Using maximal information auxiliary variables to improve synthetic data generation based on TabPFN foundation models: preliminary results

Elias Chaibub Neto
Sage Bionetworks
Seattle, WA 98121
elias.chaibub.neto@sagebase.org

Abstract

Synthetic data generation for tabular datasets is shifting toward the use of large, general-purpose foundation models. TabPFN, a state-of-the-art example, uses in-context learning to generate probabilistic predictions conditioned on observed examples in a single forward pass. However, when variables are only weakly associated with others, the model's ability to generate realistic synthetic data deteriorates, as the context examples provide little predictive signal. To address this, we introduce the maximal information auxiliary variable (MIAV) strategy, which increases context information with auxiliary variables constructed by rank-matching random noise variables to real data. We establish theoretical properties of the approach which explain its good performance for weakly associated variables. Additional practical advantages of the MIAV approach include improved computational efficiency and invariance to variable order during the synthetic data generation process. Empirical evaluations, on simulated and real datasets, illustrate how the MIAV strategy improves data generation when compared to direct application of TabPFN.

1 Introduction

Synthetic data generation is undergoing a paradigm shift, driven by advances in large-scale/general-purpose machine learning models. Traditionally, generating realistic synthetic tabular datasets has relied on bespoke statistical models and machine learning algorithms trained and tuned in specific datasets (including, for example, (1; 8; 16; 19; 25; 29; 30; 32; 33; 35; 37)). These approaches often require substantial domain expertise, suffer from limited scalability, and produce poor out-of-the-distribution predictions and poor transfer of knowledge among distinct datasets. Tabular foundation models (14; 24; 2; 18; 12; 22; 23; 34; 36; 15; 27; 13) offer a new path forward. By learning broad, transferable representations of tabular data, they can be adapted to generate synthetic datasets across domains with minimal additional training.

In particular, TabPFN (15) represents a state-of-the-art tabular foundation model, trained on millions of diverse synthetic datasets covering a wide range of feature types, noise structures, and functional relationships. This diversity allows it to leverage a broad, transferable prior over tabular data distributions. TabPFN enjoys a solid theoretical foundation as it can be viewed as approximating a Bayesian prediction based on the broad prior defined by the synthetic datasets used in its training. TabPFN relies on in-context learning (3) (ICL) for generating probabilistic predictions. At inference time, the pre-trained foundation model employs training features, $\mathbf{X}^{tr}$, and training targets, $\mathbf{y}^{tr}$, as the "context" data, whereas the test set features, $\mathbf{X}^{ts}$, play the role of the "query". The output of the query is a sample/prediction $\hat{\mathbf{y}}^{ts}$ from the posterior predictive distribution of $\mathbf{y}^{ts}$, $P(\boldsymbol{y}^{ts} \mid \boldsymbol{X}^{ts}, \boldsymbol{X}^{tr}, \boldsymbol{y}^{tr})$, generated by a single forward pass over the model's neural network. TabPFN has been shown to

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Structured Probabilistic Inference & Generative Modeling: Probabilistic Inference in the Era of Large Foundation Models.

achieve state-of-the-art performance in classification and regression tasks (15) in small datasets and, due to its generative nature, it can also be directly used to perform synthetic data generation.

Despite all this potential, the direct application of TabPFN to synthetic data generation has an important limitation: it performs poorly for variables that are weakly associated with other variables in the dataset. This limitation is not surprising. When the target data $\mathbf{y}$ is uncorrelated to the feature data $\mathbf{X}$, the "context" examples $\{\mathbf{X}^{tr}, \mathbf{y}^{tr}\}$ cannot "teach" the model how to predict $\mathbf{y}$ from the values of $\mathbf{X}$, and when queried with $\mathbf{X}^{ts}$ the model (at least in its current version) is unable to approximate the distribution of $\mathbf{y}^{ts}$. While this is a lesser problem for supervised learning tasks, this can be a very important caveat when the goal is synthetic data generation (see Appendix A for detailed discussion of this point). While, in principle, this issue could be potentially alleviated by carefully designing an exemplar-based declarative programming strategy capable of improving model behavior in this setting, this would require fine tuning or even the full training of new foundation models.

In this paper, we tackle this problem and describe how to generate high quality synthetic datasets using the current TabPFN model. Our procedure relies on the use of maximal information auxiliary variables (MIAV) for in-context learning. We describe how to construct these variables using simple rank-matching of random noise variables to the real data and prove two key theoretical properties of these variables: (i) that conditional on it's MIAV, a variable X_j is independent from all other variables; and (ii) the MIAV of variable X_j contains maximal information about X_j in a non-parametric information theoretic sense. (See Theorem 1.) Furthermore, we also show that the MIAV synthetic data generation strategy corresponds to the correct factorization of the posterior predictive distribution obtained by conditioning on the original data and on the MIAVs.

In addition to these nice theoretical properties and the ability to generate high quality synthetic data under weak association settings, our proposed strategy has a few bonus practical advantages. First, contrary to the direct synthetic data generation proposed in (15), which is impacted by the variable order used during data generation (and thus requires aggregating results across multiple runs based on permutations of the variable's order) our approach is invariant with respect to variable order. Second, as far as computational efficiency goes, for a fixed sample size, the most impactful aspect of compute time of TabPFN is the number of variables used as context (TabPFN's complexity scales quadratically with the number of features). Because our approach employs a single feature per variable when generating the synthetic data, it achieves the best possible computational efficiency. (Not to mention the fact that it does not require aggregation across multiple runs with different variable orders.)

To illustrate the issues around the direct use of TabPFN for synthetic data generation under weak association settings, we describe a couple of direct implementations and compare their performance against the MIAV approach in simulated data experiments (where we are able to control the strength of the statistical associations among the data variables), as well as, on real-world datasets.

We conduct our evaluations in the setting of privacy-preserving data sharing (28), where the objective is to produce synthetic copies of real datasets that retain their statistical properties while simultaneously mitigating privacy risks. Accordingly, we assess the performance of the TabPFN-based synthetic data generation strategies using both data fidelity and privacy metrics.

2 Notation and previous work

Throughout the text, random variables are represented in italics, and vectors of random variables are shown in boldface, e.g., $\boldsymbol{X} = (X_1, \ldots, X_p)^t$. Data matrices and data vectors are represented in uppercase and lower case boldface, respectively. (E.g., if $\mathbf{X}$ is an $n \times p$ matrix, than the jth column of $\mathbf{X}$ is represented by $\mathbf{x}_j$.) We adopt the superscripts tr and ts to represent the training and test sets, and we let $q_\theta(\mathbf{x}_j^{ts} \mid \mathbf{X}_{-j}^{ts}, \mathbf{X}^{tr})$ represent a TabPFN model (either a regression or a classification model depending on whether the variable j is numeric or categorical), and $\hat{\mathbf{x}}_j^{ts}$ represents the prediction generated by using $\mathbf{X}^{tr}$ for in-context training and $\mathbf{X}_{-j}^{ts}$ for in-context query (where $\mathbf{X}_{-j}$ represents the matrix obtained by removing the jth column from $\mathbf{X}$). Because our goal is to generate synthetic data copies of given datasets, rather than performing supervised learning tasks, our notation does not explicitly differentiate between feature and targets variables (as the same variable can sometimes play the role of a feature and sometimes of a target during the synthetic data generation process). Hence, we use the notation $q_\theta(\mathbf{x}_j^{ts} \mid \mathbf{X}_{-j}^{ts}, \mathbf{X}^{tr}) = q_\theta(\mathbf{x}_j^{ts} \mid \mathbf{X}_{-j}^{ts}, \mathbf{X}_{-j}^{tr}, \mathbf{x}_j^{tr})$ instead of the notation $q_\theta(\mathbf{y}^{ts} \mid \mathbf{X}^{ts}, \mathbf{X}^{tr}, \mathbf{y}^{tr})$, more commonly used in the TabPFN literature.

Previous work addressing synthetic data generation with the TabPFN model included the TabPFGen algorithm (21), which was based on an older version of TabPFN (14), unable to handle regression tasks. To generate continuous data, TabPFGen employs an energy based algorithm for generating features conditional on the classification labels. Energy based algorithms, however, are not necessary for synthetic data generation using TabPFN's current version (15), which handles classification and regression tasks and can be directly used to generate categorical and numerical data.

Hollmann et al. (15) lists synthetic data generation as one of the potential uses of the TabPFN model, and describes it can be used to synthesize new tabular data samples that mimic the statistical properties of a real dataset, by simply sampling each feature step-by-step according to the full factorization of the joint probability distribution of the data (described in equation 1 below). The authors, however, illustrate this strategy using a single dataset and do not provide a formal evaluation of its performance. To the best of our knowledge, the synthetic data generation performance of the TabPFN model has not been systematically evaluated in the literature.

3 Direct strategies for synthetic data generation based on TabPFN

Here we describe the simple strategy, suggested in (15), for performing synthetic data generation with the TabPFN model, alongside an alternative variation of this direct approach. Their main limitations are discussed and illustrated using simulated datasets (where we can control the strength of the statistical associations between the variables).

Let $P(\boldsymbol{X}^{ts} \mid \boldsymbol{X}^{tr})$ represent the posterior predictive distribution (PPD) of the test data conditional on the training data. This conditional joint probability distribution can be fully factorize as,

$$P(\boldsymbol{X}^{ts} \mid \boldsymbol{X}^{tr}) = \prod_{j=1}^{p} P(X_j^{ts} \mid \boldsymbol{X}_{<j}^{ts}, \boldsymbol{X}^{tr}), \tag{1}$$

where $\boldsymbol{X}_{<j}$ represents a subset of $\boldsymbol{X}$ containing elements 1 to $j-1$.

Hollmann et al. (15) suggested generating synthetic data by following the factorization of the joint PPD,

$$P(\boldsymbol{X}^{ts} \mid \boldsymbol{X}^{tr}) \approx \prod_{j=1}^{p} q_\theta(\mathbf{x}_j^{ts} \mid \mathbf{X}_{<j}^{ts}, \mathbf{X}_{<j}^{tr}, \mathbf{x}_j^{tr}), \tag{2}$$

where, due to the in-context learning (ICL) nature of TabPFN, the conditioning on the training set is done on $\mathbf{X}_{<j}^{tr}$ and $\mathbf{x}^{tr}$ rather than $\mathbf{X}^{tr}$. Note that the approximation in equation 2 represents, in actuality, two different levels of approximation. The first is w.r.t. the approximation of the distribution $P(X_j^{ts} \mid \boldsymbol{X}_{<j}^{ts}, \boldsymbol{X}^{tr})$ by the distinct distribution $P(X_j^{ts} \mid \boldsymbol{X}_{<j}^{ts}, \boldsymbol{X}_{<j}^{tr}, X_j^{tr})$, which no longer conditions on training variables $X_{j'}$ for $j' > j$. The second is w.r.t. the approximation of $P(X_j^{ts} \mid \boldsymbol{X}_{<j}^{ts}, \boldsymbol{X}_{<j}^{tr}, X_j^{tr})$ by the transformer model $q_\theta(\mathbf{x}_j^{ts} \mid \mathbf{X}_{<j}^{ts}, \mathbf{X}_{<j}^{tr}, \mathbf{x}_j^{tr})$.

