A flexible synthetic data generation framework for tabular data

José Pinto[a,*], Ricardo Sousa[a]

[a]*LIAAD-INESC TEC, FEUP Campus, Street Dr. Roberto Frias, Porto, 4200 - 465, Portugal*

ARTICLE INFO

Keywords:
Data Synthesis
Synthetic Data Generation
Value Imputation
Machine Learning
Power Transformer
Mechanical Engineering

ABSTRACT

With the growing interest in artificial intelligence and in particular, deep learning, a lack of sufficient data to meet various needs is an ever increasing problem. Adding to this, issues of data privacy and property are ever more prevalent. An attractive solution to these problems is the generation of new synthetic data with the intended characteristics. As such, in this work, we present a new synthetic data generation framework for tabular data. We show its efficacy and flexibility in generating both independent and time series data. Furthermore, we complement this approach with a novel imputation-based data generator, which we show to be able to automatically generate great amounts of realistic data from a small sample, with this output being able to improve downstream model performance.

1. Introduction

In recent years, the field of Artificial Intelligence (AI) has been increasingly marked by its relation with data, and therefore the growing AI subfield of data science. The generation, transmission, manipulation and consumption of data have grown at an exponential rate, with heavily data-dependent pipelines and Machine Learning (ML) methods now constituting the state of the art in many fields Anderson, Kennedy, Ngo, Luckow and Apon (2014); Tsirikoglou, Kronander, Wrenninge and Unger (2017); Ekbatani, Pujol and Segui (2017); Devaranjan, Kar and Fidler (2020).

With the advent of deep learning, first brought into the limelight by the AlexNet Krizhevsky, Sutskever and Hinton (2012) neural network, the interest of developing and employing these tools has all but grown Ekbatani et al. (2017); Sun, Cuesta-Infante and Veeramachaneni (2019); Tripathi, Chandra, Agrawal, Tyagi, Rehg and Chari (2019). With the advances brought by these methods to fields such as computer vision, natural language processing and robotics, such surge in interest is no surprise Ekbatani et al. (2017); Martinez-Gonzalez, Oprea, Garcia-Garcia, Jover-Alvarez, Orts-Escolano and Garcia-Rodriguez (2020); Boikov, Payor, Savelev and Kolesnikov (2021); Shakeri, Santos, Zhu, Ng, Nan, Wang, Nallapati and Xiang (2020). However, all these different methods and applications have one thing in common, the necessity for extreme quantities of high quality (usually labeled) data Tsirikoglou et al. (2017); Ekbatani et al. (2017).

The gathering of high quality, sufficiently large datasets is a costly and time-consuming process, sometimes exacerbated by other constrains of accessibility or even existence (sometimes more data simply does not exist, e.g. data showing trends for multiple decades) Dahmen and Cook (2019); Zhang, Gonzalez-Garcia, Van De Weijer, Danelljan and Khan (2018); Sun et al. (2019); Boikov et al. (2021). Other times, when such data already exists, using it to train systems or providing it to third parties is not a viable option, due to privacy concerns or ownership issues K Dankar and Ibrahim (2021); Ayala-Rivera, McDonagh, Cerqueus and Murphy (2013); Abay, Zhou, Kantarcioglu, Thuraisingham and Sweeney (2018).

Given this underlying conflict (i.e. the desire to use data intensive methods while data may be scarce), it becomes obvious why there has been increasing interest in the development and usage of data augmentation and generation tools. If fact, such tools present not only the possibility of addressing the aforementioned problems, such as the lack of sufficient data or the impossibility of sharing sensitive private data due to legal or ethical limitations Soltana, Sabetzadeh and Briand (2017); Abay et al. (2018); Ayala-Rivera et al. (2013); but also many others, such as creating datasets with the required level of difficulty for teaching, validating algorithms on a variety of datasets with differing characteristics, creating pseudo-real data to fill under-development products and testing a variety of software systems and communication frameworks Houkjær, Torp and Wind (2006); Dahmen and Cook (2019); Soltana et al. (2017); Mannino and Abouzied (2019).

jose.f.pinto@inesctec.pt (J. Pinto)
josepedropinto.com (J. Pinto)
ORCID(s): 0000-0003-1019-6687 (J. Pinto)

However, the generation of synthetic data is not an easy process. Important characteristics of any data generation methodology include data throughput (amount of data generated per unit of time), data realism (similarity to the target or real data in terms of size and distribution), control and flexibility (allow the user to define complex intervariable relations of many kinds) and ease of use Houkjær et al. (2006); Dahmen and Cook (2019); Anderson et al. (2014); Krishnan and Jawahar (2016). Data realism in particular is one of the most difficult to achieve, while also being, for most applications, one of the most important, as the degree of data realism greatly affects the performance of ML systems Anderson et al. (2014); Tripathi et al. (2019); Martinez-Gonzalez et al. (2020).

It is taking into consideration the variety of existing problems and subject areas that we developed and now present a new synthetic data generation framework for creating realistic tabular datasets. In this framework, a great focus was placed on expansibility, flexibility, ease of implementation and use. To complement this framework, we also present a novel imputation-based synthetic data generation method to automatically expand small, limited datasets, while maintaining complex intravariable and intervariable relations and distributions.

Thus, with this in mind, the rest of this paper is organized as follows: (1) First, we present the related work in the synthetic data generation field at large. (2) Then, a description of the framework's design concepts, our proposed API and examples of some functions and operations are given. (3) An in-depth explanation of our imputation-based data generation method follows. (4) We complement this with examples of our framework in practice, as well as other results. (5) Finally, we discuss our main conclusions and future work.

2. Related Work

Within the various fields that utilize data as a critical part of their processes, there are many formats in which such data can appear. The most basic format is highly structured tabular data, where tables or relational tables, containing rows (entries) and columns (variables) allow easy access to all information. Semi-structured data can come in several forms, such as XML or JSON formats, which typically have a well-defined set of rules for the structure and relation of elements. Unstructured data are one of the most common and difficult to use data types, including text files, images and videos.

In general, the generation of synthetic tabular datasets consists of modeling the joint probability distribution of the data Sun et al. (2019). Modeling this distribution is critical to ensure data realism, with multiple competing approaches trying to accomplish this. In Mannino and Abouzied (2019) a Directed Acyclic Graph (DAG) is used to define the relations between different variables, ensuring that maintaining said relations is tractable. Tools such as Benerator Ayala-Rivera et al. (2013) and the approach by Jeske, Lin, Rendon, Xiao and Samadi (2006) provide a good baseline for this task. To improve upon these results, Sun et al. (2019) presented a new approach using vine copula models. As creating vine copula models requires multiple discrete decisions related to their structure, a Long Short-Term Memory (LSTM) network was trained through reinforcement learning. The work in Pei and Zaïane (2006) focuses on the task of testing clustering algorithms, presenting a generator capable of creating 2D datasets of clusters with various formats, densities and sizes, while presenting an interesting way to characterize their "difficulty". Extensive tests of 6 clustering methods were performed, with DBSCAN obtaining, in general, the best results.

Generating industrial-sized datasets (many terabytes) is also an important area of interest, with Hoag and Thompson (2007) showing their high-performance parallel computing focused synthetic data generator. They build on their previous work, by providing an highly flexible XML-based input language, Synthetic Data Description Language (SDDL). Preserving data privacy is another critical problem, where generator based approaches such as the one presented by Abay et al. (2018) have proven to be a viable alternative for anonymization. In this work, multiple deep learning autoencoders are used on a variety of datasets, not only anonymizing data but also allowing downstream ML models to achieve great results, which is critical for such a tool. Despite the variety of approaches, satisfying logical constraints (e.g., an end date being greater than a start date) is a very complex task. Soltana et al. (2017) takes a great stride forward in this aspect, by developing a method that automatically tweaks created data entries until they respect both statistical and logical constraints placed on the data.

