Generative AI: Synthetic Data Vendor Comparison and Benchmarking Best Practices

Vincent Granville, Ph.D.
vincentg@MLTechniques.com
www.MLTechniques.com
Version 1.0, June 2023

Contents

1 Introduction

The goal of data synthetization is to produce artificial data that mimics the patterns and features present in existing, real data. Many generation methods and evaluation techniques are available, depending on purposes, the type of data, and the application field. Everyone is familiar with synthetic images in the context of computer vision, or synthetic text in applications such as GPT. Sound, graphs, shapes, mathematical functions, artwork, videos, time series, spatial phenomena – you name it – can be synthesized. In this article, I focus on tabular data, with applications in fintech, the insurance industry, supply chain, and health care, to name a few.

The word "synthetization" has its origins in drug synthesis, or possibly music. Interestingly, the creation of new molecules also benefits from data synthetization, by producing virtual compounds, whose properties (if they could be produced in the real world) are known in advance to some degree. It also involves tabular data generation, where the features replicated are various measurements related to the molecules in question.

Historically, data synthetization was invented to address the issue of missing data, that is, as a data imputation technique. It did not work as expected as missing data is usually very different from observed values. But the technique has evolved to cover many applications:

- When the data is skewed, typically in fraud detection of healthcare settings, where fraud cases are rare or when dealing with rare diseases.
- When the cost of data acquisition is high, or large samples are not available. For instance in clinical trials. This may also be required with wide data, consisting of a large number of features but few observations. Some algorithms are not good at handling such datasets.
- To reduce the impact of algorithmic biases, when the data collected leads to decisions (loan acceptance) unfavorable to minorities.
- To produce more diversified data, for instance to train a predictive algorithm. Combining the original data with a synthetic version leads to what is called *augmented data*. The goal is to reduce overfitting, and to increase the chances that the algorithm will work properly when faced with data never seen before.
- To preserve privacy, for security reasons, or when sharing sensitive data with third parties. In this case, special care is needed to preserve geographic distribution or to generate synthetic credit card numbers

that correspond to the actual distribution of bank issuers. Yet making it impossible to retrieve personal data: outliers can sometimes be matched back to real people if the synthetization is not properly done.

- In my case, I use synthetization for massive simulations, or to produce unusual data to test and benchmark algorithms and statistical models, discover their weaknesses, and improve them. Another related application is stress-testing: feeding a system or platform with a large amount of varied data, to identify breaking points. Typically, this is an engineering problem.

One original contribution in this article is the discussion of alternative methods to discuss faithfulness. In particular, I show the limitations of correlation distances and one-dimensional statistical summaries. I do not discuss privacy metrics, but I briefly describe holdout methods and utility assessment, using cross-validation techniques or post-classification, illustrated with a real case study. In addition, I mention a few performance metrics that are overlooked by other authors: time-to-train (and how to improve it), replicability, ease of use, parameter optimization, and data transformation prior to synthetization.

You can synthesize data using interpolation, agent-based modeling, adding correlated zero-mean noise to the real data, using copulas or generative adversarial networks (GANs). All these techniques are discussed in details in my book [8]. However, in this article, I focus only on copulas and GANs. These are the techniques used by the vendors compared here.

2 Vendor comparison

In this section, I compare four vendors: YData.ai, Mostly.ai, Gretel and Synthesized.io, along with the open source library SDV (synthetic data vault), and my own solutions. The focus is on tabular or transactional data, and assessing how realistic the synthetization is, compared to the real data it is supposed to mimic. The comparison involves three case studies: the insurance, diabetes and circle datasets, each with its own challenges. These datasets have ordinal, categorical, and continuous features. Most of the distributions are not Gaussian, and even bimodal in one case. The dependency structure is sometimes complex and non-linear, thus not properly summarized by correlations. The datasets were cleaned before being processed.

The context is regulated industries, where the fit between the original and synthetic data must be very good and even meet some minimum standards set up by regulators. The fact that all the algorithms are by default unable to generate observations outside the range of observations may be seen as a quality, or even a requirement. By contrast, in a different context such as testing black box systems, such a feature is considered a defect.

Nevertheless, overfitting is still a potential issue that must be addressed. Generated data that is too similar to the original may share the same drawbacks. In particular, reproducing algorithmic bias against minorities, present in the real data. Also, by transforming the data (standardization, decorrelation, non-linear scaling), it is possible to generate synthetic data with the exact same correlation matrix and same marginal empirical distributions as in the original data. Likewise, the basic method consisting of adding some amount of noise – small or large – to the original method, combined with resampling, can lead to exceptionally good results but potential overfitting.

The above techniques to produce exceptionally high quality synthetic data amount to cheating. In section 2.3, I discuss the holdout method (a portion of the real data not used for synthetization) to evaluate results in contexts where this is relevant, to reject synthetizations that are "too good to be true". However, I did not find artificial, unfair boosting in the solutions that I tested. To the contrary, in many (but not all) cases, defects show up even without using holdout, when looking at hard-to-detect patterns in the real data. My goal here is to identify vendors that do a better job at detecting unusual structures, those that are not captured by quality metrics such as correlation or Kolmogorov-Smirnov distances. When using simplistic quality metrics or on easy data, most everyone do well. But easy data is rare in the real world.

2.1 What are we comparing?

I used the default version available on each platform. Some vendors allow you to fine-tune parameters, but typically recommend a specific method depending on your dataset, with pre-set parameters. Each time, I chose the recommended method. Thus, I am not comparing the best potential synthetizations offered by each platform, but the standard version available in the free trial.

Fine-tuning hyperparameters can be very time-consuming especially if you don't know all the details of the underlying algorithms, usually kept secret for obvious reasons. It can also lead to overfitting. Then, my goal was to replicate the experience of a potential client, who expects a quick, easy, efficient solution to his problems.

Comparing processing times is difficult, as it depends on how much bandwidth the vendor offers in its free trial. Some vendors such as YData pre-process the data and create an environment around it. This step takes

time, but once done, data generation is very fast. Other vendors require more CPU time later in the process. The use of GPU can further accelerate the training. Methods purely based on copulas are the fastest, and yield good results if the evaluation metrics are Kolmogorov-Smirnov, correlation distances and similar KPIs. However, as we shall see, these metrics fail to capture complex interactions. Most vendors offer evaluation methods. But here, I used my own.