Furthermore, because the TabPFN model cannot condition on an empty set (as you need to provide the model with some input for it to perform ICL) the first product term in equation 2 requires conditioning in a variable X_0, which is not part of the data. Following the suggestion by (15), we adopt a random noise feature as our X_0. A detailed description of the implementation of this strategy, denoted "factorization of the joint PPD" (or JF for short), is provided in Algorithms 2 and 3 in Appendix B. (As pointed by (15), the order of the variables in the joint factorization of the PPD can also affect the results, and (15) suggests using a permutation sampling approximation of Janossy pooling to remedy this issue. This requires, however, the generation and aggregation of multiple synthetic datasets generated from random permutations of the order of the columns of the real data and is not implemented in our experiments.)

Figure 1 illustrates the application of the JF generation strategy (and other approaches that will be described later) to a simulated dataset containing data draw from highly correlated beta distributions. (See Appendix C for details.) To simulate an uninformative feature, we randomly shuffled the data of variable X_2, so that it is completely uncorrelated with the other variables. (Figure 5a in Appendix D shows the correlation matrix for these variables.) The black densities represent the original ("real") data while the orange ones show the synthetic data generated with the JF strategy. Not surprisingly, this example shows that TabPFN provided very poor approximations for the distributions of X_1 and X_2. In the case of X_1, the ICL based on X_0 is poor because X_0 is a random noise variable which

contains no information about X_1. In the case of X_2, the ICL again fails because X_1 does not contain information about X_2 (which is uncorrelated from all other variables).

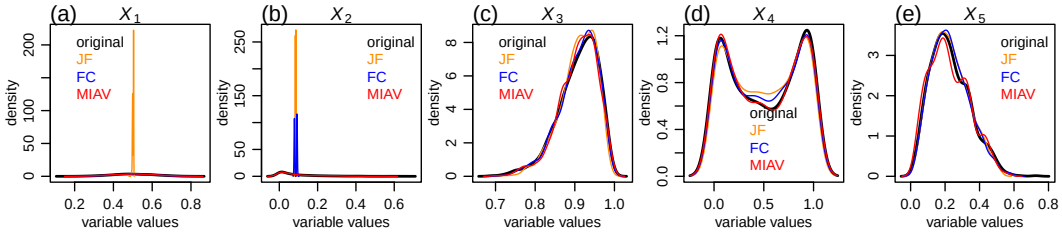

Figure 1: Comparison of marginal distributions generated with the JF, FC, and MIAV strategies.

An alternative approach is to use the full conditional distributions of each variable in the synthetic data generation (denoted as FC, for short). This strategy is implemented using the factorization,

$$\prod_{j=1}^{p} q_\theta(\mathbf{x}_j^{ts} \mid \mathbf{X}_{-j}^{ts}, \mathbf{X}^{tr}) \tag{3}$$

where $\mathbf{X}_{-j}$ represents a subset matrix obtained by dropping column j from $\mathbf{X}$. A detailed description of our implementation of this strategy is provided in Algorithms 4 and 5 in Appendix B.

Despite the fact that this fully conditional factorization does not correspond to a proper factorization of the PPD, this strategy has a few practical advantages. First, it eliminates the need for coming up with a X_0 variable, as the data for X_1 is generated from it's full conditional distribution. Second, the generation of each variable leverages information from all other variables. Third, this approach is unaffected by the order of the variables. It's main practical disadvantage is that it is more expensive to compute since increasing numbers of variables lead to increases in compute time. (As pointed by (27), the time complexity of TabPFN is $O(p^2\, n + n^2\, p)$, where n and p represent, respectively, the number of rows and columns of the data.)

The blue densities in Figure 1 illustrate the application of the FC generation strategy. Now, the distribution of X_1 is nicely recovered by the FC strategy. (This is easier to visualize in Figure 5m in Appendix D, which reports the same results as Figure 1 using a different display.) The approach, however, still fails for X_2 because this variable is uncorrelated to all other variables.

In Appendix E we describe additional variations of the JF direct data generation strategy.

3.1 Performance degradation in datasets containing weakly associated variables

The illustrative examples in Figure 1 were based in data simulated with very strong correlations. The performance of direct strategies such as JF and FC, however, is strongly influenced by the strength of the statistical associations among the variables. Because TabPFN relies on ICL for generating predictions, and weakly associated features provide little information about the target variable, its performance suffers in datasets with weakly associated variables. To illustrate this point, Figure 2 presents the application of the JF and FC strategies (among others) to datasets with decreasing correlation strengths. (The association strength is controlled by the ρ parameter, as described in Appendix C). Due to space limitations, the figure only reports results for a single variable (X_4). The full results are presented in Figures 5 to 9 in the Appendix D. Figure 2 clearly shows that the quality of the synthetic data generated by the JF and FC approaches (orange and blue densities) decreases with decreasing correlation strengths.

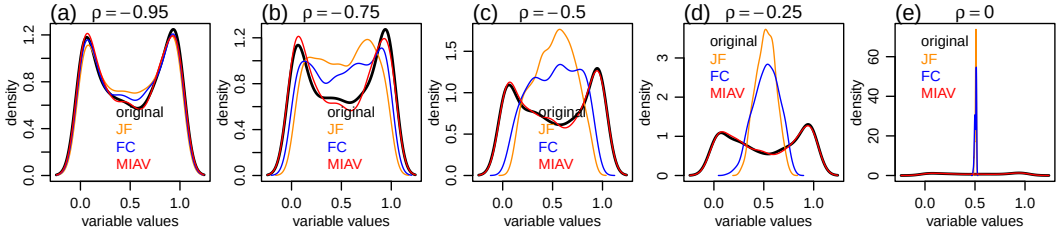

Figure 2: Degrading performance of JF and FC as association strength decreases. MIAV is unaffected.

4 Constructing maximal information auxiliary variables

The previous section illustrates how direct application of TabPFN for synthetic data generation is problematic when variables lack strong statistical associations with other variables. A simple strategy for improving the synthetic data quality is to augment the dataset with auxiliary variables that are highly associated with the real data variables.

To this end, inspired by recent work in non-parametric and model free data synthesis (6), we a propose a simple approach in which we rank-match arbitrary noise variables to the real data variables. The procedure is described in detail in Algorithm 1, and its output is what we denote as a maximal information auxiliary variable (MIAV).

Algorithm 1 GenerateMaximalInformationAuxiliaryVariable($\mathbf{x}_j$)

1: **Input:** a variable data, $\mathbf{x}_j$ {comment here}
2: $\mathbf{m}_j \leftarrow$ Sort(GenerateRandomNoiseVector($n = \text{length}(\mathbf{x}_j)$)) {Generate a sorted random noise vector.}
3: **if** $\mathbf{x}_j$ is numeric **then**
4: $\quad \mathbf{r} \leftarrow \text{Rank}(\mathbf{x}_j)$ {Compute the ranks of $\mathbf{x}_j$. Ties are broken at random.}
5: **end if**
6: **if** $\mathbf{x}_j$ is categorical **then**
7: $\quad \mathbf{r} \leftarrow$ NumericRankEncondingOfCategoticalVariables($\mathbf{x}_j$) {Described in Algorithm 10 in the Appendix.}
8: **end if**
9: $\mathbf{m}_j \leftarrow \mathbf{m}_j[\mathbf{r}]$ {Re-order the entries of $\mathbf{m}_j$ according to the ranks of $\mathbf{x}_j$. The result is a vector $\mathbf{m}_j$ with identical ranks as $\mathbf{x}_j$.}
10: **Output:** the auxiliary variable $\mathbf{m}_j$

The basic idea is to induce a monotonic mapping between a random noise variable and the real data vector $\mathbf{x}_j$. Starting in line 2, the algorithm first draws a sample of size n (equal to the length of $\mathbf{x}_j$) from an arbitrary random noise variable and then sorts it from lowest to highest values. (In our implementation we sample random noise from a uniform distribution in the [0, 1] interval.) In lines 3 to 8 the algorithm ranks the values of $\mathbf{x}_j$. If $\mathbf{x}_j$ is numeric, the algorithm computes its ranks in the standard way (line 4), breaking ties among identical values using the random assignment approach. If $\mathbf{x}_j$ is categorical, the algorithm applies a numeric rank encoding to the categorical variables, as described in Algorithm 10 in Appendix F, and originally proposed by (6). (In a nutshell, Algorithm 10 counts the number elements of $\mathbf{x}_j$ in each factor level and distributes numerical ranks ranging from 1 to n randomly within each level of the categorical variable $\mathbf{x}_j$[1]. Table 1 provides an illustrative example.) Finally, in line 9 the algorithm re-orders the entries of $\mathbf{m}_j$ according to the ranks of $\mathbf{x}_j$. The result is a vector $\mathbf{m}_j$ with identical ranks to $\mathbf{x}_j$, that is, the position of the lowest value of $\mathbf{m}_j$ is the same as the position of the lowest value of $\mathbf{x}_j$, the position of the 2nd lowest value of $\mathbf{m}_j$ is the same as the second lowest value of $\mathbf{x}_j$, and so on. Figure 3 provides some examples. Panels a to e show the distributions of the X_j variables in black and their respective MIAVs, M_j, in red. (The MIAVs follow uniform distributions since in our implementation we draw random noise from uniform distributions.)

[1]As an example, suppose $\mathbf{x}_j = (A, A, A, A, B, B, B, C, C, C, C, C, D, D, D)^t$. This categorical variable has four levels A, B, C and D with counts $n_A = 4$, $n_B = 3$, $n_C = 5$ and $n_D = 3$. To generate the numerical rank encoding $\mathbf{r}$ presented in Table 1, the algorithm transforms the categorical values according to the mapping, map_r, in Table 1. Note that this mapping corresponds to an arbitrary ranking of the categorical levels according to the arbitrary order $A < B < C < D$. That is, starting with class A, the mapping assigns ranks 1, 2, 3, and 4 (in random order) to the four tied elements A in $\mathbf{x}_j$. (Note that by assigning the ranks in random order the mapping is effectively using the "at random" method to break ties among identical values of the categorical variable.) For class B, the mapping assigns ranks 5, 6, and 7 (in random order) to the three tied elements B in $\mathbf{x}_j$. For class C it assigns ranks 8 to 12 (in random order) to the five tied elements C. Finally, for class D, the mapping assigns ranks 13, 14, and 15 (in random order) to the three tied elements D in $\mathbf{x}_j$.