The generation of time series data adds a new degree of difficulty, where previously independent data entries are now highly related. Much like for simple tabular data, there is a myriad of different approaches. In Dahmen and Cook (2019), nested Hidden Markov Chains (HMMs) are used to model the hierarchical nature (series inside series) of smart home datasets. A deep learning synthetic data generation approach is provided by Alzantot, Chakraborty and Srivastava (2017), where a combination of stacked LSTMs is used to create the statistical parameters (mean and variance) for a Mixture Density Network (MDN), which is in turn responsible for the final generation process. A further degree of complexity is created by the addiction of spatial relations to the data, on spatio-temporal datasets. To address this, an interpolation-based data generation approach is proposed by Yu, Ganesan, Girod, Estrin and Govindan (2003), with which it is possible to create values for locations/times that were not measured, permitting the generation of data points at new times and places, while preserving spatio-temporal data relations.

Given the large amount of synthetic tabular data generation approaches, application areas and strategies for integration into data driven pipelines, there are several questions relating to best practices that a practitioner might have. With this in mind, K Dankar and Ibrahim (2021) presents a set of results and guidelines for synthetic data generation. They address questions related to parameter optimization, transfer learning, relations between synthetic data quality measures and downstream model performance, and more. This is performed on a very healthy set of 15 datasets and 4 popular data generators.

On the other hand, the different and more complex structure of relational tables and semi-structured formats, such as JSON and XLM, give rise to the need for new algorithms. In the work of Houkjær et al. (2006), where synthetic relational datasets are created, concept graphs are used to enforce the intended variable distributions, primary/foreign key relations, and dependencies between the same row and entries of the same column. As an alternative, Anderson et al. (2014) focuses on the task of data quantity and anonymization. A large multi-TB Internet of Things (IOT) dataset is employed to automatically select the best statistical distributions and parameters for numerical variables, while unbalanced Huffman trees efficiently model categorical value frequencies. Much concern and detail was placed on the XML structure of the data, with the different paths in the XML documents being modeled, as well as their combinations and frequencies, with the variable models being independent for each path. Although inter-variable relations are not maintained, the method is highly efficient (generating TBs of data) and scalable.

In addition to tabular generation, the generation of synthetic images is one of the most common tasks. In some instances, the image generated is the final intended output, and a goal by itself; but also about as often, the generated images are merely a means to better train or validate computer vision algorithms. Most effective existing methods use one of two approaches; a deep learning approach, where images are generated by applying deep learning models on random noise; and a rendering approach, where typically game engines (already designed to create many high-fidelity images) are employed to create diverse scenarios and image sets.

COVID-19 detection in CT scan images is performed in Amin, Sharif, Gul, Kadry and Chakraborty (2021), which utilizes something quite novel in the field of deep learning, quantum computing, which not only allowed the obtention of the results faster, but also significantly improved model performance. In rendering-based methods, the addition of camera defects, such as motion blur and distortions, is often critical for downstream model performance. Both Planche, Wu, Ma, Sun, Kluckner, Lehmann, Chen, Hutter, Zakharov, Kosch et al. (2017) and Tsirikoglou et al. (2017) take this into account for generating depth images from CAD models and creating cityscape images from a procedural world, respectively. Devaranjan et al. (2020) takes a different approach to the rendering problem, using reinforcement learning to generate worlds from context-free grammars. They developed on their previous work by automatically extracting scene structure (number of objects of each type and relations; for example, 1 road with 2 cars) and distribution parameters (object location, orientation, etc.) from existing images. Unlike other works, this method of rendering image generation requires minimal manual work, being mostly automated.

The problem of generating data for robotic tasks is addressed in Martinez-Gonzalez et al. (2020), where Unreal Engine 4 is used. Unlike other works, a large variety of output image types is provided, including not only the typical RGB images, but also depth, segmentation and normal masks; while permitting real-time interaction with the space via Virtual Reality (VR) headsets. Although large labeled RGB image datasets are now quite common, other types of imagery can be very hard to come by. This is problem that Zhang et al. (2018) aims to solve, presenting a method to translate RGB images into Thermal InfraRed (TIR) images. To tackle this, a deep learning approach is used, where two Generative Adversarial Networks (GANs) are tested, Pix2Pix and CycleGAN. The usage of Pix2Pix led to the best results, with trained downstream models achieving improved performance by complementing the real data with the generated one.

Image compositing is another alternative for synthetic image generation, where different components of various images are combined to create a new one. The main challenges with this kind of approach stem from differing conditions (lighting intensity, direction, strength, color, etc.) between various source images, as well as artifacts that can be added during the compositing process. While in the work by Tripathi et al. (2019) these artifacts are embraced, with even more added to reduce their discriminatory ability for downstream models, other approaches such as Ekbatani et al. (2017) try their best to remove them by using a variety of transformations. At the same time, Boikov et al. (2021) uses blender to create and add defects to steel images. As both the base image and the defects are created in the same setting, with adjustments to lighting and other effects being performed later in the process, no compositing artifacts are present in the final result.

Finally, the generation of text, handwritten images and symbols is another area of great interest. Both Krishnan and Jawahar (2016) and Taranta, Maghoumi, Pittman and LaViola Jr (2016) address the problem of generating handwritten text and symbols. While the former applies changes to the distance between points, taking into consideration characteristics such as the velocity of the pen; the latter uses handwritten fonts and a set of transformations (shears, translation, line stroke width, etc.) to accomplish this task. Although the generation of handwritten text and symbol images is not easy, the creation of realistic language directly in text form is by no means simpler. Shakeri et al. (2020) tries to tackle the generation of question-answer pairs by employing an encoder-decoder transformer deep learning architecture, BART. Extensive validation tests were performed with several datasets and models, with the data generated by the new method allowing models to surpass state-of-the-art results in question answering systems.

From the large variety of application fields, problems and approaches, it should be clear to the reader that there is a lot of interest in the usage of synthetic data generation methods. Taking this into consideration, throughout the next section we will present our take on this problem by detailing our tabular data generation framework.

3. Generator Framework

There are many competing attributes, characteristics and capabilities that data generators strive for, which include data quantity and quality, user control, ease of use, ease of implementation, resource efficiency and efficacy. As we have shown before, various synthetic data generators focus on a few of these areas, neglecting others; while always showing some degree of efficacy in all of them. Taking this into account, we place our focus on four of these, data quality, user control, ease of use, and ease of implementation.

To accomplish this, we based our framework on a very simple core concept; only base variables (creating independent variables) and transformations (creating dependent variables) exist. Despite its simplicity, a wide array of data distributions and inter-variable relations can be easily defined. Furthermore, should some capability be found lacking or the generation of specific distributions to be burdensome, new base variables and transformations can very simply be added.

To tie the different base variables and transformations together, we employ a pipeline which applies these simple operations in the order they are added. As any practitioner familiarized with the fields of data science or AI in general would be familiar with the concept of using pipelines, it should be a very simple concept to understand and utilize. A diagram of a typical generator pipeline can be seen on figure 1.