Ease of use is also an important metric. YData allows you to run Python code on its platform. This is convenient and easy, in particular since all the required libraries, with the correct versions, are pre-installed and maintained. Running the synthesizer is truly as simple as running 3 lines of Python code. By contrast, Synthesized.io offers an SDK that you run in your environment. In principle, it offers more flexibility, but the libraries conflicted with mines, and running it in an virtual environment was not as simple as one would think. The open source library SDV, in terms of ease of use, is intermediate between YData and Synthsized.io. Again, it's 3 lines of Python code, and the problem is also with library compatibility. Metadata identification is an extra step that needs special care. The other vendors, Gretel and Mostly.io, offer a pure Web API. You don't need to be a machine learning scientist, much less a MLops engineer, to use their platform.

In all but one exception, all the synthetized data stays within the range of observations. It seems that this is a built-in feature, possibly a desirable property required by the customers. It would be easy to change this, for instance by injecting noise in the output data. My own GAN is not constrained to stay within the observed range, as I need truly unusual observations and outliers in my research projects when testing classifiers or predictive algorithms. Note that copulas based on empirical quantiles can not generate outside the range; parametric copulas or model-based systems relying for instance on GMMs (Gaussian mixture models) are able to do so. See my article "Smart Grid Search for Faster Hyperparameter Tuning" [7].

Finally, unlike my home-made solutions, none of the vendors offer replicable results. Running the synthesizer twice produces two different output datasets. However, variations between successive runs are small, typically much smaller than the discrepancy between a typical synthetized dataset, and the real version. These small variations are of course a desirable property. I assume you can always save the GAN model that produced the best results, if you use the full version offered by the platform. By contrast, my GAN is a lot more volatile, probably the reason why I designed it to produce replicable results, controlled by a seed parameter. A benefit of high volatility is that you can test various seeds and keep the best result, identified by the seed in question. When using synthetizations from vendors, you can achieve similar improvements by producing a sample much larger than you need, and removing selected observations one at a time in the synthesized data, in order to increase the quality.

2.1.1 Evaluation methods

Evaluation methods depend on the goals. Here we are interested in measuring faithfulness: how well the synthetizations mimic the real data. I focus specifically on three kinds of metrics:

- The correlation distance matrix Δ. This symmetric $m \times m$ matrix is defined as $\Delta_{ij} = |R_{ij} - S_{ij}|$, where R, S are the correlation matrices respectively measured on the real and synthetic data. Here m is the number of features, including the response or outcome if there is one. The indices i, j represent two features. In particular Δ_{avg} and Δ_{max} are the average and maximum value of Δ. These values are always between 0 (best match) and 1 (worst match).
- The Kolmogorov-Smirnov distance vector K. This vector has m components, one for each feature. Each component represents the normalized distance between two empirical distributions, for the corresponding feature: one attached to the synthetic data, and the other one to the real data. In particular K_{avg} and K_{max} are the average and maximum value of K. These values are always between 0 (best match) and 1 (worst match).
- Additional metrics capturing non-linear inter-dependencies among features, and how well these non-linear patterns are reproduced in the synthetic data. I use visualizations to show the match (or lack of) between real and synthetic data. These metrics are important, as correlations alone focus on linear dependencies only, and Komogorov-Smirnov are one-dimensional summaries and do not take into account the feature dependencies.

The Hellinger distance is a popular metric, similar to Kolmogorov-Smirnov (KS) but based on empirical probability functions rather than empirical distributions. Unlike KS, it generalizes to multivariate distributions and even features with categorical data. However, it relies on feature binning. The number of bins grows exponentially with the number of features, resulting in bins with very few observations. This problem can be addressed by bin aggregation and focusing on large bins only. However, most practitioners still use Hellinger separately for each feature rather than jointly across all features. In this case, KS is a more stable metric. Besides Hellinger, other multivariate metrics are available, such as the Hausdorff distance to compare two datasets. See the chapter on synthesizing and comparing shapes, in my book on generative AI [8].

For categorical features, correlations can be computed using Cramer's V statistic. The range is between 0 and 1, instead of -1 to $+1$ for standard correlation. But the resulting correlation distance is also between 0 (best fit) and 1 (worst fit). Another way to deal with a categorical feature with k potential values is to replace it with $k-1$ binary features, called dummy variables.

Other evaluation techniques include cross-validation and the holdout method, see section 2.3. Some practitioners blend the real and synthetic data together, and run an unsupervised clustering algorithm, with the number of clusters set to 2. If real and synthetic observations are well separated, your synthetization is bad. Conversely, if each cluster contains about the same proportion of synthetic and real data, you did a good job! This idea is key to training GAN models. Classification scores in this context are known as utility metrics.

Finally, to benchmark synthetic data vendors, one can generate real data with patterns not detectable using correlation distances or KS-like metrics. Indeed, great scores are accomplished by copulas alone if you only use these two evaluation methods. I created such a dataset, referred to as "circle8d" in this article. All cross-correlations are zeros, and the data points lie on a subspace in the 9-dimensional feature space, by design. In this case, you need other metrics to see how GANs significantly outperform copulas. I discuss the details later in this article.

Replicability and the ability to generate synthetic data outside the observation range may be important or not, depending on the user. None of the vendors offer these options in their free trial option, although my GAN models do. However, all vendors and even SDV focus on fast computations. In the case of Gretel and SDV (actually, SDV Lite which uses fast gradient descent), this could explain the lower quality. Vendors also offer easy solutions with little or no coding, though SDV and especially Synthesize.io required more efforts on my part due to force-installing older versions of Python libraries and customer support not responding.

2.1.2 Note on parametric models

Some synthetizations use parametric distributions to represent the real data. In this case, the synthetization process simply consists of simulations, generating a sample from the same parametric distribution, with parameter values estimated on the real data. A typical example is multivariate or univariate GMMs (Gaussian mixture models), where the parameters are estimated via the EM algorithm. One version of my copula method includes a 2-parameter zeta-geometric distribution for the number of children, in the insurance dataset discussed in this article, instead of using empirical quantiles. For details, see [7]. The parameters are estimated via smart grid search. It should be noted that Gaussian copulas (used in my examples) or any type of copula work with non-Gaussian or discrete features, or a combination of both, regardless of the underlying distributions in the real data, in the same way that Gaussian latent variables work regardless of the actual distribution, in GAN models. The choice of non-Gaussian is favored when distribution tails are very thick and may lead to under-sampling extremes.