Table 1: Toy example illustrating the application of the numeric rank encoding procedure for categorical variables implemented by Algorithm 10 to $\mathbf{x}_j = (A, A, A, A, B, B, B, C, C, C, C, C, D, D, D)^t$.

map_r	$A = \{1, 2, 3, 4\}$,				$B = \{5, 6, 7\}$,			$C = \{8, 9, 10, 11, 12\}$					$D = \{13, 14, 15\}$		
$\mathbf{x}_j$	A	A	A	A	B	B	B	C	C	C	C	C	D	D	D
$\mathbf{r}$	3	1	4	2	7	5	6	9	8	11	12	10	14	15	13

Variable X_5 is discrete assuming values 1, 2, 3, and 4. Panels g to k show scatterplots of the M_j vs X_j data, illustrating the monotonic relations. Panels f and l show the pairwise associations for the X_j and M_j variables, respectively, and illustrate that, as expected, the M_j variables recapitulate the associations of the X_j data.

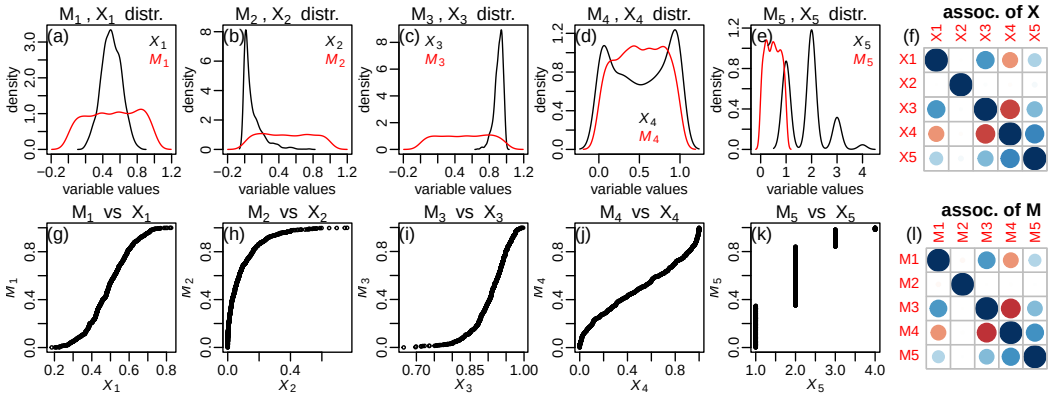

Figure 3: MIAVs illustrative example.

This procedure has some nice theoretical properties, described in the following result.

Theorem 1. *Let M_j represent the auxiliary variable generated from X_j by Algorithm 1, and Y represent an arbitrary variable other than X_j or M_j. Then, non-parametrically,*

1. *$I(X_j; Y \mid M_j) = 0$, i.e., the conditional mutual information of X_j and Y given M_j is 0.*
2. *$H(X_j \mid M_j) = 0$, i.e., the conditional entropy of X_j given M_j is 0.*

The proof is presented in Appendix G. Note that this result holds non-parametrically, in the sense that any continuous variable is first discretized into n bins (where n represents sample size), so that we can use the discrete probability (and non-parametric) definitions of these information-theoretic quantities. This is justifiable because a sample of size n from a continuous variable can always be viewed as the sample of a discrete variable with n distinct levels, each observed with frequency $1/n$.

This result shows that the auxiliary variable M_j, generated by Algorithm 1 has the following two key properties. First, it contains maximal information about X_j. (This holds in the conditional entropy sense, since $H(X_j \mid M_j) = 0$ implies that X_j is completely determined by M_j.). Second, conditional of M_j, X_j is statistically independent of any other variables. These two properties are key for the synthetic data generation approach that we propose next.

5 Synthetic data generation using maximal information auxiliary variables

Let $\boldsymbol{M} = (M_1, \ldots, M_p)^t$ represent the set of random variables M_j generated by Algorithm 1. Now, consider the augmented posterior predictive distribution of $\boldsymbol{X}^{ts}$ given $\boldsymbol{X}^{tr}$, $\boldsymbol{M}^{ts}$, and $\boldsymbol{M}^{tr}$,

$$P(\boldsymbol{X}^{ts} \mid \boldsymbol{X}^{tr}, \boldsymbol{M}^{ts}, \boldsymbol{M}^{tr}) = \prod_{j=1}^{p} P(X_j^{ts} \mid \boldsymbol{X}_{<j}^{ts}, \boldsymbol{X}^{tr}, \boldsymbol{M}^{ts}, \boldsymbol{M}^{tr}). \tag{4}$$

Now re-writing,

$$P(X_j^{ts} \mid \boldsymbol{X}_{<j}^{ts}, \boldsymbol{X}^{tr}, \boldsymbol{M}^{ts}, \boldsymbol{M}^{tr}) = P(X_j^{ts} \mid \boldsymbol{X}_{<j}^{ts}, \boldsymbol{X}^{tr}, \boldsymbol{M}_{-j}^{ts}, M_j^{ts}, \boldsymbol{M}_{-j}^{tr}, M_j^{tr}) \,, \tag{5}$$

and recalling that X_j^{ts} is independent from all other variables conditional on M_j^{ts}, its follows that,

$$P(X_j^{ts} \mid \boldsymbol{X}_{<j}^{ts}, \boldsymbol{X}^{tr}, \boldsymbol{M}_{-j}^{ts}, M_j^{ts}, \boldsymbol{M}_{-j}^{tr}, M_j^{tr}) = P(X_j^{ts} \mid M_j^{ts}) \,, \tag{6}$$

which, for the same reason, can also be re-expressed as,

$$P(X_j^{ts} \mid M_j^{ts}) = P(X_j^{ts} \mid M_j^{ts}, M_j^{tr}, X_j^{tr}) \,, \tag{7}$$

a format that is better suited for performing in-context learning with a PFN model. Hence, the PPD augmented with the set of maximal information auxiliary variables can formally be expressed as,

$$P(\boldsymbol{X}^{ts} \mid \boldsymbol{X}^{tr}, \boldsymbol{M}^{ts}, \boldsymbol{M}^{tr}) = \prod_{j=1}^{p} P(X_j^{ts} \mid M_j^{ts}, M_j^{tr}, X_j^{tr}), \tag{8}$$

and readily approximated by a trained TabPFN model as,

$$P(\boldsymbol{X}^{ts} \mid \boldsymbol{X}^{tr}, \boldsymbol{M}^{ts}, \boldsymbol{M}^{tr}) \approx \prod_{j=1}^{p} q_\theta(\mathbf{x}_j^{ts} \mid \mathbf{m}_j^{ts}, \mathbf{m}_j^{tr}, \mathbf{x}_j^{tr}), \tag{9}$$

where ICL for each variable X_j is performed by training on $\mathbf{m}_j^{tr}$ and $\mathbf{x}_j^{tr}$ and querying on $\mathbf{m}_j^{ts}$.[2]

This formulation, denoted the "Maximal Information Auxiliary Variables" strategy (or MIAV strategy for short) has several practical advantages. First, it is the most efficient strategy in terms of computation, since ICL of each variable X_j is performed using a single variable M_j (recall that the complexity of the TabPFN model scales quadratically on the number of columns of the table). Second, contrary to all the other direct generation strategies described in Section 3, the MIAV approach is based on a proper factorization of the (augmented) PPD (the approximation in equation 9 is only w.r.t. the transformer model approximation to the predictive distribution). Third, contrary to the JF strategy, the MIAV approach is invariant with respect to the order of the dataset columns. Fourth, and most importantly, the MIAV approach handles uninformative features and performs well in datasets containing weakly associated features. This last point is illustrated by the red densities in Figures 1 and 2 (see also panels s to w in Figures 5 to 9 in Appendix D), where the MIAV approach closely recapitulates the original marginal distributions. Quite importantly, note that while the augmented joint PPD in equation 8 factorizes into separate $P(X_j^{ts} \mid X_j^{tr}, M_j^{ts}, M_j^{tr})$ components, the synthetic data generated by the $q_\theta(\mathbf{x}_j^{ts} \mid \mathbf{x}_j^{tr}, \mathbf{m}_j^{tr}, \mathbf{m}_j^{ts})$ still recapitulates the association structure of $\mathbf{X}$ because this association is indirectly induced by the MIAVs (recall that $\mathbf{M}$ mimics the association structure in $\mathbf{X}$, as illustrated in panels f and l of Figure 3. Algorithms 6 and 7 in Appendix B provide implementation details about the MIAV synthetic data generation approach.

6 Experiments

We performed two sets of experiments. One using simulated data draw from correlated beta distributions (generated as described in Appendix C), for which we can control the strength of the associations among the variables, as well as, using 21 real-world datasets. Each dataset was split into two equal sized subsets. The first was fed to the synthesizers and is denoted the "original" data. The second is never touched by the synthesizers and is denoted the "holdout" data. (Appendix H.1 describes these data splits alongside the additional splits used for ICL.) Synthetic datasets generated from the original data were evaluated with respect to fidelity and privacy metrics. The fidelity metrics included the average KS-test statistic (KS), the L2 distance between association matrices (L2D), and the energy distance (ED) between multivariate distributions. The privacy metrics included the distance to closest record (DCR), the sorted distance-based record linkage (SDBRL) metric, and the sorted standard deviation interval distance (SSDID) metric. In addition to comparing the MIAV-based synthetic data generation strategy against the joint factorization (JF) and full conditional (FC) approaches, our evaluations also include the SMOTE generator (7) (which is well known for generating very high fidelity synthetic data which also tends to be less private than other baselines in the literature (19; 17)). We selected the SMOTE baseline because, contrary to the other deep-learning based generators, it can be applied to the small datasets used in our evaluations. Furthermore, we also include comparisons against the holdout sets to get a sense about the range of values we would expect to see for each evaluation metric in the ideal case of a generator truly able to draw samples from the same distribution as the original data. To increase the statistical validity of our experimental results, for each real-world dataset we report the results from 10 experimental replications based on distinct original/holdout data splits. The results for the correlated beta distribution experiments were based on 10 distinct replications (based on different simulation parameters) per simulation setting. We considered 5 distinct

[2]Here, it is important to clarify that we are able to condition on the test set auxiliary variables, $\boldsymbol{M}^{ts}$, because we always have unrestricted access to the full $\mathbf{X}$ data (which is only split into $\mathbf{X}^{tr}$ and $\mathbf{X}^{ts}$ for the sake of ICL). Hence, the test set is always available and we can generate the corresponding MIAV matrix $\mathbf{M}^{ts}$. (Our goal is to generate a synthetic copy of the original data, rather than making predictions about unseen test data.)

simulation settings based on absolute correlations $|\rho| = \{0, 0.25, 0.5, 0.75, 0.95\}$. Descriptions of the evaluation metrics and further details about the experiments are presented in Appendix H.