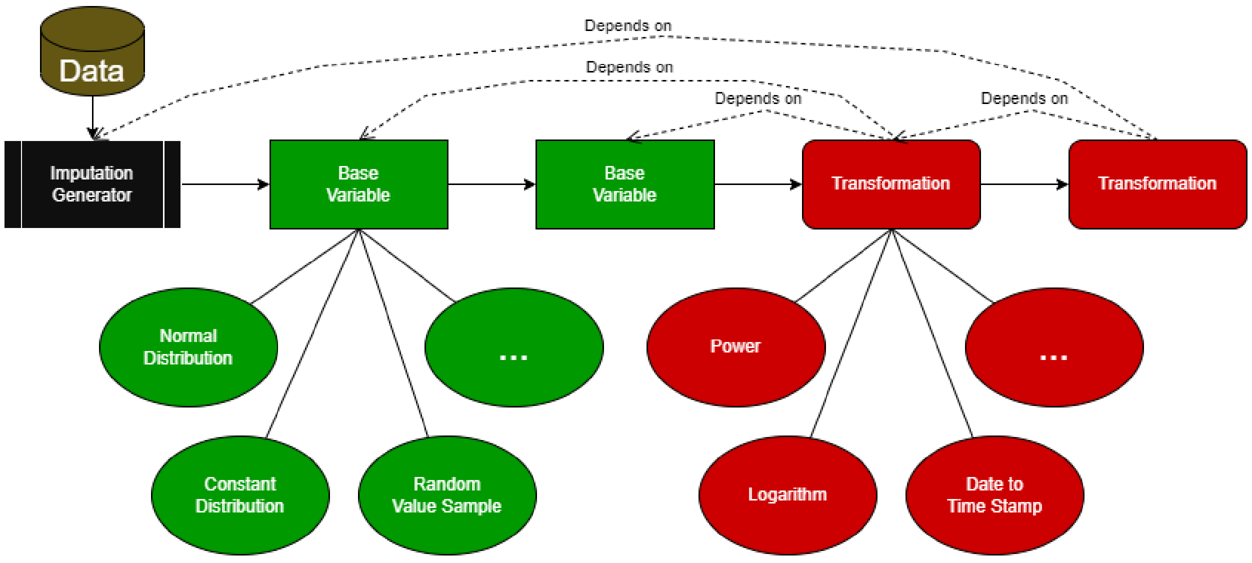

Figure 1: Diagram of an example generator pipeline.

Base variables can thus be divided into two categories, tabular and time series. We will not enumerate all base variables and transformations that we have implemented, as by the very nature of the framework, which ones are, as well as the total amount, can change to meet the user's required needs. Nevertheless, we will present a few in order to provide a good understanding of what should be expected for each kind. Tabular base variables include as such, statistical distributions, like normal, exponential, and Poisson, as well as random sampling of categorical variables or creation of random timestamps and dates. Time series variables, on the other hand, include counters, random increments, increasing dates and times according to some distribution (for example, Poisson) and Markov chains for the generation of categorical variables.

It bears mentioning that for both the base variables previously described, as well as the transformations which we will soon describe, the various parameters, such as, statistical moments, transition functions, offsets and extremes are defined at the time the operations are added to the pipeline.

Taking this into consideration, the set of transformations is significantly larger. It includes algebraic transformations, such as exponential and logarithm; arithmetic, such as addition or multiplication by a value; trigonometry, such as sin, tanh, arccosh, conversion to radians; addition and multiplication of columns; rounding and floor operations; clipping, addition of noise, such as normal noise, transients and swapping of categorical values; and many others.

In addition to the various operation parameters mentioned above, which change on an operation-by-operation basis, there are two parameters that are always considered for transformations. These are the names of the columns that will be transformed and the names of the output columns. Depending on the nature of the transformation, output columns' names may be automatically generated or a column may be transformed in place, for ease of use.

It can be seen that with sufficient variety in the supplied operations, an unbounded degree of control and complexity of the output datasets is achievable. However, the main drawback of this approach is that it requires a thorough understanding of the intended data characteristics, as well as, a good understanding of how to achieve them with a combination of different operations. As can be imagined, depending on the complexity of our intended results, this can be a difficult task; a difficulty that we aim at easing with the data driven approach detailed in the next section.

4. Imputation based data generation

The usage of data-driven ML models for the task of data generation is nothing new, with a variety of deep learning, classical ML and interpolation-based approaches existing. However, to our knowledge, we are the first to propose an imputation-based approach for data generation. Even if such an approach exists, unbeknown to us, it is surprising that it would be so little widespread, as imputation seems to lend itself very well to data generation. In fact, missing value imputation is already generating data, if only in the more limited setting of filling partial rows. Therefore, harnessing the power of imputation-based algorithms for data generation seems to have great potential.

Given this, we propose a method that would function as a wrapper to any imputation method, allowing its capabilities to be used for synthetic data generation. It works the following way: Given a dataset, firstly the imputation method is trained on the data. Then any missing values in the original data are imputed to create a complete dataset. For each entry to be generated, a seed row is created by randomly sampling a row from the original data and M variables. Finally, the values for the rows that were not sampled are filled using the imputation method. This process is repeated to generate an arbitrary number of new entries.

It should be clear that the selection of the imputation method is critical, with an important property being that such method should be non-stochastic, as this will increase the variety and quality of the output. Therefore, we employed an imputation method which was developed in one of our previous works. This method presents two crucial improvements for the task at hand over other existing imputation methods. The first is that it is non-stochastic, introducing some degree of randomness and variability of the output, even for the same inputs. The second is that it works under a different principle from other imputation methods. Almost all imputation methods work, much like regular regression and classification methods, by trying to reduce the overall error rate of imputed values (difference between imputed and real values). Although this error reduction is very important it tends to introduce some degree of bias and not preserve variable and inter-variable distributions. Our method tries to better preserve variable distributions at the cost of increased error, which we believe could lead to increased downstream model performance. As a mock example, we could imagine a normal distribution. In an error reduction approach, a method would always predict the mean (if there is no more information), while a distribution-preserving approach would randomly create normal samples (which would very likely increase the overall error).

As we have already extensively detailed our method in our previous work, now only a brief overview will be provided. This method, which we named Sampling Imputation (SI), works in the following way: For each distinct variable subset of size K, regression and/or classification models are trained. All models are stored together with their performance metrics, R^2 for regression and accuracy for classification[1]. To impute new values, first valid models (with the intended variable as target and other non-missing variables as features) are identified, then from these a random model is sampled, with the likelihood of being picked proportional to its performance, and finally, the value is imputed according to the model's prediction.

Given the fact that multiple models are created by using data subsets, the similarities to ensemble-based methods are clear. The main difference is that when it comes to imputing a new value, instead of combining the different predictions, one at random is selected. This can therefore be seen as, instead of averaging the predictions per value, as averaging over the whole dataset. Similarly to ensemble methods, any number of identical or different methods can be combined, generating new values with the intended amount of variety. The main drawback of this method is the computational power required to train all the different models.

The astute reader might have noticed a limitation with the described approach, a limit on the unique number of seed rows that can be created. There are several options to raise this limit. In increasing order of effectiveness, (1) obtain data with a higher number of rows or columns N; (2) set the number of sampled columns M to $\lceil\frac{N}{2}\rceil$; (3) set the number of columns used per model K to $\lceil\frac{M}{2}\rceil$; (4) set M to $\lceil\frac{2N}{3}\rceil$ and K to $\lceil\frac{N}{3}\rceil$; (5) use all combinations of M and K such that M<N and K<M; (6) increase the number of unique models for each column combination. While using only option 5, the limit on the number of unique generated rows is given by equation 1. As an example, if we have a dataset with 1000 rows and 10 columns, this limit would be $5.8e^7$ (about 58 million).

$$R_{lim} = \binom{N}{M} \times \binom{M}{K} = \frac{N!}{M!(N-M)!} \times \frac{M!}{K!(M-K)!} = \frac{N!}{K!(N-M)!(M-K)!} \tag{1}$$

Given this, it should be clear that the limitation on the number of unique entries can be easily surpassed, with only the limitation in terms of computational time remaining. With this addressed and as we have thoroughly detailed our methodology, giving the reader an in-depth understanding of our approach, framework and methods, throughout the next section we will present examples of our proposed API, generation capabilities and other results.

