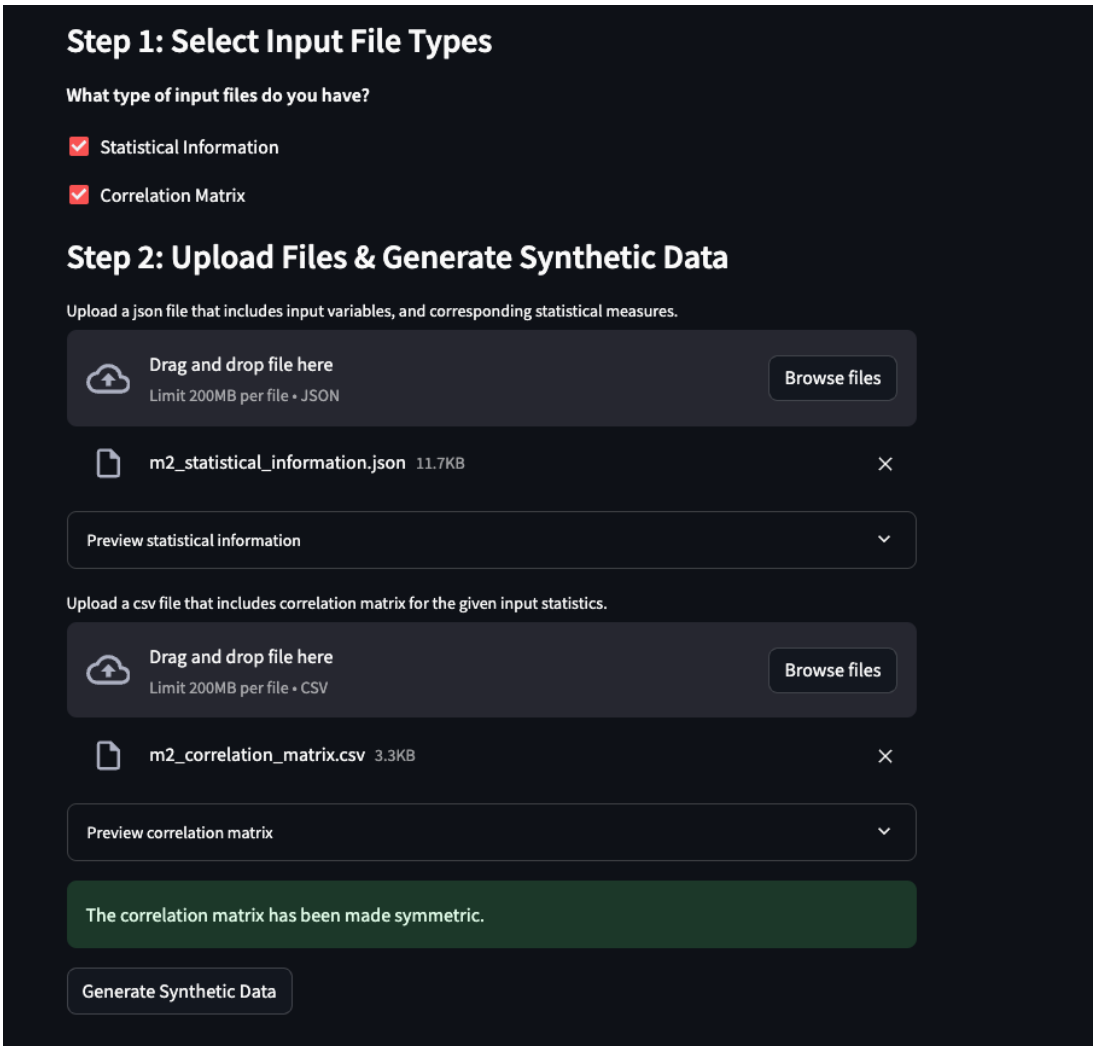

Figure 5: The file-upload of our application.