Figure 4 reports the results (pooled across datasets). In terms of fidelity, SMOTE tended to perform slightly better than MIAV, which clearly outperformed JF and FC (see panels a, b, c, g, h, i, where lower values of the KS, L2D, and ED metrics indicate better fidelity). MIAV tended to generate more private data than SMOTE (see panels d and j, where higher DCR values indicate better privacy), and considerably more so w.r.t. the SDBRL and SSDID metrics (see panels e and k for the SDBRL metric, and panels f and l for SSDID, where lower SDBRL and SSDID values indicate better privacy). JF and FC tended to produce more private data than MIAV (often times being even more private than the holdout set). This was achieved, however, at the expense of considerably lower fidelity. Overall, the fidelity vs privacy tradeoff patterns observed in the simulated datasets (top panels) were similar to the real datasets (bottom panels).

These experiments suggest a competitive performance of the MIAV-based synthetic data generator, as it achieved a better privacy-vs-fidelity tradeoff than SMOTE (namely, a higher reduction in privacy risk at the expense of a small decrease in data fidelity). We expect the MIAV approach to be particularly effective in small data settings (which tend to be challenging for traditional synthetic data generators, and are better handled by TabPFN).

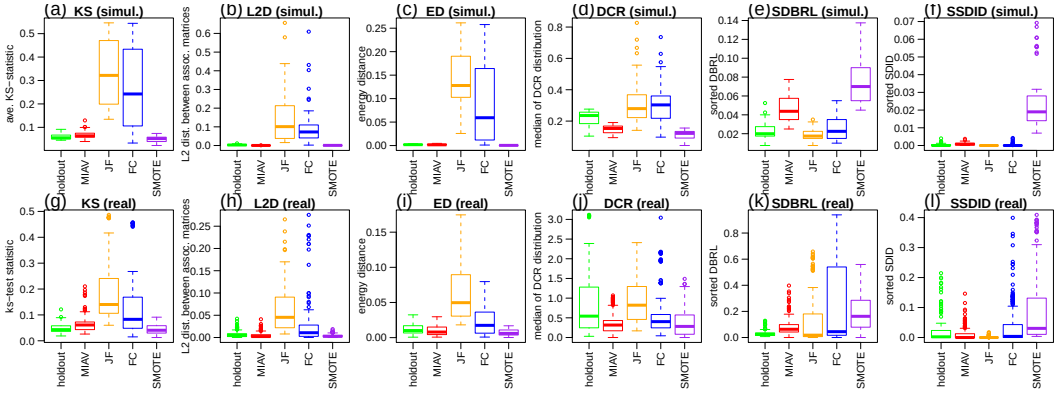

Figure 4: Pooled experimental results. Top panels show results pooled across the 5 simulated dataset settings. The bottom panels show results pooled across 21 real-world datasets. The KS, L2D, and ED metrics evaluate the synthetic data with respect to the quality of the marginal distributions, quality of the pairwise statistical associations, and quality of the joint probability distribution, respectively. The DCR, SDBRL, and SSDID metrics measure privacy risks with respect to re-identification and attribute disclosure risks. See Appendix H.4 for further details. See also Figures 11 and 12 in H.5.

7 Final remarks

Our approach relies on TabPFN for in-context learning and therefore inherits its limitations. These include the current data size cap of roughly 10,000 rows (though future releases are expected to relax this), memory usage that scales linearly with dataset size, and slower inference compared with some baselines. The more scalable PFN-based TabICL model (27) alleviates some of these constraints, but at present supports only classification. Future versions of TabICL that extend to regression could potentially be combined with the MIAV strategy to address these limitations.

For tabular foundation models that do not approximate Bayesian inference, our approach may offer a natural in-context strategy for synthetic data generation, though assessing such extensions is left for future work. Another interesting future research direction is to improve our approach's privacy. Although it already tends to produce more private data than SMOTE, further privacy improvements can be achieved through the addition of controlled amounts of noise to the MIAVs.

Finally, we emphasize that the MIAV strategy is intended solely for synthetic data generation. Because computing MIAVs requires access to test-set targets, the method cannot be used to improve supervised learning performance. In particular, when a full dataset is simply split into training and test sets for evaluation, generating MIAVs for the test targets would leak target information into the inputs, leading to artificially inflated predictive performance due to data leakage.

References

[1] Borisov, V., Sessler, K., Leemann, T., Pawelczyk, M., and Kasneci, G. (2022). Language models are realistic tabular data generators, in: The Eleventh International Conference on Learning Representations (ICLR), 2023.

[2] den Breejen, F., Bae, S., Cha, S., and Yun, S.-Y. (2024). Why In-Context Learning Transformers are Tabular Data Classifiers. arXiv:2405.13396v1.

[3] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, et al. (2020). Language Models are Few-Shot Learners. NeurIPS 2020.

[4] Bischl, B., Casalicchio, G., Feurer, M., Hutter, F., Lang, M., Mantovani, R. G., van Rijn, J. N., & Vanschoren, J. (2019). OpenML Benchmarking Suites. NeurIPS 2021.

[5] Chaibub Neto, E. (2024). Statistical disclosure control for numeric microdata via sequential joint probability preserving data shuffling. Transactions on Data Privacy, 17(3):147-179, 2024.

[6] Chaibub Neto, E. (2025). TabSDS: a Lightweight, Fully Non-Parametric, and Model Free Approach for Generating Synthetic Tabular Data. ICML, 2025.

[7] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16, 321-357.

[8] Cresswell, J. C., and Kim, T. (2024). Scaling up diffusion and flow-based XGBoost models. arXiv 2408.16046

[9] Domingo-Ferrer, J., Muralidhar, K., and Bras-Amoros, M. (2020). General confidentiality and utility metrics for privacy-preserving data publishing based on the permutation model. IEEE Transactions on Dependable and Secure Computing, 18(5):2506-2517. DOI: 10.1109/TDSC.2020.2968027.

[10] Domingo-Ferrer, J., and Torra, V. (2001). A quantitative comparison of disclosure control methods for microdata. In Confidentiality, disclosure, and data access: theory and practical applications for statistical agencies, edited by P. Doyle, J. Lane, J. Theeuwes, and L. Zayatz. 111 133. Elsevier.

[11] Drechsler, J. (2011). Synthetic Data Sets for Statistical Disclosure Control. Springer, New York.

[12] Feuer, B., Schirrmeister, R. T., Cherepanova, V., Hegde, C., Hutter, F., Goldblum, M., Cohen, N., and White, C. (2024) TuneTables: Context Optimization for Scalable Prior-Data Fitted Networks. arXiv preprint arXiv:2402.11137.

[13] Garg, A., Ali, M., Hollmann, N., Purucker, L., Muller, S., Hutter, F. (2025). Real-TabPFN: Improving Tabular Foundation Models via Continued Pre-training With Real-World Data. Proceedings of the 1st ICML Workshop on Foundation Models for Structured Data.

[14] Hollmann, N., Muller, S., Eggensperger, K., Hutter, F.. TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second. International Conference on Learning Representations (ICLR) 2023.

[15] Hollmann, N., Muller, S., Purucker, L., Krishnakumar, A., Korfer, M., Hoo, S. B., Schirrmeister, R. T., & Hutter, F. (2025). Accurate predictions on small data with a tabular foundation model. Nature, 637, 319-326.

[16] Jolicoeur-Martineau, A., Fatras, K., and Kachman, T. (2024). Generating and imputing tabular data via diffusion and flow-based gradient-boosted trees. AISTATS 2024

[17] Kindji, G. C. N., Rojas-Barahona, L. M., Fromont, E., and Urvoy, T. (2024). Under the hood of tabular data generation models: benchmarks with extensive tuning. arXiv:2406.12945

[18] Koshil, M., Nagler, T., Feurer, M., and Eggensperger, K. (2024). Towards Localization via Data Embedding for TabPFN. Third Table Representation Learning Workshop at NeurIPS 2024.

[19] Kotelnikov, A., Baranchuk, D., Rubachev, I., and Babenko, A. (2023). Tabddpm: Modelling tabular data with diffusion models. In International Conference on Machine Learning (pp. 17564-17579). PMLR.

[20] Mateo-Sanz, J.M., Sebe, F., and Domingo-Ferrer, J. (2004). Outlier protection in continuous microdata masking. International Workshop on Privacy in Statistical Databases. PSD 2004: Privacy in Statistical Databases pp 201-215.

[21] Ma, J., Dankar, A., Stein, G., Yu, G., & Caterini, A. L. (2023). TabPFGen - Tabular Data Generation with TabPFN. Poster presented at the Table Representation Learning Workshop, NeurIPS 2023

[22] Ma, J., Thomas, V., Hosseinzadeh, R., Kamkari, H., Labach, A., Cresswell, J. C., Golestan, K., Yu, G., Volkovs, M., and Caterini, A. L. (2024). TabDPT: Scaling Tabular Foundation Models. arXiv:2410.18164.

[23] Ma, J., Thomas, V., Yu, G., and Caterini, A. (2024). In-Context Data Distillation with TabPFN. arXiv preprint arXiv:2402.06971.

[24] Muller, A., Curino, C., and Ramakrishnan, R. (2023). MotherNet: Fast Training and Inference via Hyper-Network Transformers. ICLR 2025.

[25] Nowok, B., Raab, G. M., and Dibben, C. (2016). synthpop: bespoke creation of synthetic data in R. Journal of Statistical Software, 74(11), 1-26. https://doi.org/10.18637/jss.v074.i11

[26] Pagliuca, D., and Seri, G. (1999). Some results of individual ranking method on the system of enterprise accounts annual survey, Esprit SDC Project, Deliverable MI-3/D2.

[27] Qu, J., Holzmller, D., Varoquaux, G., & Le Morvan, M. (2025). TabICL: A Tabular Foundation Model for In-Context Learning on Large Data. In Proceedings of the 42nd International Conference on Machine Learning (ICML 2025).

[28] Rajotte, J. F., Bergen, R., Buckeridge, D. L., El Emam, K., Ng, R., & Strome, E. (2022). Synthetic data as an enabler for machine learning applications in medicine. iScience, 25(11), 105331.

[29] Reiter, J. P. (2005). Using CART to generate partially synthetic, public use microdata. Journal of Official Statistics, 21(3):441-462.

[30] Shi, J., Xu, M., Hua, H., Zhang, H., Ermon, S., and Leskovec, J. (2025). TabDiff: a mixed-type diffusion model for tabular data generation. ICLR 2025.

[31] Szekely, G. J., & Rizzo, M. L. (2023). The Energy of Data and Distance Correlation. Chapman Hall/CRC, Monographs on Statistics and Applied Probability, Vol. 171. 466 pages.

[32] Watson, D. S., Blesch, K., Kapar, J., and Wright, M. N. (2023). Adversarial random forests for density estimation and generative modeling. In International Conference on Artificial Intelligence and Statistics (pp. 5357-5375). PMLR.