5. Results

Data can come in an effectively endless variety, with any combination of characteristics, such as distributions, relations, number of variables, etc. Therefore, it would be impossible to enumerate all possible dataset structures, which then extends to all use cases of an effective data generator. Given this, we will focus on showing the flexibility of our proposed framework, and how with just a simple set of operations we can create any desired dataset. We will start by presenting two simple generation examples, one of time series and another of independent tabular data, where both the code and visualization of the output will be inspected. Throughout the remainder of the section, we will perform a more in-depth analysis of an application of our imputation generator, comprised of data similarity metrics, comparison plots and the effects on downstream model performance.

In signal analysis, most common types of data have simple sinusoidal structures with some sort of perturbance. These perturbations include interharmonics (multiple overlapping signals with different amplitudes and frequencies), transients (sudden changes in the values), notches (cuts in the sinusoidal structure), among others. From these we opted to generate an example of "sag", where the amplitude of the wave temporarily decreases. The code to generate these data can be found on listing 1, while the output in figure 2.

There are a few important aspects to note regarding our proposed API and the way in which we generated this signal, as shown in the code listing. The first point of interest is that all the different operations are defined in VARIABLES and TRANSFORMATIONS. Although this is not the most typical approach, it has two advantages in terms of usability, related to Integrated Development Environments (IDEs). The first is that most IDEs contain autocomplete features, and thus would automatically provide a listing of all operations. The second is that it allows direct access to information about the operation (parameters, docstrings, etc.). Both reduce the need for external documentation or peeking inside

[1]Since the original data may have missing values, different models may have various sizes of validation data, with the performance not being directly comparable. To solve this, the performance values saved are the lower bound of the Wilson score confidence interval.

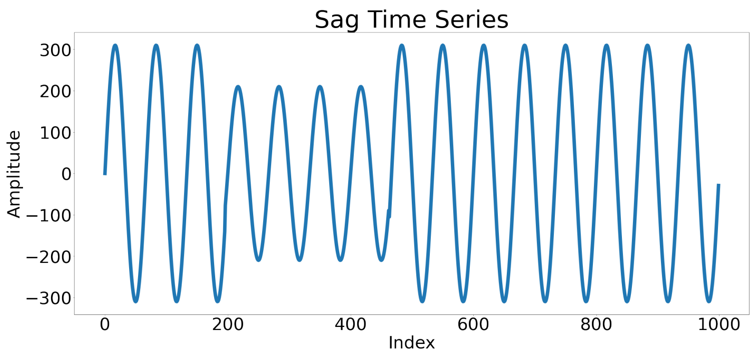

Figure 2: Example of signal with "sag".

the code, which greatly enhances ease of use. The second and last point regards the way in which we created the signal, in which Markov chains were used to create state transitions, while each of the states (two signals with the same frequency and different amplitudes) were individually created. Using this approach, any two signals can be switched or interpolated in accordance with our required parameters, and therefore it provides a great degree of flexibility. It would, of course, be possible to create a single operation that would generate this kind of data; however, we consider it to be both more important and interesting that many different datasets can be generated from much simpler operations.

Building upon these ideas, the next generation example shows the same principles in action; with independent tabular data being created. This example, however, requires some more background, with an understanding of said background being important not only for this example, but also for the remainder of this section. The necessity of creating this generator was brought about by a problem related to one of our previous works, and therefore it only makes sense to use the same target domain; power systems in general and power transformers in particular. There are a variety of problems related to power transformers, with two of the most common being fault diagnosis and Health Index (HI) prediction, with this next example pertaining to the latter. Predicting HI entails identifying a single number or value which can represent the overall condition of a system (a power transformer for our case), with typical numerical values ranging between 0 and 5, while categorical ones, between very bad and very good. When designing systems to predict HI, experts' opinions are typically considered ground truth, with models trying to approximate it using a series of measured features.

For this example, we do not aim at creating a new HI prediction model, but instead at replicating a model for which all parameters are already available. Given this, we chose one of the models presented in Zuo, Yuan, Shang, Liu and Chen (2016); the one with the highest number of features, which although does not present the best results, seems to be the most interesting to replicate, given its complexity. The model in question is of binary logistic regression with the full function definition being given by equation 2, where x1 to x6 represent respectively water content, acidity, oil breakdown voltage, dissipation factor, dissolved combustible gases and 2-Furfuraldehyde content.

$$HI = \frac{1}{1 + e^{(5.45 - 0.246_{x_1} + 28.604_{x_2} - 0.171_{x_3} + 0.112_{x_4} + 0.01_{x_5} + 0.254_{x_6})}} \tag{2}$$

To carry out the generation process, we first need to generate each of the individual features, then perform a linear combination of them according to the given coefficients, and finally apply the remaining mathematical operations to perform the logistic function. The first step is the most complex, requiring the selection of a distribution and the respective parameters. To identify these, we used a dataset that was provided to us by Efacec, which is thoroughly detailed in our previous work. It contains not only the features required by this model but many others, while having more data entries (about 1000) than those used in the Zuo et al. (2016) paper (about 30). Given this, we selected the normal distribution for all features and used the mean and standard deviation of these features extracted from our data. The code to generate this data can be seen on listing 2, where the means, standard deviations and various coefficients have already been filled in, while the features can be seen in figure 3 and the final HI in figure 4.

As with the previous example it would be possible to create a single operation that does the same as the set used, for example, a single logistic transformation. Conversely, even without the "Linear" transformation which does the linear

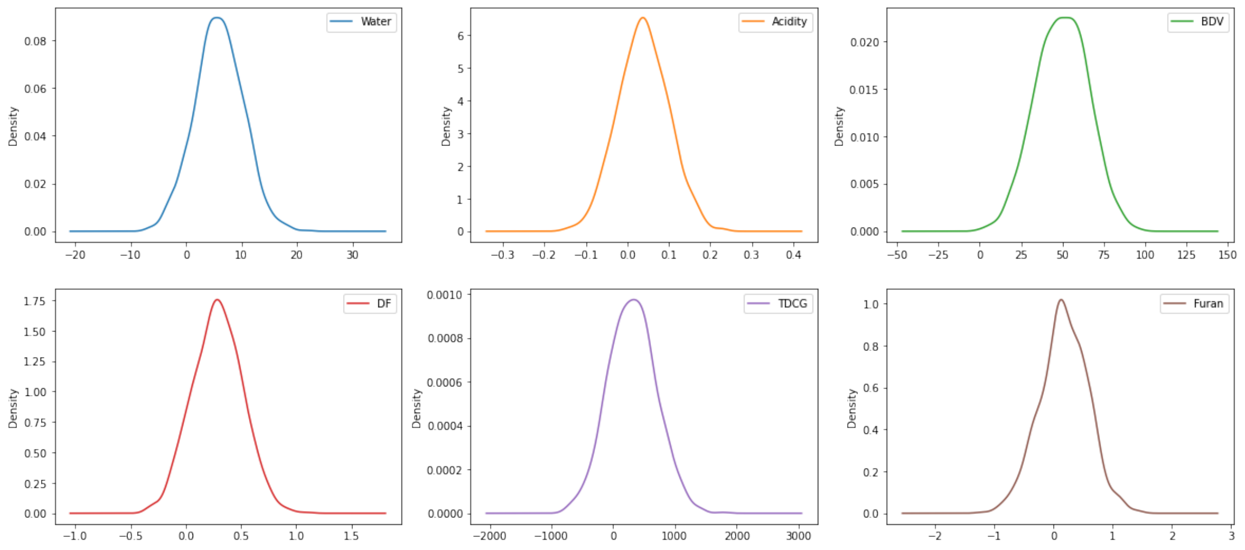

Figure 3: Density plot of features generated to predict HI.