2.1.3 Enhanced synthetizations

I only tested the default settings in the free trial black-boxes offered by the vendors. The datasets were small, with a mix of categorical, discrete and continuous features (up to 9 features). Most were non-Gaussian; some were multimodal features with a complex inter-dependencies structure. Two of the real datasets are well-known and discussed on Kaggle, the third one is artificially generated to test the limitations of the platforms, and assess the superiority of GANs over copulas.

There is no doubt that each vendor offers more sophisticated solutions to paid customers, and continuously works on improving its algorithms. Yet the free trial version (automated black-box) is what many potential customers are going to see first when deciding which vendor to choose. Even then, there are different ways to enhance the output synthetic data produced by these platforms. Possible front-end enhancements include:

- Transform your data before running any synthesizer, to remove cross-correlations and normalize the feature distributions. Note that some vendors may do that already. The YData system in particular spends some time preparing the data, which may involve data transformations. This could explain why the correlation structure is so well rendered.
- Do multiple runs. Since each run produces a different synthetization, choose the one with the best fit. Or produce a large number of observations, and remove the ones that negatively impact the quality.
- The insurance dataset has 3 categorical features: gender, smoking status, and regions (Northeast, Southeast, Southwest, Northwest). Split your real data into multiple subsets, one per bin. A bin is a combination of these 3 features. Then run a separate synthetization on each bin.
- Perform feature clustering before synthesizing. This is an alternative to PCA. It allows you to identify features that are barely correlated to the other ones. Synthesize these features separately and independently. Use multivariate synthetization only for feature clusters with strong enough inter-correlations. This may

be useful when you have a large number of features. See how it works in one of my recent articles [5], applied to the diabetes dataset investigated in this paper.

- Train the synthesizer on a fraction of the dataset, by randomly deleting observations from the real data. You may save the deleted observations for cross-validation or holdout as described in section 2.3. The goal here is to accelerate the time-consuming learning process in GAN models used in synthetization. Yet it has almost no impact on the quality of the output. See how it works in my article on data thinning [6], applied to the insurance dataset investigated in this paper. Another similar approach is data distillation, discussed in [14].

Further optimization can be achieved by customizing the source code, when possible. For instance, in my home-made GAN synthesizer, you can use a specific loss function L that corresponds to your goal. In particular, you may want to minimize the customized loss function $L = \Delta_{\text{avg}} + K_{\text{avg}}$, with Δ_{avg} and K_{avg} defined in section 2.1.1. This back-end modification requires a good knowledge of the Keras library. An easier fix is to do the same on the front-end: every 50 epochs, you evaluate L on the synthetic data produced at the current epoch, and stop as soon as L becomes stable and good enough. Instead of running 10,000 epochs, you may stop the training much earlier thus saving a significant amount of time.

Finally, if you can fine-tune hyperparameters, you can do so automatically using an efficient method, such as smart grid search described in one of my articles [7]. Also, using a light version of the gradient descent such as LightGBM instead of the traditional Adam, helps accelerate the training, but the quality may suffer.

2.2 Case studies

In this section, I discuss the methodology and results. Case studies include insurance and healthcare, as well as an artificial and challenging dataset. Vendors include YData.ai, Synthesized.io, Gretel, Mostly.ai, as well as open source SDV Lite (the synthetic data vault library), and my home-made solutions (copulas and GANs). In each case, I tested the free trial default version, using the sample code posted on the respective platforms (YData, Synthesized, SDV) or the Web API black-box version when no code is available (Gretel, Mostly.ai). I produced two synthetizations for each vendor, to assess replicability and the variance between runs. Except for my home-made solutions, two runs produce two different synthetizations, typically very similar. For a same vendor, there is considerably more variations between the real and synthetic data than between two synthetizations. Thus, I used the first synthetization obtained for comparison purposes.

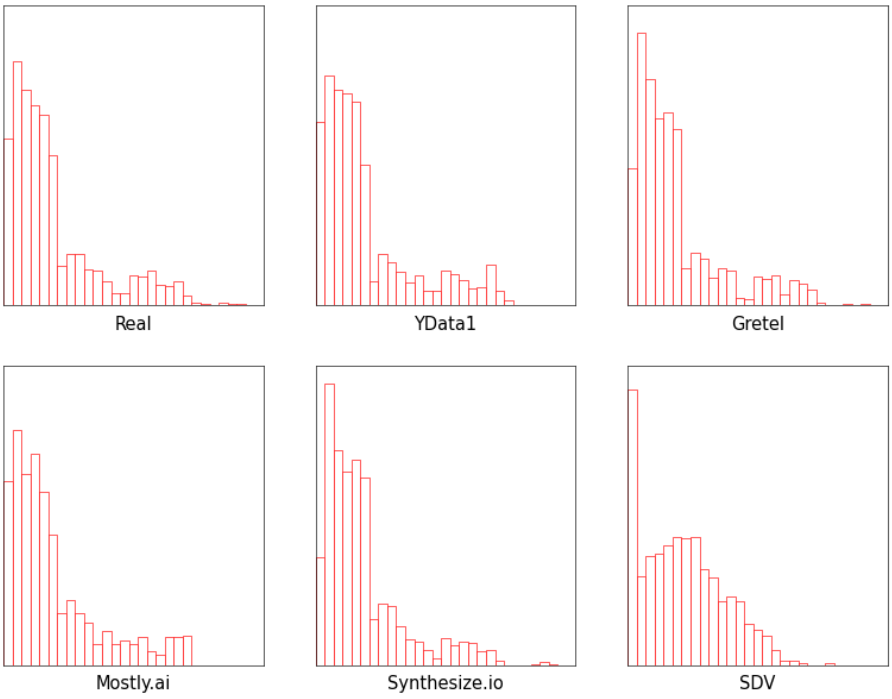

Figure 1: Insurance data, charges distribution, real (top left) vs synthetic

2.2.1 Datasets

The number of observations range from 400 to 1400. The diabetes data sets consists of 9 features: age, cancer status (the binary outcome), number of children, and 6 continuous measurements such as body mass index and

glucose level. Many observations have missing values and were removed. One could treat them as a separate dataset, for synthetization purposes. The circle dataset – artificially created – also has 9 features, all continuous. It exhibits strong non-linear dependencies. For instance, the two first coordinates lie in two concentric circles and are thus uncorrelated, yet highly dependent. Finally, the insurance dataset has 6 features, including continuous, ordinal, and categorical ones.