[33] Xu, L., Skoularidou, M., Cuesta-Infante, A., and Veeramachaneni, K. (2019). Modeling tabular data using conditional GAN. Advances in Neural Information Processing Systems, 32, 2019.

[34] Xu, D., Cirit, O., Asadi, R., Sun, Y., and Wang, W. (2024) Mixture of In-Context Prompters for Tabular PFNs. ICLR 2025.

[35] Young, J., Graham, P., and Penny, R. (2009). Using bayesian networks to create synthetic data. Journal of Official Statistics, 25(4):549.

[36] Zeng, Y., Kang, W., and Mueller, A. C. (2024) Tabflex: Scaling tabular learning to millions with linear attention. In NeurIPS 2024 Third Table Representation Learning Workshop.

[37] Zhang, H., Zhang, J., Srinivasan, B., Shen, Z., Qin, X., Faloutsos, C., Rangwala, H., and Karypis, G. (2023). Mixed-type tabular data synthesis with score-based diffusion in latent space. in: The Twelfth International Conference on Learning Representations, 2024.

A Supervised learning versus synthetic data generation in the presence of uninformative features

TabPFN achieves state-of-the-art performance in supervised learning tasks (15). However, its direct application to synthetic data generation does not yield comparable results, as weakly associated variables make it challenging for in-context learning methods to capture the true data distribution. This issue is less pronounced in supervised learning, where noninformative features simply fail to contribute to the prediction. Even when the prediction is based solely on such features, TabPFN behaves appropriately. For instance, in binary classification with predictors completely uncorrelated with the target, it produces an AUROC of about 0.5, consistent with random guessing. Although uninteresting, this outcome is exactly what one would expect from any well-behaved classifier in this scenario. By contrast, in the data generation setting, TabPFN's reasonable performance in classification does not carry over, as it fails to approximate the marginal distributions of the data and to capture its statistical association structure. An illustrative example is presented in Figure 9 in Section D, where direct application of TabPFN to datasets containing completely uncorrelated data fails to recover the marginal distributions (see panels g, h, i, j, and k and m, n, o, p, and q), and fails to recover the correlation structure of the original data (compare panels b and c against panel a).

B TabPFN-based synthetic data generators

In Algorithms 2, 4, and 6, the function `GeneratePredictionUsingTabPFN(.)` represents a call to a TabPFN model $q_\theta(.)$.

Algorithm 2 ICLwithJointFactorizationTabPFN($\mathbf{X}^{tr}$, $\mathbf{X}^{ts}$)

Input: training data for ICL, $\mathbf{X}^{tr}$; query data for ICL, $\mathbf{X}^{ts}$
$n_{tr} \leftarrow \text{NumberOfRows}(\mathbf{X}^{tr})$ {Obtain number of samples of $\mathbf{X}^{tr}$.}
$n_{ts} \leftarrow \text{NumberOfRows}(\mathbf{X}^{ts})$ {Obtain number of samples of $\mathbf{X}^{ts}$.}
$p \leftarrow \text{NumberOfColumns}(\mathbf{X}^{ts})$ {Obtain number of columns of $\mathbf{X}^{ts}$.}
$\mathbf{Z}^{ts} \leftarrow [,]$ {Create empty matrix to store the synthetic data.}
$\mathbf{x}_0^{tr} \leftarrow \text{GenerateUniformlyDistributedNoise}(n_{tr})$ {Draw a sample of size n_{tr} from a uniform distribution.}
$\mathbf{x}_0^{ts} \leftarrow \text{GenerateUniformlyDistributedNoise}(n_{ts})$ {Draw a sample of size n_{ts} from a uniform distribution.}
$\mathbf{Z}^{ts}[,1] \leftarrow \text{GeneratePredictionUsingTabPFN}(\mathbf{x}_0^{ts}, \mathbf{x}_0^{tr}, \mathbf{x}_1^{tr})$ {Predict $\mathbf{x}_1^{ts}$ using $\mathbf{x}_0^{tr}$ and $\mathbf{x}_1^{tr}$ as context, and $\mathbf{x}_0^{ts}$ as query. The prediction can be from a regression or classification TabPFN model, depending on whether $\mathbf{x}_1^{tr}$ is continuous or categorical.}
for $j = 2$ **to** p **do**
    $\mathbf{X}_{<j}^{tr} \leftarrow \mathbf{X}^{tr}[,1:(j-1)]$ {Select the first $j-1$ columns of $\mathbf{X}^{tr}$.}
    $\mathbf{X}_{<j}^{ts} \leftarrow \mathbf{X}^{ts}[,1:(j-1)]$ {Select the first $j-1$ columns of $\mathbf{X}^{ts}$.}
    $\mathbf{Z}^{ts}[,j] \leftarrow \text{GeneratePredictionUsingTabPFN}(\mathbf{X}_{<j}^{ts}, \mathbf{X}_{<j}^{tr}, \mathbf{x}_j^{tr})$ {Predict $\mathbf{x}_j^{ts}$ using $\mathbf{X}_{<j}^{tr}$ and $\mathbf{x}_j^{tr}$ as context, and $\mathbf{X}_{<j}^{ts}$ as query. The prediction can be from a regression or classification TabPFN model, depending on whether $\mathbf{x}_j^{tr}$ is continuous or categorical.}
end for
Output: synthetic data $\mathbf{Z}^{ts}$

Algorithm 3 JointFactorizationTabPFNGenerator($\mathbf{X}$)

Input: the original data, $\mathbf{X}$
$\mathbf{X}_1, \mathbf{X}_2 \leftarrow \text{DataSplit}(\mathbf{X})$ {Split the original data $\mathbf{X}$ into two subsets, $\mathbf{X}_1$ and $\mathbf{X}_2$.}
$\mathbf{Z}_1 \leftarrow \text{ICLwithJointFactorizationTabPFN}(\mathbf{X}^{tr} = \mathbf{X}_2, \mathbf{X}^{ts} = \mathbf{X}_1)$ {Generate a synthetic data copy of $\mathbf{X}_1$ using Algorithm 2.}
$\mathbf{Z}_2 \leftarrow \text{ICLwithJointFactorizationTabPFN}(\mathbf{X}^{tr} = \mathbf{X}_1, \mathbf{X}^{ts} = \mathbf{X}_2)$ {Generate a synthetic data copy of $\mathbf{X}_2$ using Algorithm 2.}
$\mathbf{Z} \leftarrow \text{Concatenate}(\mathbf{Z}_1, \mathbf{Z}_2)$ {Concatenate the synthetic datasets $\mathbf{Z}_1$ and $\mathbf{Z}_2$.}
$\mathbf{Z} \leftarrow \text{RoundIntegerVariables}(\mathbf{X}, \mathbf{Z})$ {This function uses $\mathbf{X}$ to determine which variables have integer type and round the values of the corresponding variables in $\mathbf{Z}$ to the nearest integer.}
Output: synthetic data $\mathbf{Z}$

Algorithm 4 ICLwithFullConditionalsTabPFN($\mathbf{X}^{tr}$, $\mathbf{X}^{ts}$)

Input: training data for ICL, $\mathbf{X}^{tr}$; query data for ICL, $\mathbf{X}^{ts}$
$p \leftarrow$ NumberOfColumns($\mathbf{X}^{ts}$) {Obtain number of columns of $\mathbf{X}^{ts}$.}
$\mathbf{Z}^{ts} \leftarrow [,]$ {Create empty matrix to store the synthetic data.}
for $j = 1$ **to** p **do**
  $\mathbf{X}^{tr}_{-j} \leftarrow \mathbf{X}^{tr}[,-j]$ {Drop the j-th column of $\mathbf{X}^{tr}$.}
  $\mathbf{X}^{ts}_{-j} \leftarrow \mathbf{X}^{ts}[,-j]$ {Drop the j-th column of $\mathbf{X}^{ts}$.}
  $\mathbf{Z}^{ts}[,j] \leftarrow$ GeneratePredictionUsingTabPFN($\mathbf{X}^{ts}_{-j}, \mathbf{X}^{tr}_{-j}, \mathbf{x}^{tr}_{j}$) {Predict $\mathbf{x}^{ts}_{j}$ using $\mathbf{X}^{tr}_{-j}$ and $\mathbf{x}^{tr}_{j}$ as context, and $\mathbf{X}^{ts}_{-j}$ as query. The prediction can be from a regression or classification TabPFN model, depending on whether $\mathbf{x}^{tr}_{j}$ is continuous or categorical.}
end for
Output: synthetic data $\mathbf{Z}^{ts}$

Algorithm 5 FullConditionalsTabPFNGenerator($\mathbf{X}$)

Input: the original data, $\mathbf{X}$
$\mathbf{X}_1, \mathbf{X}_2 \leftarrow$ DataSplit($\mathbf{X}$) {Split the original data $\mathbf{X}$ into two subsets, $\mathbf{X}_1$ and $\mathbf{X}_2$.}
$\mathbf{Z}_1 \leftarrow$ ICLwithFullConditionalsTabPFN($\mathbf{X}^{tr} = \mathbf{X}_2, \mathbf{X}^{ts} = \mathbf{X}_1$) {Generate a synthetic data copy of $\mathbf{X}_1$ using Algorithm 4.}
$\mathbf{Z}_2 \leftarrow$ ICLwithFullConditionalsTabPFN($\mathbf{X}^{tr} = \mathbf{X}_1, \mathbf{X}^{ts} = \mathbf{X}_2$) {Generate a synthetic data copy of $\mathbf{X}_2$ using Algorithm 4.}
$\mathbf{Z} \leftarrow$ Concatenate($\mathbf{Z}_1, \mathbf{Z}_2$) {Concatenate the synthetic datasets $\mathbf{Z}_1$ and $\mathbf{Z}_2$.}
$\mathbf{Z} \leftarrow$ RoundIntegerVariables($\mathbf{X}, \mathbf{Z}$)
Output: synthetic data $\mathbf{Z}$

Algorithm 6 ICLwithMIAVTabPFN($\mathbf{X}^{tr}$, $\mathbf{X}^{ts}$)