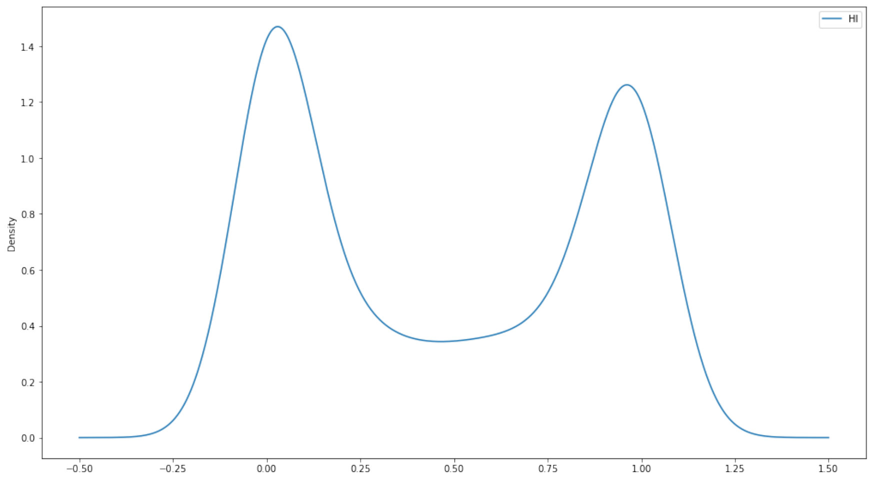

Figure 4: Density plot of HI generated from logistic model.

combination of the features, the same result could be achieved with a set of sums and multiplications of columns and scalars. This number of options should reinforce the flexibility of our approach and be a strong proof of its efficacy.

Although the two previous examples have been relatively simple, the next one, which relates to the usage of the imputation generator will be more in depth, as to show its efficacy, a far more detailed analysis is required. For this case we aim at generating more entries of the previously mentioned Efacec provided power transformer dataset which contains over 30 variables, only 1000 entries and a very large number of missing values; with some columns having over 80% of values missing. We will discuss the preprocessing and generation process, do a graphical comparison of generated and real distributions and relations, showcase similarity metrics, and finally inspect the effects of using the generated data on downstream model performance.

The first step in our approach involves some data preprocessing. The bare minimum required was done, which should show that our methodology functions with minimal manual work. This preprocessing consisted of standardizing categorical values (some values are different but represent the same thing), changing numerical values from scientific notation to regular floats, changing "," to "." in float values and other minor non data changing alterations. Then the generator was created, and the generation process was started. To maximize the quality of the synthetic data, the number of models trained was similarly increased as far as our computational resources allowed; with training taking roughly a week. A total of 500000 (half a million) data rows were generated, with this taking about 18 hours.

The first step to ascertain the quality of the generated data is, as stated above, a graphical analysis. As there are more than 30 variables, an exhaustive investigation of each is beyond the scope of this work, with the same being even more true for relations between different variables. Therefore, we selected one representative example of each to inspect; each showing the strengths but also possible limitations or problems of the approach.

The first example can be seen in figure 5, where the distributions of the numerical variable Carbon Dioxide (CO) are compared. It is very clear that the distribution of the generated data is extremely similar to the real one, with there being some tendency to the mean; this tendency was identified to be inversely proportional to the number of used models.

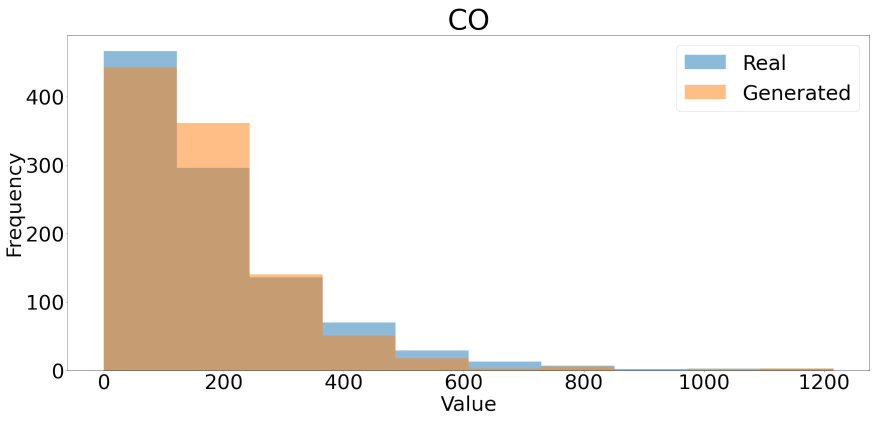

Figure 5: Histogram comparing real and generated values of Carbon Dioxide (CO).

The second example, in figure 6, is very similar to the first one, with the values of a categorical variable, Oil Type, being compared. The frequency of the values here is even more similar than for the case of numerical variables, with this being the case for all other categorical variables. Although we might expect a trend to the mode, similar to the trend for numerical variables, and as this bar plot seems to confirm, in practice we concluded that this was not the case, with the variations being attributed to what is, in our opinion, just an effect of the random nature of the method; as other variables or samples of the same variable do not present this effect.

To inspect whether the inter-variable relations are preserved by our generator, we created sets of scatter plots. Figure 7 shows the relation between CO and Carbon Dioxide (CO2) for both real and generated data. This pair was selected because it has a relatively high correlation, which means that any discrepancies should be easier to detect. Overall, we conclude that the relations are very similar, with the correlation for the generated data being somewhat attenuated, with a larger spread than for the real data.

To further analyse the quality of the generated data we searched for data similarity metrics. An ideal metric would present the following set of qualities: (1) improve as variable distributions become more similar, (2) improve as intervariable relations grow closer, (3) be independent of the order of rows in each dataset, and (4) Converge as the size of the samples for each dataset increases. Although there are many data similarity metrics, only one was found to satisfactorily fulfill these requirements, propensity. As described in K Dankar and Ibrahim (2021), propensity employs a model in a binary regression problem (predict in the range [0,1]) that entails for each data sample identify if it is real or not. Further more, the degree of "confidence" that the model has is taken into account, with predictions closer to 0.5 indicating less confidence. The formula for propensity is given by equation 3, where $\hat{p}$ is the predicted value.

$$pMSE = \frac{1}{N}\sum_{i}(\hat{p}_i - 0.5)^2 \tag{3}$$

Therefore, the value of propensity lies in $[0, 0.25]$, with a lower value indicating a higher similarity between the data samples. One problem that was identified is that sometimes models give very confident incorrect predictions, with it

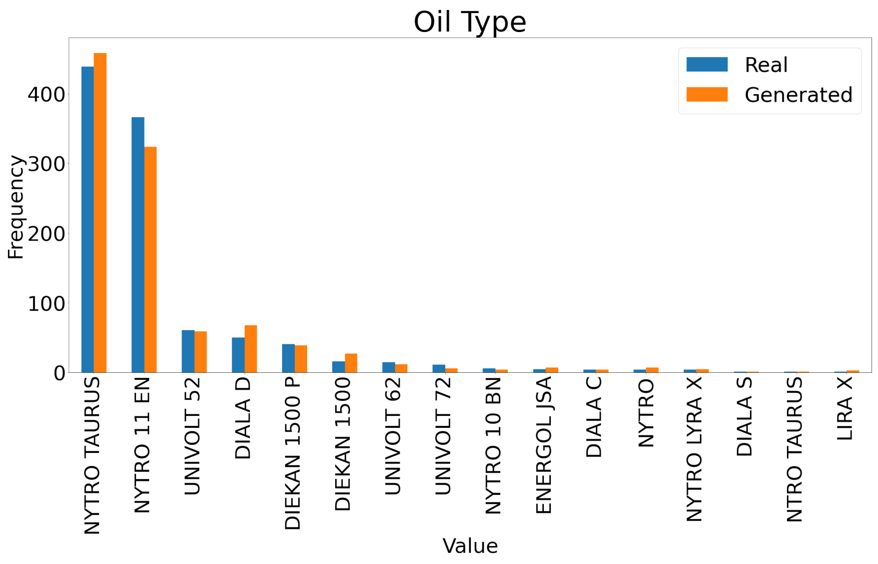

Figure 6: Histogram comparing real and generated values of Oil Type.