All datasets are on my GitHub repository, here. Figures 1 shows the bimodal, non-Gaussian distribution for "charges per policyholder", incurred by the insurance company in the insurance dataset. Each observation corresponds to a customer. The top left plot corresponds to the real data, while the other plots show the same distribution in the synthetic data, for each vendor. Firgure 2 shows the distribution for the first feature, in the circle dataset. As we shall see, good performance based on one-dimensional metrics does not guarantee that the synthetization is good. This is particularly true for Gretel. Yet, SDV already shows its poor performance. Mostly.ai (for the insurance dataset) exhibits unusual concentration of values within a subset of the observation range, missing extremes, a pattern frequently observed with this vendor. It looks as if the synthesized distribution is truncated.

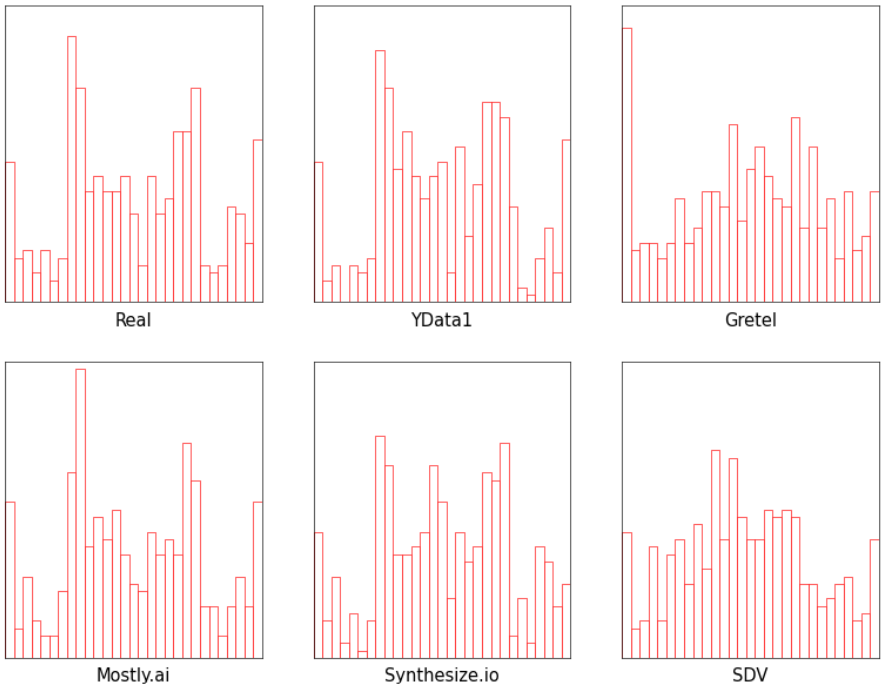

Figure 2: Circle data, first coordinate distribution, real (top left) vs synthetic

2.2.2 Python code and spreadsheets

The real data is on my GitHub repository, here. Look for the files `insurance.csv`, `circle8d.csv`, and `diabetes_clean.csv`. The three corresponding spreadsheets are in the same folder. They contain summary statistics, the synthetised data for each vendor, and when applicable, the Python code to run the synthetizations. The sample code below is available as `insurance_compare.py`, `circle8d_compare.py` and `diabetes_compare.py` in the same folder. It produces the visualizations shown in this article, and computes the evalution metrics $\Delta_{\text{avg}}, \Delta_{\text{avg}}, K_{\max}, K_{\max}$ described in section 2.1.1 and presented in Table 1.

For convenience, I also put the real data and synthetizations (the input files) in three separated documents: `insurance_compare.csv`, `circle8d_compare.csv` and `diabetes_compare.csv`. The first field specifies the type of data: real, or synthesized. In the latter, it also indicates the vendor.

If you use the holdout method described in section 2.3, the real data is split into two sets, labeled as "validate" and "train". In this case, the synthesizer is trained on the "train" data, and the "validate" set plays the role of the real data. The `insurance_compare_holdout.csv` input file, and the related Python program `insurance_compare_holdout.py` in the same folder, illustrate how it works.