Input: training data for ICL, $\mathbf{X}^{tr}$; query data for ICL, $\mathbf{X}^{ts}$
$p \leftarrow$ NumberOfColumns($\mathbf{X}^{ts}$) {Obtain number of columns of $\mathbf{X}^{ts}$.}
$\mathbf{Z}^{ts} \leftarrow [,]$ {Create empty matrix to store the synthetic data.}
for $j = 1$ **to** p **do**
  $\mathbf{m}^{tr}_{j} \leftarrow$ GenerateMaximalInformationAuxiliaryVariable($\mathbf{x}^{tr}_{j}$) {Generate the MIAV for $\mathbf{x}^{tr}_{j}$ using Algorithm 1.}
  $\mathbf{m}^{ts}_{j} \leftarrow$ GenerateMaximalInformationAuxiliaryVariable($\mathbf{x}^{ts}_{j}$) {Generate the MIAV for $\mathbf{x}^{ts}_{j}$ using Algorithm 1.}
  $\mathbf{Z}^{ts}[,j] \leftarrow$ GeneratePredictionUsingTabPFN($\mathbf{m}^{ts}_{j}, \mathbf{m}^{tr}_{j}, \mathbf{x}^{tr}_{j}$) {Predict $\mathbf{x}^{ts}_{j}$ using $\mathbf{m}^{tr}_{j}$ and $\mathbf{x}^{tr}_{j}$ as context, and $\mathbf{m}^{ts}_{j}$ as query. The prediction can be from a regression or classification TabPFN model, depending on whether $\mathbf{x}^{tr}_{j}$ is continuous or categorical.}
end for
Output: synthetic data $\mathbf{Z}^{ts}$

Algorithm 7 MIAVTabPFNGenerator($\mathbf{X}$)

Input: the original data, $\mathbf{X}$
$\mathbf{X}_1, \mathbf{X}_2 \leftarrow$ DataSplit($\mathbf{X}$) {Split the original data $\mathbf{X}$ into two subsets, $\mathbf{X}_1$ and $\mathbf{X}_2$.}
$\mathbf{Z}_1 \leftarrow$ ICLwithMIAVTabPFN($\mathbf{X}^{tr} = \mathbf{X}_2, \mathbf{X}^{ts} = \mathbf{X}_1$) {Generate a synthetic data copy of $\mathbf{X}_1$ using Alg. 6.}
$\mathbf{Z}_2 \leftarrow$ ICLwithMIAVTabPFN($\mathbf{X}^{tr} = \mathbf{X}_1, \mathbf{X}^{ts} = \mathbf{X}_2$) {Generate a synthetic data copy of $\mathbf{X}_2$ using Alg. 6.}
$\mathbf{Z} \leftarrow$ Concatenate($\mathbf{Z}_1, \mathbf{Z}_2$) {Concatenate the synthetic datasets $\mathbf{Z}_1$ and $\mathbf{Z}_2$.}
$\mathbf{Z} \leftarrow$ RoundIntegerVariables($\mathbf{X}, \mathbf{Z}$) {This function uses $\mathbf{X}$ to determine which variables have integer type and round the values of the corresponding variables in $\mathbf{Z}$ to the nearest integer.}
Output: synthetic data $\mathbf{Z}$

In Algorithms 3, 5, and 7 the function `RoundIntegerVariables`($\mathbf{X}$, $\mathbf{Z}$) uses the original data, $\mathbf{X}$, to determine which variables have integer type and round the values of the corresponding variables in the synthetic data, $\mathbf{Z}$, to the nearest integer. This post-processing step is necessary because TabPFN (and most other data synthesizers) return real values for variables that are originally of integer type. In our experiments, we apply the same post-processing step to the SMOTE baseline.

C Simulating correlated beta distributions

Here, we describe how we simulated correlated beta distributions used in the simulated data experiments, as well as, for the illustrative examples provided in Figures 1, 2, and 3 in the main text, and Figures 5, 6, 7, 8, and 9 in Appendix D.

The data was simulated as follows. First, we simulate data from a multivariate normal random variable $\boldsymbol{Y} \sim N_p(\mathbf{0}, \boldsymbol{\Sigma})$. Next, for $j = 1, \ldots, p$, we compute the correlated uniform variables $U_j = \Phi(Z_j)$, and the correlated beta random variables $X_j = G^{-1}_{a,b}(U_j)$, where Φ and $G_{a,b}$ represent, respectively, the cumulative distribution functions of standard normal variable and a beta variable with shape parameters a and b.

The multivariate Gausssian variable $\boldsymbol{Y}$ is generated with a Toeplitz structured covariance matrix $\boldsymbol{\Sigma}$ with off-diagonal entries $\sigma_{ij} = \rho^{|i-j|}$, and diagonal entries $\sigma_{jj} = 1$, for $\rho \in [-1, 1]$. (Note that under this correlation structure, neighboring variables are more highly associated than more distant variables, and the association decreases the farther apart the variables are. Also, for negative ρ values the direction of the association flips depending on whether the exponent $|i - j|$ is even or odd.)

For the illustrations presented in Figures 1, 2, and 3 we further randomly shuffle the data of variable X_2 in order to simulate an uninformative feature uncorrelated with all other variables in the dataset. Furthermore, for the illustrations presented in Figure 3, we also discretize variable X_5 into four classes.

D Supplementary Figures

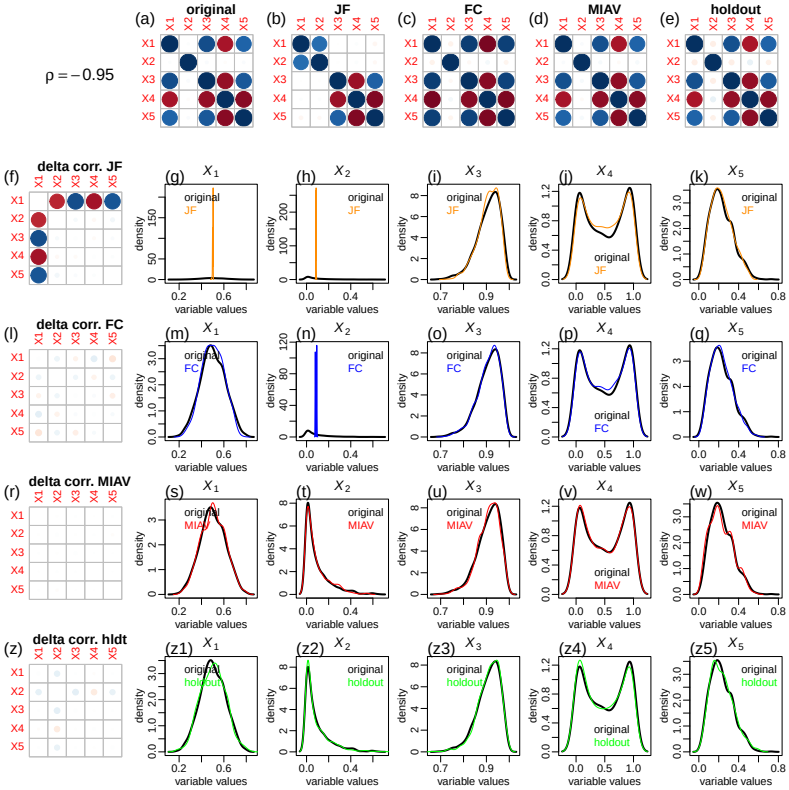

Figure 5: Comparison of the JF, FC, and MIAV synthetic data generation strategies for correlated beta variables generated with $\rho = -0.95$. Panel a shows the Pearson correlation matrices for the original data, while panels b to e show the correlation matrices for the JF, FC, MIAV synthetic datasets and the holdout set. Panels f, l, r, and z show the difference between the original data correlation matrix and the respective synthetic datasets and holdout set. The remaining panels show the marginal distributions generated by the distinct synthetic data generation approaches and for the holdout set.

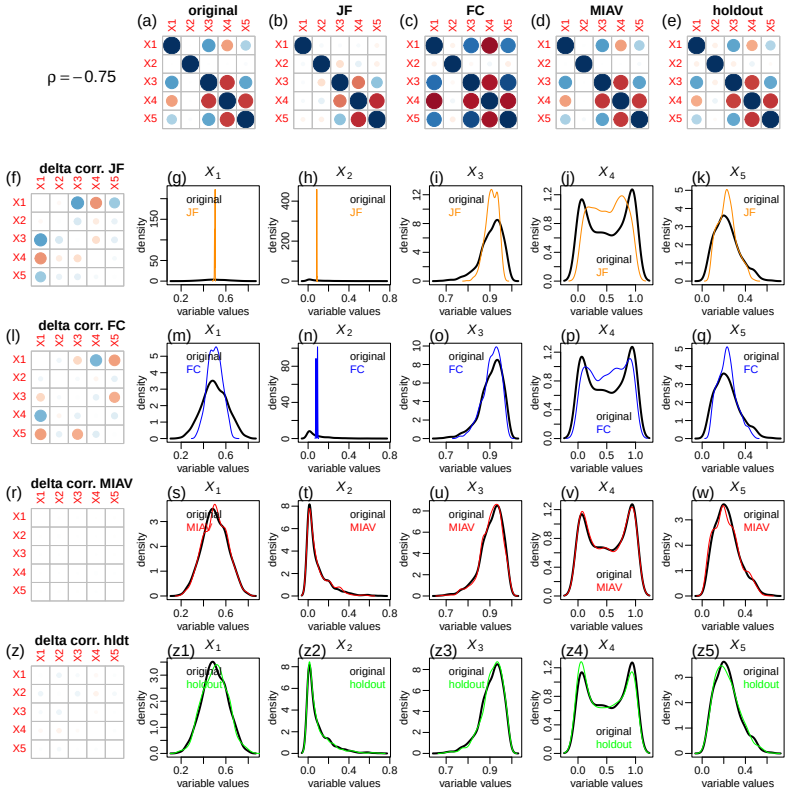

Figure 6: Analogous comparisons as in Figure 5, but for data simulated with $\rho = -0.75$.

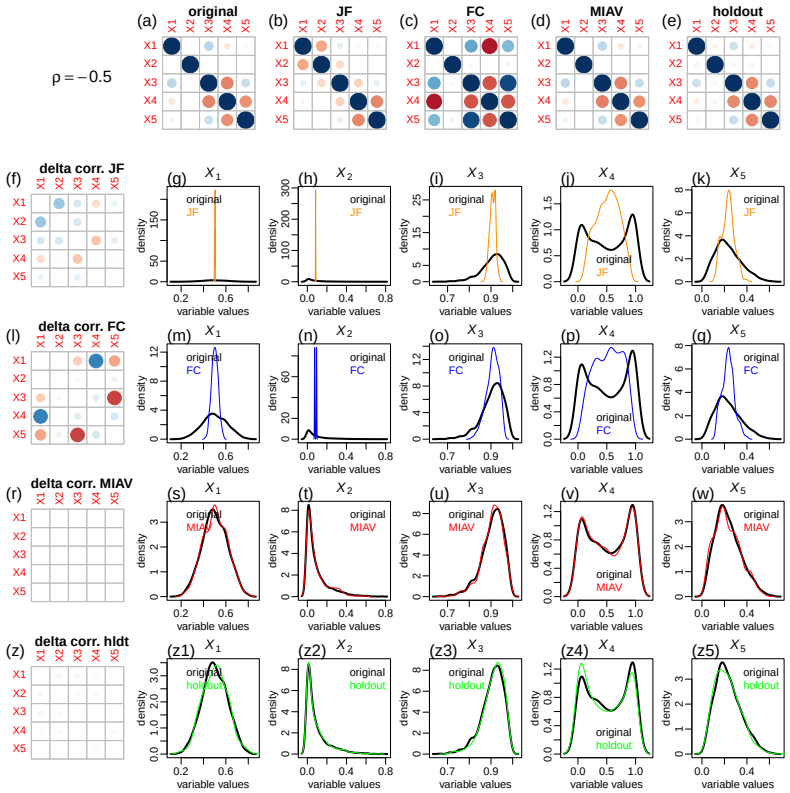

Figure 7: Analogous comparisons as in Figure 5, but for data simulated with $\rho = -0.5$.