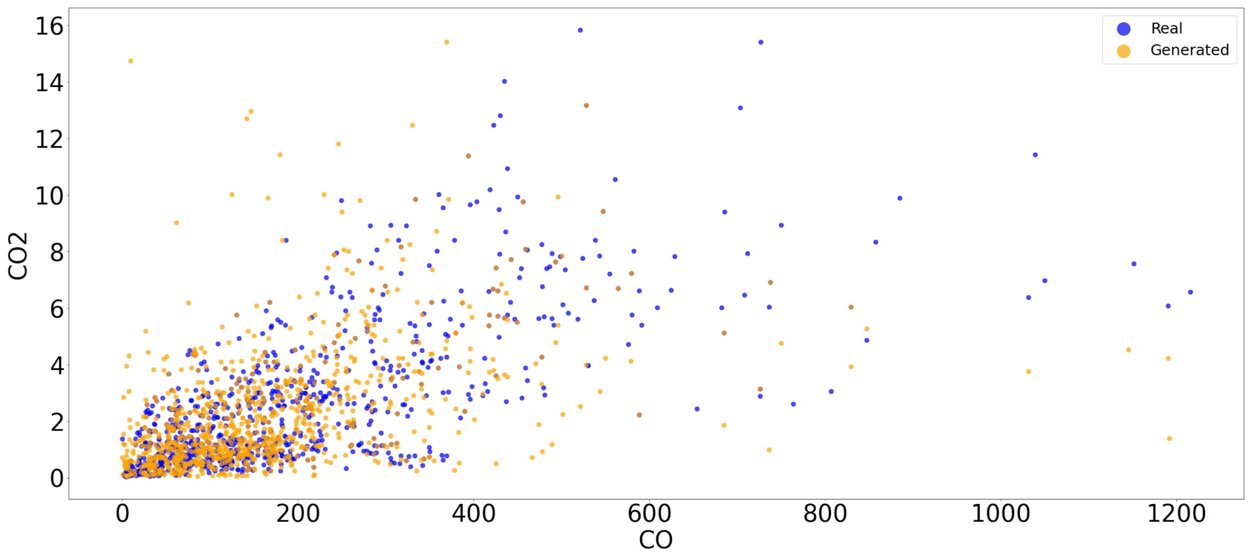

Figure 7: Scatter plot comparing the relation between CO and CO2 in the real and generated datasets.

being possible that a model presents 0% accuracy while achieving a minimum (best) value for propensity. To counteract this, whenever a model made an incorrect prediction, the value of $\hat{p}$ was considered to be 0.5, which indicates minimal confidence. The choice of model to calculate propensity is also very important, where logistic regression is a very common alternative, with some recent results using decision trees, such as K Dankar and Ibrahim (2021). Much like in the aforementioned work, we found that logistic regression did not achieve sufficient discriminatory performance, with accuracy around 50% even for mock datasets; we selected as such decision trees.

At this point we need to look back at the real data, which as we have mentioned has a very large quantity of missing values. This poses a problem, as missing values present very easy to learn discriminatory information. Although we could generate data with missing values, this would impede the training of downstream models, as most require completely filled data. Discarding entries with missing values is also not viable, as only 16 data rows would

Table 1
Table containing similarity metrics for different datasets. Bold indicates best value ("Real" column excluded).

Metric	Real	Constant	Simple Distribution	Generated	Generated, shuffled
Propensity ↓	0	0.25	0.25	**0.193**	0.215
Accuracy (%) ↓	50	100	100	**77.1**	85.9

remain. Furthermore, filling the spaces with simple values, such as 0 or the mean, would still lead to great unintended discriminatory ability. Given this, we recall that one of the first steps of our generator is to fill the source (real) data using the trained imputer. Therefore, we opted to use this filled data, as since any filled values should present the same characteristics as the generated data, only the real data components would be useful to differentiate between real and generated datasets.

Other than calculating the propensity of the generated data we also tested two mock datasets. The first simply uses a **constant** for all values in each column, with the mean for numerical variables and mode for categorical ones. The second defines **simple distributions** for each column, extracting the required data parameters from real data to create normal distributions for numerical variables, and random sampling of values for categorical ones. A final, fourth dataset was also created from the generated data, by randomly and independently shuffling the values for each column, thus maintaining distributions, but destroying relations. From this set, we would expect the simple distribution dataset to outperform the constant one, since the data distributions are better represented. Then, we would expect the shuffled generated data to further improve upon this, showing even more similar variable distributions. Finally, we would expect the untampered generated data to obtain the best results, thus proving that a significant degree of variable relations is maintained.

Table 1 shows the different propensity scores for each of the aforementioned datasets. For completeness, we opted to also include model accuracy and the values for the real data (by comparing the real data with itself). The obtained metrics are mostly in accordance to our expectation, except for the two mock examples, for which the decision tree managed to archive perfect accuracy, indicating that the datasets are not very similar. For the real data, although accuracy is exactly as expected, propensity is not necessarily, since, as we noted, sometimes models provide very confident inaccurate predictions. When we tested logistic regression, even with less than 50% accuracy, the propensity score was approximately 0.13.

Finally, in this section, we will present the effects of the generated data on downstream model performance. This was carried out in the form of fault diagnosis, where the fault type (from amongst 7) was predicted in a supervised multiclass classification problem. To do this the results from 3 different datasets were compared, real, generated and mixed, containing a combination of real and generated data. In all instances, the real data was used for testing, with 5-fold cross validation being used to split the real data. To better contrast the different approaches, only the bare minimum pre-processing was applied, which was done equally for all datasets. This preprocssing involves only 2 steps, first, the data are balanced in accordance to the target ("fault" columns) using SMOTE and random undersampling, keeping the data size constant. Then, all categorical columns are one hot encoded. The first step is required because without it, caused by the extreme data imbalances, only one class is always predicted, which does not allow any differentiation between the different datasets. Conversely, the second step is also mandatory since the models cannot directly use categorical values.

Given the goals of data generation tasks, the full dataset might not fit in memory in some settings, and any model used should be capable of iterative streamed training. The most commonly effective methods for tabular data, such as decision tree ensembles and many Support Vector Machine (SVM) variants, are not capable of this. Therefore, we choose to employ a simple Artificial Neural Network (ANN) model, Multi Layer Perceptron (MLP). Much like for preprocessing, the models' hyperparameters were kept the same for each dataset, and a minimal amount of optimization was performed. Thus, all parameters, except hidden layer sizes, were kept at the default values for PySpark, while the aforementioned hidden layer sizes were set to (64, 64, 32, 32, 16). This approach, with all the self-imposed limitations, is, of course, not conducive to state-of-the-art model performance; as with such a goal, extensive preprocessing, including feature selection and engineering; model comparison and selection; and significant hyperparameter optimization would be required, which would significantly hinder our ability to evaluate the quality of the generated data.