Finally, I want to acknowledge Mulyadi Kurniawan, a former colleague at Visa, for his invaluable help to run the Synthesized.io SDK in a virtual environment and produce the corresponding synthetizations.

```
import pandas as pd
import numpy as np
import scipy
```

```
from scipy.stats import ks_2samp
from statsmodels.distributions.empirical_distribution import ECDF

dataset = 'insurance_compare.csv'
url = "https://raw.githubusercontent.com/VincentGranville/Main/main/" + dataset
df = pd.read_csv(url)
# df = pd.read_csv(dataset)
if dataset == 'insurance_compare.csv':
   df = df.drop('region', axis=1)
   df = df.dropna(axis='columns')
print(df.head())

data_real = df.loc[df['Data'] == 'Real']
data_real = data_real.drop('Data', axis=1)
data_real = data_real.to_numpy()
print(data_real)

r_corr = np.corrcoef(data_real.T) # need to transpose the data to make sense
print(r_corr)

ltests = df.Data.unique().tolist()
popped_item = ltests.pop(0) # remove real data from the tests
print(ltests)

#--- main loop

for test in ltests:

   data_test = df.loc[df['Data'] == test]
   data_test = data_test.drop('Data', axis=1)
   data_test = data_test.to_numpy()
   t_corr = np.corrcoef(data_test.T)
   delta = np.abs(t_corr - r_corr)
   dim = delta.shape[0] # number of features

   ks = np.zeros(dim)
   out_of_range = 0
   for idx in range(dim):
      dr = data_real[:,idx]
      dt = data_test[:,idx]
      stats = ks_2samp(dr, dt)
      ks[idx] = stats.statistic
      if np.min(dt) < np.min(dr) or np.max(dt) > np.max(dr):
         out_of_range = 1
   str = "%20s %14s %8.6f %8.6f %8.6f %8.6f %1d" % (dataset, test, np.mean(delta),
          np.max(delta), np.mean(ks), np.max(ks), out_of_range)
   print(str)

#--- visualizing results

def vg_scatter(df, test, counter):

   # customized plots, insurance data
   # one of 6 plots, subplot position based on counter

   data_plot = df.loc[df['Data'] == test]
   x = data_plot[['age']].to_numpy()
   y = data_plot[['charges']].to_numpy()
   plt.subplot(2, 3, counter)
   plt.scatter(x, y, s = 0.1, c ="blue")
   plt.xlabel(test, fontsize = 7)
   plt.xticks([])
   plt.yticks([])
   plt.ylim(0,70000)
   plt.xlim(18,64)
   return()
```

```
def vg_histo(df, test, counter):

    # customized plots, insurance data
    # one of 6 plots, subplot position based on counter

    data_plot = df.loc[df['Data'] == test]
    y = data_plot[['charges']].to_numpy()
    plt.subplot(2, 3, counter)
    binBoundaries = np.linspace(0, 70000, 30)
    plt.hist(y, bins=binBoundaries, color='white', align='mid',edgecolor='red',
            linewidth = 0.3)
    plt.xlabel(test, fontsize = 7)
    plt.xticks([])
    plt.yticks([])
    plt.xlim(0,70000)
    plt.ylim(0, 250)
    return()

import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['axes.linewidth'] = 0.3

vg_scatter(df, 'Real', 1)
vg_scatter(df, 'YData1', 2)
vg_scatter(df, 'Gretel', 3)
vg_scatter(df, 'Mostly.ai', 4)
vg_scatter(df, 'Synthesize.io', 5)
vg_scatter(df, 'SDV', 6)
plt.show()

vg_histo(df, 'Real', 1)
vg_histo(df, 'YData1', 2)
vg_histo(df, 'Gretel', 3)
vg_histo(df, 'Mostly.ai', 4)
vg_histo(df, 'Synthesize.io', 5)
vg_histo(df, 'SDV', 6)
plt.show()
```

2.2.3 Results

On average, the best results come from YData, followed by Synthesized and Mostly.ai. Performance metrics are summarized in Table 1, broken down per vendor and dataset. The rankings, directly derived from that table, are shown in Table 2.

	Insurance data				Circle data				Diabetes data			
Vendor	Δ_{avg}	Δ_{max}	K_{avg}	K_{max}	Δ_{avg}	Δ_{max}	K_{avg}	K_{max}	Δ_{avg}	Δ_{max}	K_{avg}	K_{max}
Gretel	0.021	0.093	0.040	0.086	0.134	0.887	0.233	0.444	0.091	0.689	0.209	0.412
Mostly.ai	0.017	0.058	0.023	0.042	0.050	0.139	0.038	0.063	0.123	0.499	0.048	0.125
SDV	0.034	0.351	0.093	0.213	0.035	0.104	0.078	0.109	0.030	0.117	0.092	0.152
Synthesized	0.026	0.071	0.026	0.040	0.055	0.163	0.055	0.060	0.032	0.115	0.051	0.080
YData	0.016	0.052	0.020	0.030	0.048	0.130	0.048	0.055	0.054	0.174	0.050	0.084

Table 1: Real vs created: correlation distances Δ and Kolmogorov-Smirnov K

Figures 1 and 2 show one-dimensional statistical summaries. On the other hand, the scatterplots in Figures 3 and 4 show two dimensional interactions, and provide deeper insights regarding the performance of each vendor, in terms of the ability to find and render the non-linear patterns present in the real data. There are many more histograms and scatterplots than those displayed in this article. My selection is limited to those that exhibit the most striking differences between vendors, and linked to challenges present in the joint distribution attached to

the real data.

For instance, "charges" in the insurance dataset appears to be segmented, as if there were three distinct customer segments, including a rather small one responsible for heavy medical expenses. Combined with "age", it produces the three bands in each scatterplot in Figure 3. Clearly, YData is the winner here. Gretel seems to be doing quite well too based on the visualization. With YData, it is the only vendor to properly identify the three bands. However, this is only one of the three scatterplots for the insurance dataset, the other ones being bmi vs charges and age vs bmi. Also, the Gretel scatterplot looks more sparse than in the real data, despite the fact that both have the same number of observations. This is caused by Gretel synthetic observations not being spread well enough. The overall variance may be well rendered because it is mostly explained by the three bands, but within each band, the variance is not well rendered. This a reason why not to rely on just one metric to assess the overall quality.

For the two first coordinates in the circle dataset (Figure 3), YData is the only vendor to detect and replicate the fact that the observations are distributed on two concentric circles. Mostly.ai and Synthesized do a good job too, but with synthetizations exhibiting some diffusion not present in the real data. YData exhibits a drift on the right-hand side of the scatterplot. Same with Mostly.ai, but less pronounced. Despite the poor visual performance, SDV actually ranks first (among the 5 vendors) for the Δ metrics. However, all vendors in this example detect the fact that there is zero correlation, and all but Gretel have a very good Δ. This is another reason to not just rely on correlations to measure inter-dependencies.

	Insurance data				Circle data				Diabetes data				Rank
Vendor	Δ_{avg}	Δ_{max}	K_{avg}	K_{max}	Δ_{avg}	Δ_{max}	K_{avg}	K_{max}	Δ_{avg}	Δ_{max}	K_{avg}	K_{max}	
Gretel	3	4	4	4	5	5	5	5	4	5	5	5	4.50
Mostly.ai	2	2	2	3	3	3	1	3	5	4	1	3	2.67
SDV	5	5	5	5	1	1	4	4	1	2	4	4	3.42
Synthesized	4	3	3	2	4	4	3	2	2	1	3	1	2.67
YData	1	1	1	1	2	2	2	1	3	3	2	2	1.75

Table 2: Vendor rank for each metric in Table 1; rightmost column is average rank