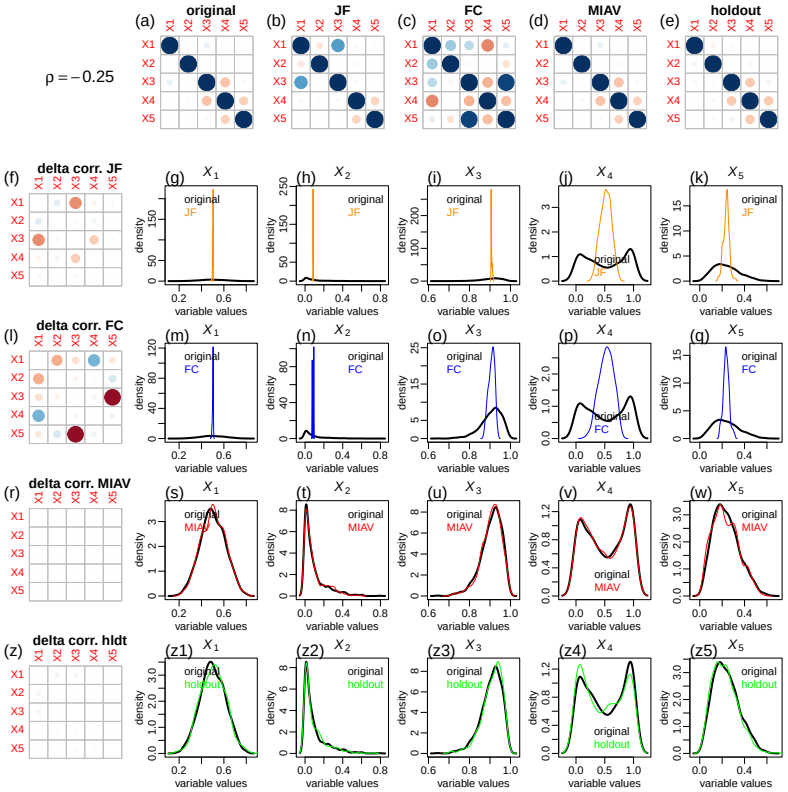

Figure 8: Analogous comparisons as in Figure 5, but for data simulated with $\rho = -0.25$.

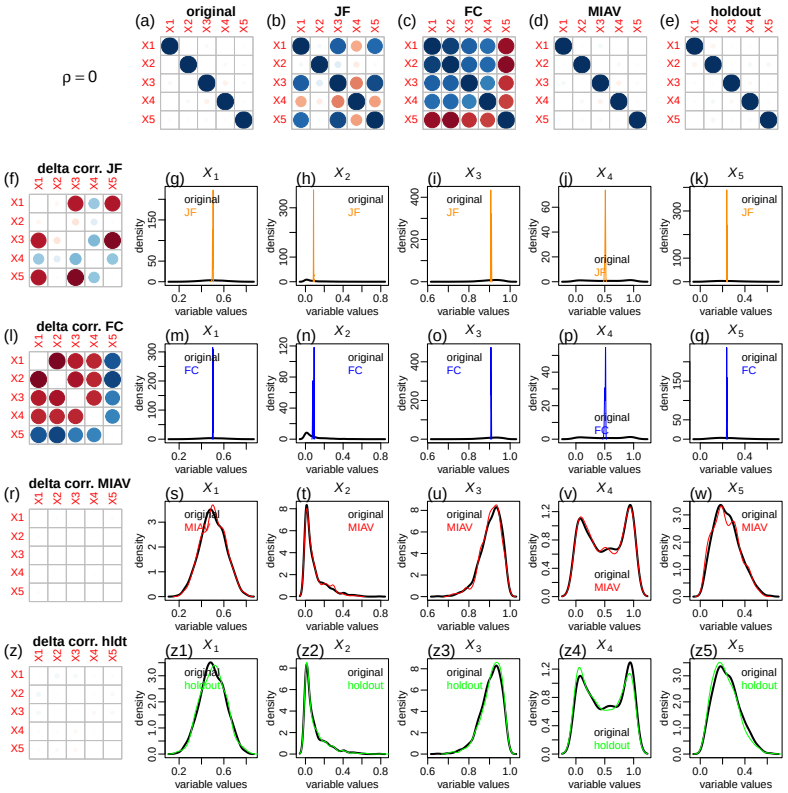

Figure 9: Analogous comparisons as in Figure 5, but for data simulated with $\rho = 0$.

E Additional direct synthetic data generation strategies

As described in Algorithms 2 and 4 our implementations of the JF and FC strategies use the real test data when performing the in-context queries. In the case of the JF strategy, an alternative approach would be to use the synthetic data generated in the previous steps when we query the model in the current generation step. That is, instead of generating the data according to equation 2 we generate it according to,

$$\prod_{j=1}^{p} q_\theta(\mathbf{x}_j^{ts} \mid \hat{\mathbf{X}}_{<j}^{ts}, \mathbf{X}_{<j}^{tr}, \mathbf{x}_j^{tr}) \,, \tag{10}$$

where $\hat{\mathbf{X}}_{<j}^{ts}$ represents the predictions (i.e., the synthetic data) generated in the previous $j-1$ generation steps. We denote this strategy as the "updated joint factorization" (or UJF for short), and implement it in Algorithms 8 and 9. Note that Algorithm 8 implements two versions of the UJF approach. The first, simply takes a bootstrap sample from the original X_1 data as the "synthetic version" of X_1. The second, uses the random noise variable X_0 to generate X_1 (similarly to the JF strategy). Intuitively, we would expect UJF to under-perform in comparison with the JF strategy, since any drifts in the distribution of the synthetic data generated in the previous steps would propagate to the later ones.

Algorithm 8 ICLwithUpdatedJointFactorizationTabPFN($\mathbf{X}^{tr}$, $\mathbf{X}^{ts}$, $version$ = 1)

Input: training data for ICL, $\mathbf{X}^{tr}$; query data for ICL, $\mathbf{X}^{ts}$; $version$ of the UJF strategy
$n_{tr} \leftarrow$ NumberOfRows($\mathbf{X}^{tr}$) {Obtain number of samples of $\mathbf{X}^{tr}$.}
$n_{ts} \leftarrow$ NumberOfRows($\mathbf{X}^{ts}$) {Obtain number of samples of $\mathbf{X}^{ts}$.}
$p \leftarrow$ NumberOfColumns($\mathbf{X}^{ts}$) {Obtain number of columns of $\mathbf{X}^{ts}$.}
$\mathbf{Z}^{ts} \leftarrow [,]$ {Create empty matrix to store the synthetic data.}
if version $==$ 1 **then**
    $\mathbf{Z}^{ts}[,1] \leftarrow$ BootstrapSample($\mathbf{x}_1^{ts}$)
end if
if version $==$ 2 **then**
    $\mathbf{x}_0^{tr} \leftarrow$ GenerateUniformlyDistributedNoise(n_{tr}) {Draw a sample of size n_{tr} from a uniform distribution.}
    $\mathbf{x}_0^{ts} \leftarrow$ GenerateUniformlyDistributedNoise(n_{ts}) {Draw a sample of size n_{ts} from a uniform distribution.}
    $\mathbf{Z}^{ts}[,1] \leftarrow$ GeneratePredictionUsingTabPFN($\mathbf{x}_0^{ts}, \mathbf{x}_0^{tr}, \mathbf{x}_1^{tr}$) {Predict $\mathbf{x}_1^{ts}$ using $\mathbf{x}_0^{tr}$ and $\mathbf{x}_1^{tr}$ as context, and $\mathbf{x}_0^{ts}$ as query. The prediction can be from a regression or classification TabPFN model, depending on whether $\mathbf{x}_1^{tr}$ is continuous or categorical.}
end if
for $j = 2$ **to** p **do**
    $\mathbf{X}_{<j}^{tr} \leftarrow \mathbf{X}^{tr}[,1:(j-1)]$ {Select the first $j-1$ columns of $\mathbf{X}^{tr}$.}
    $\mathbf{Z}_{<j}^{ts} \leftarrow \mathbf{Z}^{ts}[,1:(j-1)]$ {Select the first $j-1$ columns of $\mathbf{Z}^{ts}$.}
    $\mathbf{Z}^{ts}[,j] \leftarrow$ GeneratePredictionUsingTabPFN($\mathbf{Z}_{<j}^{ts}, \mathbf{X}_{<j}^{tr}, \mathbf{x}_j^{tr}$) {Predict $\mathbf{x}_j^{ts}$ using $\mathbf{X}_{<j}^{tr}$ and $\mathbf{x}_j^{tr}$ as context, and $\mathbf{Z}_{<j}^{ts}$ as query. The prediction can be from a regression or classification TabPFN model, depending on whether $\mathbf{x}_j^{tr}$ is continuous or categorical.}
end for
Output: synthetic data $\mathbf{Z}^{ts}$

Algorithm 9 UpdatedJointFactorizationTabPFNGenerator($\mathbf{X}$)

Input: the original data, $\mathbf{X}$
$\mathbf{X}_1, \mathbf{X}_2 \leftarrow$ DataSplit($\mathbf{X}$) {Split the original data $\mathbf{X}$ into two subsets, $\mathbf{X}_1$ and $\mathbf{X}_2$.}
$\mathbf{Z}_1 \leftarrow$ ICLwithUpdatedJointFactorizationTabPFN($\mathbf{X}^{tr} = \mathbf{X}_2, \mathbf{X}^{ts} = \mathbf{X}_1$) {Generate a synthetic data copy of $\mathbf{X}_1$ using Algorithm 8.}
$\mathbf{Z}_2 \leftarrow$ ICLwithUpdatedJointFactorizationTabPFN($\mathbf{X}^{tr} = \mathbf{X}_1, \mathbf{X}^{ts} = \mathbf{X}_2$) {Generate a synthetic data copy of $\mathbf{X}_2$ using Algorithm 8.}
$\mathbf{Z} \leftarrow$ Concatenate($\mathbf{Z}_1, \mathbf{Z}_2$) {Concatenate the synthetic datasets $\mathbf{Z}_1$ and $\mathbf{Z}_2$.}
Output: synthetic data $\mathbf{Z}$