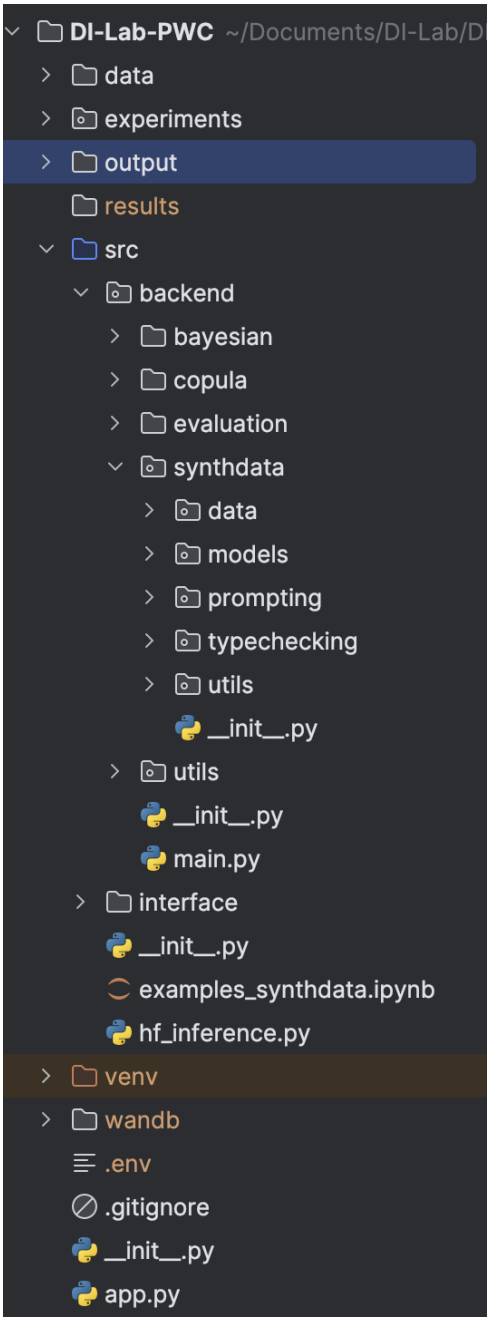

Figure 6: The file structure of our code base.

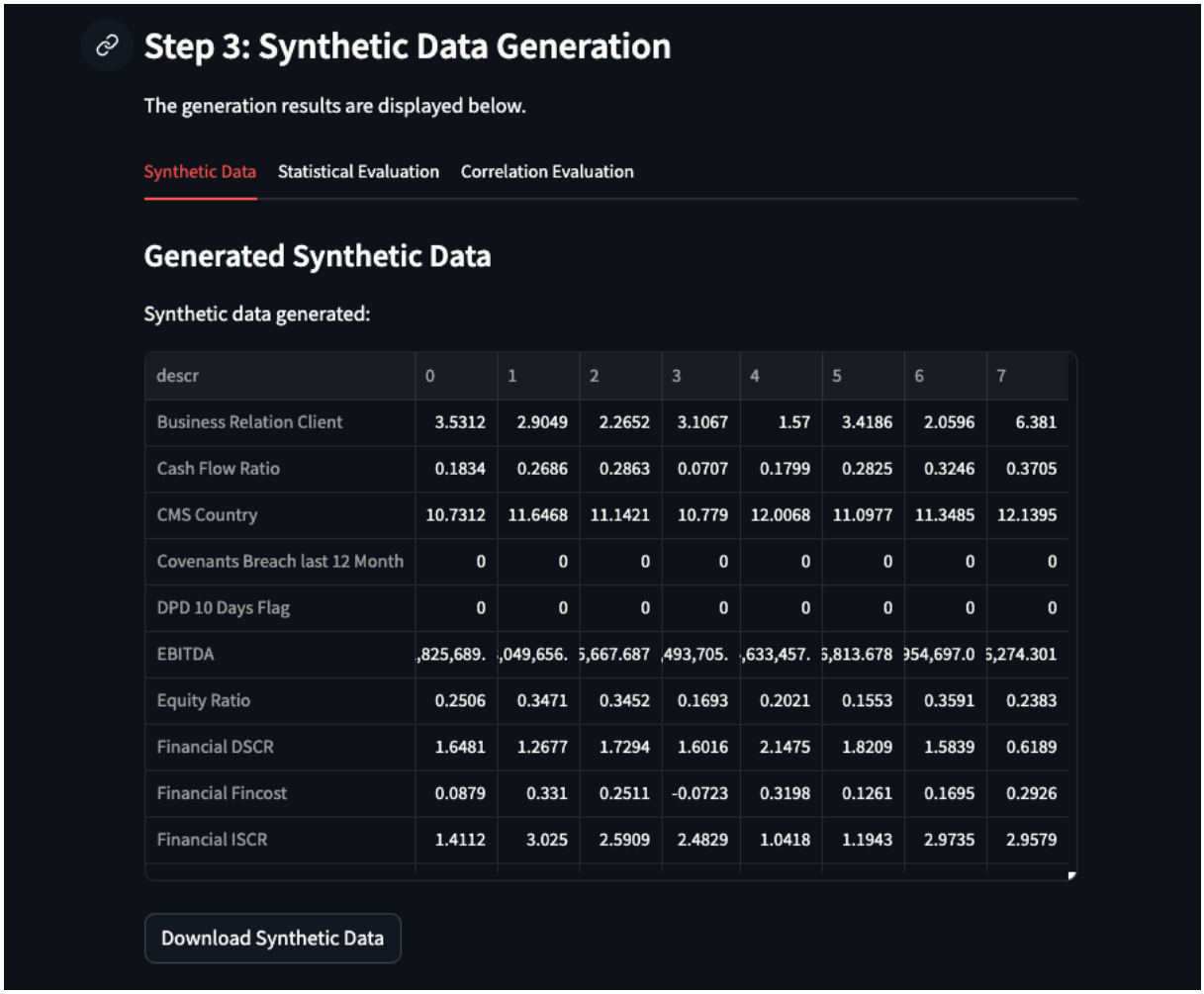

Figure 7: A sample of the generated data that our application produces.