Taking this into account, the train and test accuracy curves for real data can be seen in Figure 8, where it is obvious that the test accuracy quickly plateaus at around 30%. When inspecting Figure 9, which contains the same set of results for the generated data, a significant performance improvement is evident, with accuracy reaching 50%. For the mixed dataset, the results shown in Figure 10 are almost identical, showing that the addition of some real data made no significant difference. With the jump in accuracy from 30% to 50%, the utility of the generated data for downstream model performance is clear.

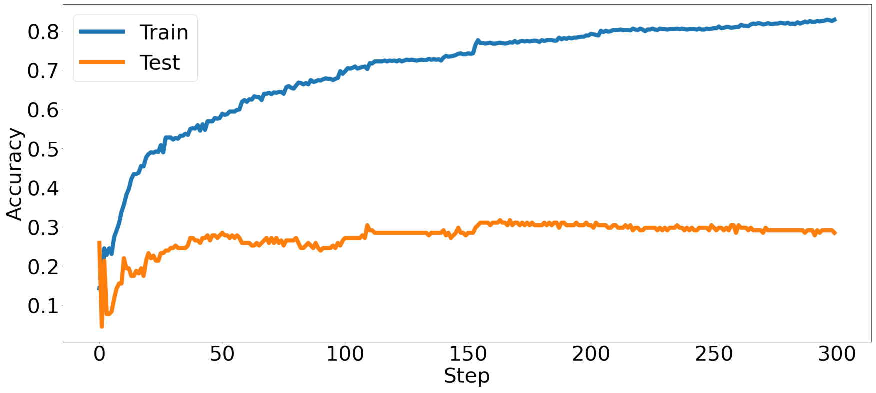

Figure 8: Train and test accuracy for training in real data.

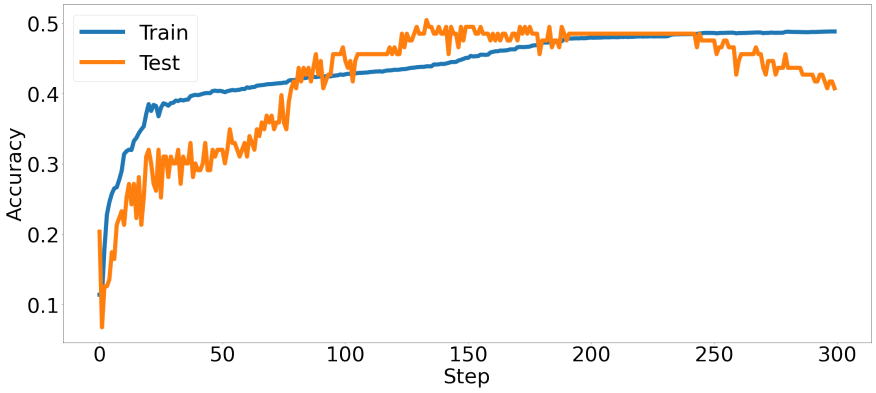

Figure 9: Train and test accuracy for training in generated data.

6. Conclusion and Future Work

We began this work with the objectives of creating a very flexible synthetic data generator that is both easy to use and implement. Towards these goals we developed our proposal for a framework and synthetic data generation system, which is complemented with an imputation-based data generation method. In general, we consider our objectives to be fully met and our approach to be a viable alternative to other existing systems.

As shown throughout the results section, the basic framework allows the generation of complex interdependent tabular datasets and time series, which is accomplished by the combination of simple operations working as building blocks. The imputation based generation greatly furthers the utility of this approach, by automatically generating data with the desired characteristics of a source dataset with a great degree of fidelity, while requiring no manual work or fine optimization. The synthetic data generated by this approach was shown to not only be robust to visual analysis but to also permit the improvement of downstream model performance.

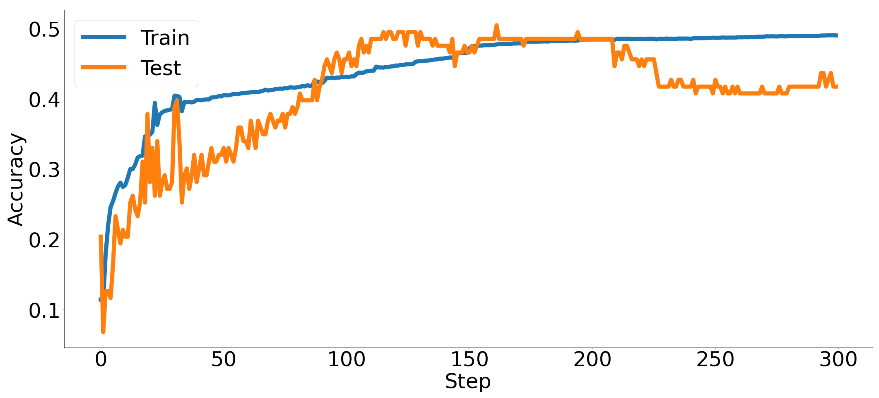

Figure 10: Train and test accuracy for training in mixed real-generated data.

6.1. Limitations

Of course, no work is without its limitations. One such limitation is the computational cost of this approach. Although high throughputs were not one of our primary goals, improving this aspect would definitely aid in the utility of this framework.

The imputation-based generator is also limited in the fact that, unlike the rest of the framework, it is not suitable to generate time series. Furthermore, given the need to create a large number of models, even if computational time is not a constraining factor, memory limits may be reached.

6.2. Future Work

Given the aforementioned limitations, one future avenue of research involves increasing the parallel computing compatibility of the tool for time series (it is trivial for independent tabular data). Possible solutions include working on independent subtrees, taking advantage of the fact that not all variables may be related; or using pipelining, with different compute units working on different operations. To expand the capability of the imputation-based generator to time series, a combination of interpolation and time series generation models, such as Long Short Term Memory (LSTM) networks, seems promising.

Other than improvements to the existing systems, we also find it important to inspect the effectiveness of the framework on a wider set of datasets from various fields and do user studies to better understand usability bottlenecks. This would help to improve the design and the proposed API further. Finally, investigating the possibility of better and more easily enforcing logical constraints by applying pre-processing and post-processing transformations to the data, could garner critical insights; for example, preserve a restriction $A > B$ by using the values of B and A-B.

7. Acknowledgments

This work was supported by Fundo Europeu de Desenvolvimento Regional (FEDER) and Programa Operacional Competitividade e Internacionalização through the project TRF4p0 - TRANSFORMER4.0: DIGITAL REVOLUTION OF POWER TRANSFORMERS with reference POCI-01-0247-FEDER-045926.

References

Abay, N.C., Zhou, Y., Kantarcioglu, M., Thuraisingham, B., Sweeney, L., 2018. Privacy preserving synthetic data release using deep learning, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer. pp. 510–526.

Alzantot, M., Chakraborty, S., Srivastava, M., 2017. Sensegen: A deep learning architecture for synthetic sensor data generation, in: 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), IEEE. pp. 188–193.

Amin, J., Sharif, M., Gul, N., Kadry, S., Chakraborty, C., 2021. Quantum machine learning architecture for covid-19 classification based on synthetic data generation using conditional adversarial neural network. Cognitive Computation , 1–12.

Anderson, J.W., Kennedy, K.E., Ngo, L.B., Luckow, A., Apon, A.W., 2014. Synthetic data generation for the internet of things, in: 2014 IEEE International Conference on Big Data (Big Data), IEEE. pp. 171–176.