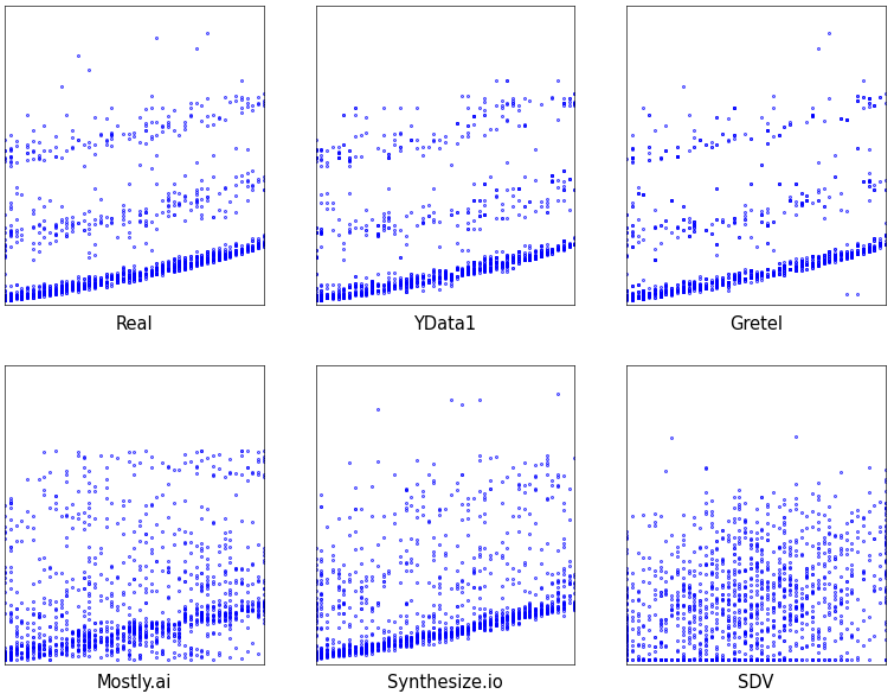

Figure 3: Insurance data scatterplot, age (X-axis) versus charges (Y-axis)

I am not sure how to explain the low performance of Gretel. Maybe it uses pure GAN, and GAN alone can be boosted using the various strategies discussed earlier. Vendors such as YData, Mostly.ai, and Synthesized most likely use such boosted GANs. For instance, my pure GAN does not perform well either. It is the

worst performer on the diabetes dataset, worse than Gretel. But my home-made GAN does moderately well on the circle data. A possible explanation is that Gretel does not use enough epochs (which is easy to fix): mine uses 10,000, and Gretel uses 600 only. Clearly, on the circle data, Gretel's GAN hasn't reached a low equilibrium if you look at Figure 5. By contrast, mine has stabilized, but after a lot more than 600 epochs: see Figure 6. However, the number of epochs could be low yet yields good results, as there are many other important parameters influencing the output, such as batch size.

Another possibility is that Gretel has a hard time doing well on small datasets, which are more difficult to synthesize than big ones. Or maybe the goal is not to optimize faithfulness. After all, personally, I am interested in the least faithful synthetizations because my goal is to test the limits of various algorithms by feeding them with unusual data. But in the context of this article, faithfulness is critical. Finally, SDV is also underperforming, possibly because I used the Lite version, or because they have incentives to sell the commercial product rather than people using the open source implementation.

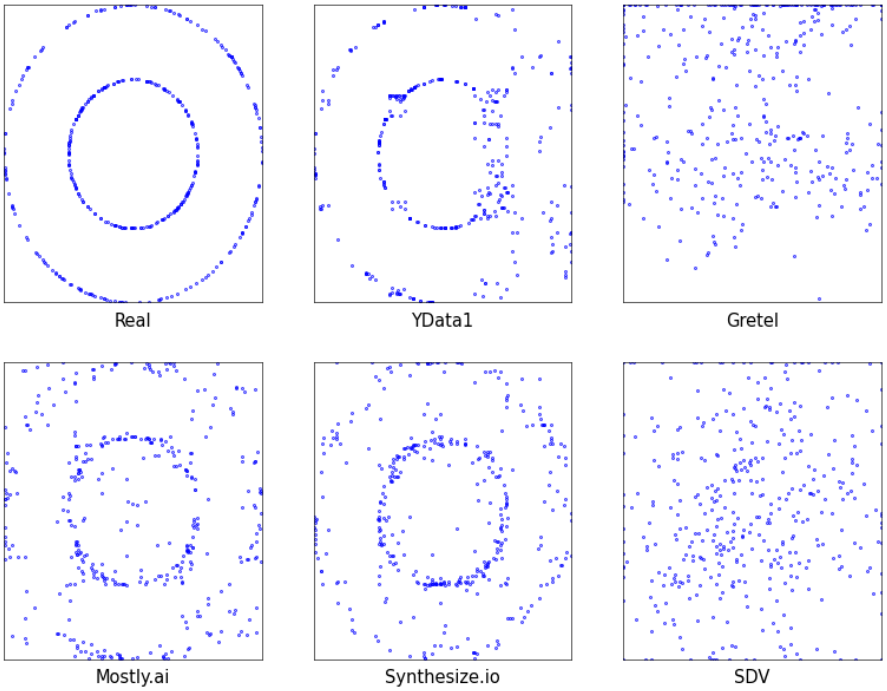

Figure 4: Circle data scatterplot, first two coordinates

2.3 Holdout and cross-validation

One of the most efficient ways to assess the quality of synthetic data is to use it to augment a training set, and see how the augmented data improves the learning capacity of your algorithm. For instance, if you run a predictive algorithm, adding artificial data to train it may lead to more robust predictions. You would expect the synthetic data to have unusual observations not present in your real data. This helps when facing new data different from what is in your original training set. The performance should always be evaluated in a cross-validation setting: using a portion of your original training set augmented with synthetic data to train your model, and using the other part of your training set for evaluation purposes.

For best results, you want your synthetic data to not perfectly mimic the data in the training set, but instead, to contain out-of-range and unusual observations. In short, in this peculiar context, you don't want your synthetic data to be as faithful as possible, quite the contrary. What you are doing by injecting synthetic data in the training set (and/or validation set) is a sensitivity analysis. The more varied your synthetic data, the better your predictive algorithm if it still passes your quality tests (R-squared on so on measured on the validation set). The goal is not to increase the accuracy of your predictions (say, by increasing R-squared) but instead to make them less sensitive to variations in the data. In short, reducing overfitting. The price to pay is actually less accurate predictions, but the gain is increased robustness and less sensitivity to noise and other factors.

What I just described applies to scientists developing new AI algorithms. But in our context, the goal is opposite: you want to maximize faithfulness. As discussed earlier, it is easy to create synthetic data that optimize the standard metrics of quality assessment. The following techniques achieve this goal:

- Adding very little noise to the real data.

- Transforming the synthetic data via a matrix transformation, so that the correlation matrices corresponding to the real and synthetic data, are identical post-transformation. In that case, the metrics Δ_{avg} and Δ_{max} in Table 1 would be zero, the best possible value.
- Apply a non-linear transform (scaling or recalibration) to each feature separately, after synthetization, so that the corresponding empirical distributions (real versus synthetic, post-transform) are identical. In that case, the metrics K_{avg} and K_{max} in Table 1 would be zero, the best possible value.
- Produce a larger synthetic dataset than needed. Then remove observations one at a time until your synthetic dataset has the desired size. At each step, choose the generated observation to delete so that after deletion, your quality metrics improve the most or on average.

To make sure that the synthetic data from a vendor is not artificially spiked (the high quality being an artifact of some of the above manipulations), you can run the same cross-validation techniques. In this case, use 50% of your real data to produce the synthetic data, and compare the synthetization with the remaining 50% of your real data. I did this test for YData, as the quality was superior to that from other vendors. The results are in Table 3.

Data	Type	Δ_{avg}	Δ_{max}	K_{avg}	K_{max}
Training	Real	0.0314	0.0687	0.0361	0.0732
Validation	Real	0.0000	0.0000	0.0000	0.0000
YData1	Synthetic	0.0394	0.1081	0.0433	0.0792
YData2	Synthetic	0.0241	0.0822	0.0386	0.0807

Table 3: Insurance data, distances to validation set

The 50% used for the synthetization is called the training set, the other 50% is called the validation set or holdout. You can't compare the values in Table 1 with those in Table 3, because the datasets are now reduced by 50%. However, the distances between the two parts of the real data (training and validation sets) shown in the "Training" row, are very similar to the distances between any of the two synthetizations produced by the training set, and the validation set, as shown in rows "YData1" and "YData2". Note that the "Validation" row has zero values, as it compares the validation set with itself.

This holdout technique is particularly useful if your real data is a time series: then the training set consists of older data, while the validation set contains the most recent data. Or when your real data is a blend from (say) 20 different sources: then you put 10 sources in your training set, and the other ones in your validation set. In the case studies explored in this article, the real data is quite homogeneous, thus the need for cross-validation is less critical.

2.4 Comparison with my synthetization algorithms

I did not include the results of my home-made synthesizers (GAN and copula) in my analysis, for two reasons. First, my synthetization techniques are designed for research or teaching purposes, and faithfulness is not a metric that matters in my own personal context: testing and improving AI/ML algorithms such as deep neural networks, classifiers, and predictive systems. To the contrary, I need to synthesize unusual observations, similar to outliers. While my copula method does a very good job on the three datasets, it is because metrics such as Δ and K tend to favor this technique. However, if you look at Figure 5, it is clear that YData, Mostly.ai and Synthesized do better. These vendors also outperform my own GAN technique.

Method	Δ_{avg}	Δ_{max}	K_{avg}	K_{max}
Copula	0.0300	0.0642	0.0389	0.0575
GAN	0.0167	0.0687	0.0678	0.0875

Table 4: Circle data, home-made synthesizers

Then, I used a separate copula for each group in the insurance dataset. This gives me an unfair advantage and makes comparisons meaningless. However, if you are curious about how I score, see Table 4 for the circle dataset.

Finally, I included the history of my home-made GAN (evolution of the loss function over successive epochs) in Figure 7. First, it shows why I needed 10,000 epochs. This makes it slow to train compared to the vendors, although the number of epochs is not the only factor determining speed. Then it may explain why Gretel, with

only 600 epochs, achieves a much lower performance, albeit a lot faster. Indeed, Figure 6 suggests that Gretel's ACTGAN was stopped prematurely. Also note that my GAN is very sensitive to the initial conditions, with two different seeds producing fairly different synthetizations. From Figure 7, it is clear that seed 103 produces a much better synthetization than seed 102. To the contrary, vendors offer more stable implementations.

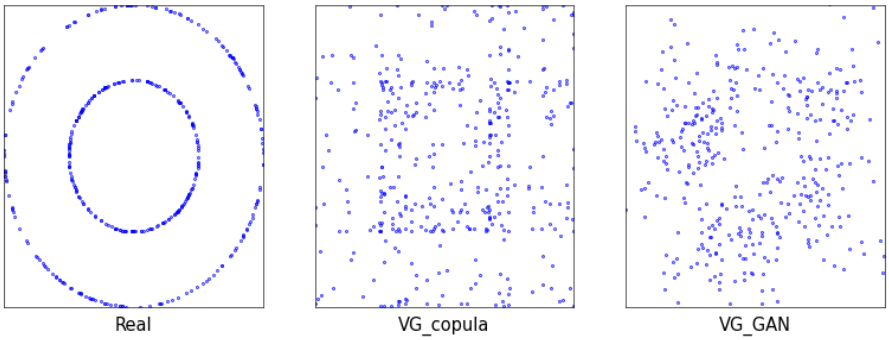

Figure 5: Circle data scatterplot, first two coordinates

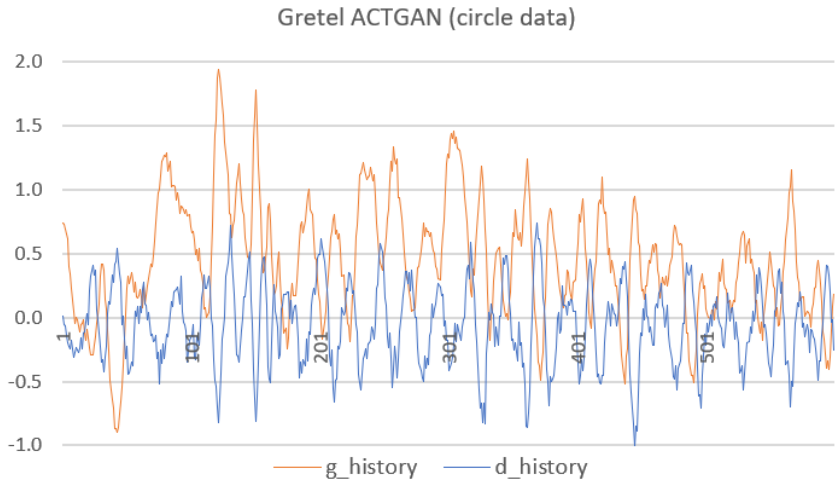

Figure 6: Training history, Gretel's ACTGAN (600 epochs)