Ayala-Rivera, V., McDonagh, P., Cerqueus, T., Murphy, L., 2013. Synthetic data generation using benerator tool. arXiv preprint arXiv:1311.3312 .

Boikov, A., Payor, V., Savelev, R., Kolesnikov, A., 2021. Synthetic data generation for steel defect detection and classification using deep learning. Symmetry 13, 1176.

Dahmen, J., Cook, D., 2019. Synsys: A synthetic data generation system for healthcare applications. Sensors 19, 1181.

Devaranjan, J., Kar, A., Fidler, S., 2020. Meta-sim2: Unsupervised learning of scene structure for synthetic data generation, in: European Conference on Computer Vision, Springer. pp. 715–733.

Ekbatani, H.K., Pujol, O., Segui, S., 2017. Synthetic data generation for deep learning in counting pedestrians., in: ICPRAM, pp. 318–323.

Hoag, J.E., Thompson, C.W., 2007. A parallel general-purpose synthetic data generator. ACM SIGMOD Record 36, 19–24.

Houkjær, K., Torp, K., Wind, R., 2006. Simple and realistic data generation, in: Proceedings of the 32nd international conference on Very large data bases, pp. 1243–1246.

Jeske, D.R., Lin, P.J., Rendon, C., Xiao, R., Samadi, B., 2006. Synthetic data generation capabilties for testing data mining tools, in: MILCOM 2006-2006 IEEE Military Communications conference, IEEE. pp. 1–6.

K Dankar, F., Ibrahim, M., 2021. Fake it till you make it: Guidelines for effective synthetic data generation. Applied Sciences 11, 2158.

Krishnan, P., Jawahar, C., 2016. Generating synthetic data for text recognition. arXiv preprint arXiv:1608.04224 .

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems 25.

Mannino, M., Abouzied, A., 2019. Is this real? generating synthetic data that looks real, in: Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology, pp. 549–561.

Martinez-Gonzalez, P., Oprea, S., Garcia-Garcia, A., Jover-Alvarez, A., Orts-Escolano, S., Garcia-Rodriguez, J., 2020. Unrealrox: an extremely photorealistic virtual reality environment for robotics simulations and synthetic data generation. Virtual Reality 24, 271–288.

Pei, Y., Zaïane, O., 2006. A synthetic data generator for clustering and outlier analysis .

Planche, B., Wu, Z., Ma, K., Sun, S., Kluckner, S., Lehmann, O., Chen, T., Hutter, A., Zakharov, S., Kosch, H., et al., 2017. Depthsynth: Real-time realistic synthetic data generation from cad models for 2.5 d recognition, in: 2017 International Conference on 3D Vision (3DV), IEEE. pp. 1–10.

Shakeri, S., Santos, C.N.d., Zhu, H., Ng, P., Nan, F., Wang, Z., Nallapati, R., Xiang, B., 2020. End-to-end synthetic data generation for domain adaptation of question answering systems. arXiv preprint arXiv:2010.06028 .

Soltana, G., Sabetzadeh, M., Briand, L.C., 2017. Synthetic data generation for statistical testing, in: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE. pp. 872–882.

Sun, Y., Cuesta-Infante, A., Veeramachaneni, K., 2019. Learning vine copula models for synthetic data generation, in: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 5049–5057.

Taranta, E.M., Maghoumi, M., Pittman, C.R., LaViola Jr, J.J., 2016. A rapid prototyping approach to synthetic data generation for improved 2d gesture recognition, in: Proceedings of the 29th Annual Symposium on User Interface Software and Technology, pp. 873–885.

Tripathi, S., Chandra, S., Agrawal, A., Tyagi, A., Rehg, J.M., Chari, V., 2019. Learning to generate synthetic data via compositing, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 461–470.

Tsirikoglou, A., Kronander, J., Wrenninge, M., Unger, J., 2017. Procedural modeling and physically based rendering for synthetic data generation in automotive applications. arXiv preprint arXiv:1710.06270 .

Yu, Y., Ganesan, D., Girod, L., Estrin, D., Govindan, R., 2003. Synthetic data generation to support irregular sampling in sensor networks. GeoSensor Networks 1, 211–234.

Zhang, L., Gonzalez-Garcia, A., Van De Weijer, J., Danelljan, M., Khan, F.S., 2018. Synthetic data generation for end-to-end thermal infrared tracking. IEEE Transactions on Image Processing 28, 1837–1850.

Zuo, W., Yuan, H., Shang, Y., Liu, Y., Chen, T., 2016. Calculation of a health index of oil-paper transformers insulation with binary logistic regression. Mathematical problems in engineering 2016.

8. Appendix

```python
N = 1000

gen = Generator()
gen.add_variable(VARIABLES.Counter, name="base_idx", start_value=0)

# create state transitions
gen.add_variable(
    VARIABLES.MarkovChain,
    name="states",
    start_value="A",
    states=["A", "B"],
    transitions=[("A", "B", 0.001), ("B", "A", 0.005)],
)

# create state 1
gen.add_transformation(
    TRANSFORMATIONS.Mult,
    in_columns=["base_idx"],
    out_columns=["h1"],
    value=(15 * 2 * pi) / N,
    drop_input=False,
)
gen.add_transformation(TRANSFORMATIONS.Sin, in_columns=["h1"])
gen.add_transformation(TRANSFORMATIONS.Mult, in_columns=["h1"], value=310)

# create state 2
gen.add_transformation(
    TRANSFORMATIONS.Mult,
    in_columns=["base_idx"],
    out_columns=["h2"],
    value=(15 * 2 * pi) / N,
)
gen.add_transformation(TRANSFORMATIONS.Sin, in_columns=["h2"])
gen.add_transformation(TRANSFORMATIONS.Mult, in_columns=["h2"], value=210)

# transition between states
gen.add_transformation(
    TRANSFORMATIONS.StateChange,
    in_columns=["h1", "h2"],
    state_column="states",
    states=["A", "B"],
    out_column="interruption",
)

df = gen.generate(N)
```

Listing 1: Example of code for generating a "sag" signal.

```python
N = 1000

gen = Generator()

# create base variables with intended normal distribution
gen.add_variable(VARIABLES.Normal, name="Water", mean=6.08, std=4.36)
gen.add_variable(VARIABLES.Normal, name="Acidity", mean=0.0382, std=0.06)
gen.add_variable(VARIABLES.Normal, name="BDV", mean=49.34, std=16.06)
gen.add_variable(VARIABLES.Normal, name="DF", mean=0.275, std=0.23)
gen.add_variable(VARIABLES.Normal, name="TDCG", mean=301.91, std=381.9)
gen.add_variable(VARIABLES.Normal, name="Furan", mean=0.171, std=0.407)

# perform logistic combination of variables
gen.add_transformation(
    TRANSFORMATIONS.Linear,
    in_columns=["Water", "Acidity", "BDV", "DF", "TDCG", "Furan"],
    out_column="HI",
    intercept=5.45,
    coefs=[-0.246, 28.604, -0.171, 0.112, 0.01, 0.254]
)
gen.add_transformation(TRANSFORMATIONS.Mult, in_columns=["HI"], value=-1)
gen.add_transformation(TRANSFORMATIONS.Exp, in_columns=["HI"])
gen.add_transformation(TRANSFORMATIONS.Add, in_columns=["HI"] value=1)
gen.add_transformation(TRANSFORMATIONS.Power, in_columns=["HI"], value=-1)

df = gen.generate(N)
```

Listing 2: Example of code for recreating the HI model.