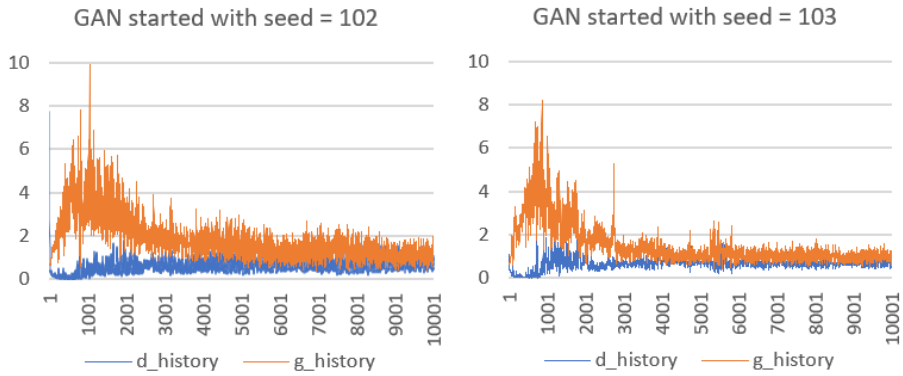

Figure 7: Training history, home-made GAN (10,000 epochs)

2.5 References

Additional reading on the subject includes the book "Synthetic Data for Deep Learning" [11] focusing on computer vision, "Practical Synthetic Data Generation" [3] focusing on heath care and copulas, chapter 10 in my book "Synthetic Data and Generative AI" [8] for tabular data (GANs and copulas). See also "Pros and Cons of GAN Evaluation Measures" [2], "Survey on Synthetic Data Generation, Evaluation Methods and GANs" [4], and "Are GANs Created Equal? A Large-Scale Study" by Google Brain [10]. Also of interest, Lei Xu's master thesis (MIT, 2017) available here and the related article on ArXiv [13], as well as this article about CTGAN.

For the Keras models used in my GAN implementations, see here. Implementation of GAN in Python are discussed in [9] and in my book [8]. Some authors use a utility metric to measure the quality of the fit between synthetic data and the real data that it represents, see [12].

Specialized references dealing with some aspects of the methodology include the article by Alaa *at al.* on faithfulness evaluation [1], the article by Yu *at al.* on data distillation [14] (a technique to reduce the training set while preserving its core structure) and my articles about feature clustering, data thinning (a much faster version of data distillation) and smart search grid for hyperparameter optimization. My articles are available in the resources section on MLtechniques.com, here.

3 Conclusions

The context is the evaluation of vendor solutions for tabular data synthetization. I focused on the faithfulness metric: the ability to replicate patterns found in the real data, without overfitting. The holdout method helps verify if vendors use artifacts to artificially boost performance, in a way that would qualify as cheating. No such manipulations were found. In each case, I tested the default free-trial version available on each platform.

YData does a particularly good job at capturing and reproducing special patterns that traditional evaluation metrics may miss. Such structures are bound to be found in datasets with many features. It also offers the easiest platform for a machine learning scientist, more flexible than a pure Web API such as Gretel or Mostly.ai, yet less complicated than SDV or Synthesized. The latter offers a Python SDK with authentication, and requires a virtual environment to avoid problems with conflicting libraries. YData also came with Python code; I run it on its platform (Fabric) and it went very smoothly. Of course, vendors offer different environments; I only tested the solutions that were easiest to implement on my end.

Based on this analysis, my preference clearly goes to YData. My second choice would be Mostly.ai or Synthesized, though I would like to understand the reason for Gretel's consistent under-performance.

References

[1] Ahmed Alaa et al. How faithful is your synthetic data? sample-level metrics for evaluating and auditing generative models. *Proceedings of the 39 th International Conference on Machine Learning*, pages 1–17, 2023. [Link]. 13

[2] Ali Borji. Pros and cons of GAN evaluation measures: New developments. *Preprint*, pages 1–35, 2021. arXiv:2103.09396 [Link]. 12

[3] Khaled Emam, Lucy Mosquera, and Richard Hoptroff. *Practical Synthetic Data Generation*. O'Reilly, 2020. 12

[4] Alvaro Figueira and Bruno Vaz. Survey on synthetic data generation, evaluation methods and GANs. *New Insights in Machine Learning and Deep Neural Networks*, 2022. MDPI [Link]. 12

[5] Vincent Granville. Feature clustering: A simple solution to many machine learning problems. *Preprint*, pages 1–6, 2023. MLTechniques.com [Link]. 5

[6] Vincent Granville. Massively speed-up your learning algorithm, with stochastic thinning. *Preprint*, pages 1–13, 2023. MLTechniques.com [Link]. 5

[7] Vincent Granville. Smart grid search for faster hyperparameter tuning. *Preprint*, pages 1–8, 2023. MLTechniques.com [Link]. 3, 4, 5

[8] Vincent Granville. *Synthetic Data and Generative AI*. Elsevier, first edition, 2024. [Link]. 2, 3, 12, 13

[9] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*. O'Reilly, third edition, 2023. 13

[10] Mario Lucic et al. Are GANs created equal? a large-scale study. *Proc. NeurIPS Conference*, pages 1–10, 2018. [Link]. 12

[11] Sergey I. Nikolenko. *Synthetic Data for Deep Learning*. Springer, 2021. 12

[12] Joshua Snoke et al. General and specific utility measures for synthetic data. *Journal of the Royal Statistical Society Series A*, 181:663–688, 2018. arXiv:1604.06651 [Link]. 13

[13] Lei Xu and Kalyan Veeramachaneni. Synthesizing tabular data using generative adversarial networks. *Preprint*, pages 1–12, 2018. arXiv:1811.11264 [Link]. 12

[14] Ruonan Yu, Songhua Liu, and Xinchao Wang. Dataset distillation: A comprehensive review. *Preprint*, pages 1–23, 2022. Submitted to IEEE PAMI [Link]. 5, 13