This intuition is confirmed in Figure 10, which shows the application of version 1 of the UJF strategy in purple (panels f to j) and of version 2 (panels l to p) in cyan. For comparison, the plot also include

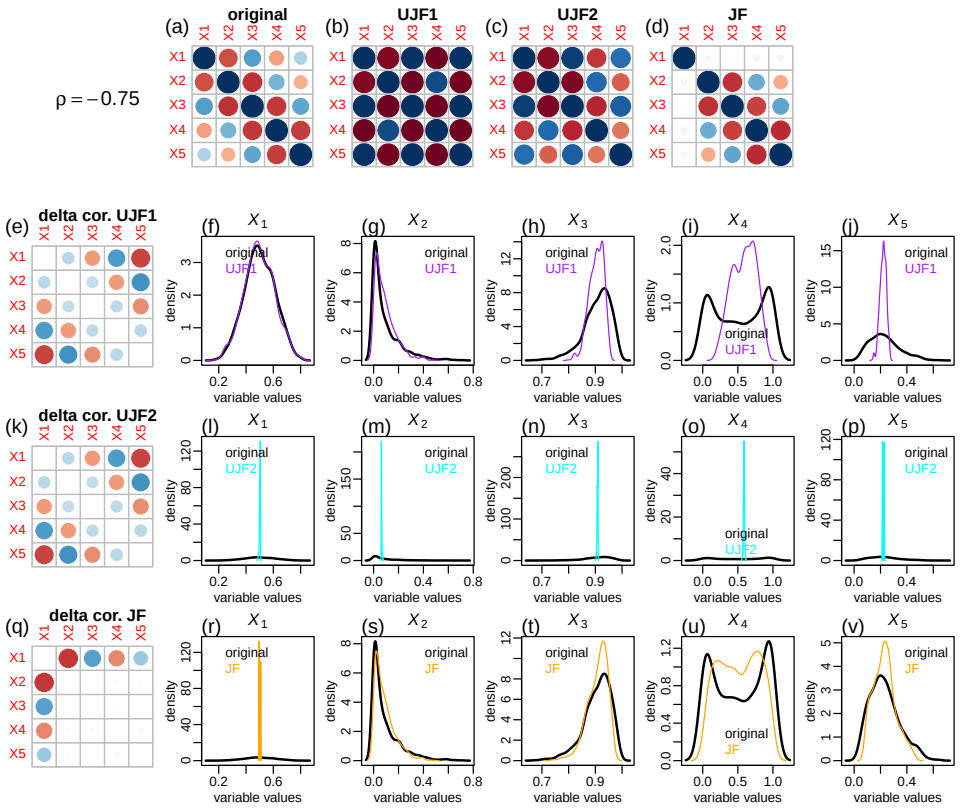

Figure 10: Comparison of JF and the UJF approaches.

the results from the JF approach (panels r to v). Even in version 1, where the algorithm samples the first variable from the correct distribution, the distributions of the synthetic data still ends drifting away from the original data distributions at later steps of the data generation approach. (Note that UJF1 performs considerably worse than the JF approach for variables X_3, X_4, and X_5.) In version 2 (panels l to p) the performance is even worse because the quality of the X_1 data generated in the first step is already very poor.

F Additional algorithms

Algorithm 10 NumericRankEncodingOfCategoticalVariables($\mathbf{x}_j$)

Input: categorical variable data $\mathbf{x}_j$
$levels \leftarrow$ ExtractLevels($\mathbf{x}_j$) {Obtain levels of variable $\mathbf{x}_j$.}
$tb \leftarrow$ Table($\mathbf{x}_j[, i]$) {Obtain level counts of variable $\mathbf{x}_j$.}
$cumcounts \leftarrow$ CumulativeSum(tb) {Compute cumulative counts from the table counts.}
$numlevels \leftarrow$ Length($levels$) {Obtain number of levels.}
$\mathbf{r} \leftarrow []$ {Create empty vector to store the numeric rank encodings.}
for $k = 1$ **to** $numlevels$ **do**
    $idx \leftarrow$ Which($\mathbf{x}_j == levels[k]$) {Obtain the indexes of the records for which $\mathbf{x}_j$ equals level k.}
    $lowerbound \leftarrow cumcounts[k] + 1$ {Compute the lower bound for the numerical encoding of level k of $\mathbf{x}_j$.}
    $upperbound \leftarrow cumcounts[k + 1]$ {Compute the upper bound for the numerical encoding of level k of $\mathbf{x}_j$.}
    $nrseq \leftarrow$ Sequence($lowerbound, upperbound$) {Create a sequence of numeric rank values starting at lower-bound and ending at upper-bound.}
    $\mathbf{r}[idx] \leftarrow$ RandomPermutation($nrseq$) {Assign randomly shuffled numeric rank values to the positions of $\mathbf{x}_j$ corresponding to level k.}
end for
Output: numerical rank encoding $\mathbf{r}$

G Proof of Theorem 1

Consider variables X_j, Y, and M_j, where Y is a placeholder notation for any variable other than X_j or M_j (Y could, for example, be any other $X_{j'}$ or $M_{j'}$ for $j' \neq j$). Variables X_j and Y might be continuous or categorical. Variable M_j is always continuous by construction. To prove the result in a non-parametric setting, we treat any continuous variable as a categorical variable with n distinct levels (where n represents the number of samples in the data). We also assume the independent samples setting, where the rows of dataset $\mathbf{X}$ are independent.

G.1 Proof of statement 1, when X_j and Y are continuous

Proof. We consider first the case where X_j and Y are originally continuous variables. In this case, all variables are treated as categorical variables with n levels, where $i = 1, \ldots, n$, $l = 1, \ldots, n$, and $k = 1, \ldots, n$ indexes the categorical levels of variables X_j, Y, and M_j, respectively.

The conditional mutual information (CMI) of X_j and Y given M_j is defined as,

$$\begin{aligned} &I(X_j; Y|M_j) = \\ &= \sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n} P(X_j = i, Y = l, M_j = k) \log\left(\frac{P(X_j = i, Y = l \mid M_j = k)}{P(X_j = i \mid M_j = k)\, P(Y = l \mid M_j = k)}\right) \\ &= \sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n} P(X_j = i, Y = l, M_j = k) \log\left(\frac{P(X_j = i, Y = l, M_j = k)\, P(M_j = k)}{P(X_j = i, M_j = k)\, P(Y = l, M_j = k)}\right), \end{aligned} \tag{11}$$

where the joint probabilities are defined from the joint frequency counts in a finite population. For instance, $P(X_j = i, Y = l, M_j = k) = f(X_j = i, Y = l, M_j = k)/n$ where $f(X_j = i, Y = l, M_j = k)$ corresponds to the number of instances in a population of size n for which $X_j = i$ and $Y = l$ and $M_j = k$. Note that because all three variables have n distinct levels, these frequencies will be either 1 or 0 depending on whether the combination of values is observed or not in the dataset.

To avoid division by 0, we consider the smoothed version of the CMI where an arbitrarily small constant ϵ is added to the frequency counts of each cell in the contingency tables defining these probabilities. Namely,

$$f^{\epsilon}(X_j = i, Y = l, M_j = k) = f(X_j = i, Y = l, M_j = k) + \epsilon, \tag{12}$$

and $P^{\epsilon}(X_j = i, Y = l, M_j = k)$ is obtained by normalization as,

$$\begin{aligned} P^{\epsilon}(X_j = i, Y = l, M_j = k) &= \frac{f(X_j = i, Y = l, M_j = k) + \epsilon}{\sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n}(f(X_j = i, Y = l, M_j = k) + \epsilon)} \\ &= \frac{f(X_j = i, Y = l, M_j = k) + \epsilon}{\sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n} f(X_j = i, Y = l, M_j = k) + \sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n}\epsilon} \\ &= \frac{f(X_j = i, Y = l, M_j = k) + \epsilon}{n + n^3\,\epsilon}\,. \end{aligned} \tag{13}$$

Similarly, we have that,

$$P^{\epsilon}(X_j = i, M_j = k) = \frac{f(X_j = i, M_j = k) + \epsilon}{n + n^2\,\epsilon}\,, \tag{14}$$

$$P^{\epsilon}(Y = l, M_j = k) = \frac{f(Y = l, M_j = k) + \epsilon}{n + n^2\,\epsilon}\,, \tag{15}$$

$$P^{\epsilon}(M_j = k) = \frac{f(M_j = k) + \epsilon}{n + n\,\epsilon} = \frac{1 + \epsilon}{n(1 + \epsilon)} = \frac{1}{n}\,. \tag{16}$$

Now, because the rank-matching procedure used in the construction of M_j implies a perfect monotonic relation between the values of X_j and M_j we have that the discretized versions of X_j and M_j will be identical (see Table 2 for an illustrative toy example). This correspondence between X_j and M_j implies that these variables cannot concomitantly assume different values at the same time (i.e., the

joint frequency $f(X_j = i, M_j = k)$ will always be 0 if $i \neq k$). It also implies that the joint counts $f(X_j = i, M_j = k)$ will always be equal to the marginal counts $f(M_j = k)$ when $i = k$. Hence, we have that,

$$f(X_j = i, M_j = k) = \begin{cases} f(M_j = k) = 1\,, & \text{if } i = k \\ 0\,, & \text{if } i \neq k \end{cases}, \tag{17}$$

and we can re-express equation (14) as,

$$P^\epsilon(X_j = i, M_j = k) = \begin{cases} \dfrac{f(M_j = k) + \epsilon}{n + n^2\epsilon} = \dfrac{1 + \epsilon}{n + n^2\epsilon}\,, & \text{if } i = k \\ \dfrac{\epsilon}{n + n^2\,\epsilon}\,, & \text{if } i \neq k \end{cases}. \tag{18}$$

Similarly, we have that,

$$f(X_j = i, Y = l, M_j = k) = \begin{cases} f(Y = l, M_j = k)\,, & \text{if } i = k \\ 0\,, & \text{if } i \neq k \end{cases}, \tag{19}$$

and we can re-express equation (13) as,

$$P^\epsilon(X_j = i, Y = l, M_j = k) = \begin{cases} \dfrac{f(Y = l, M_j = k) + \epsilon}{n + n^3\epsilon}\,, & \text{if } i = k \\ \dfrac{\epsilon}{n + n^3\,\epsilon}\,, & \text{if } i \neq k \end{cases}. \tag{20}$$

Re-expressing the conditional mutual information in terms of the smoothed probabilities and separating the summation over the cases where $k = i$ from the cases where $k \neq i$ we have,

$$\begin{aligned}
&I^\epsilon(X_j; Y|M_j) = \qquad (21)\\
&= \sum_{i=1}^{n}\sum_{l=1}^{n} P^\epsilon(X_j = i, Y = l, M_j = i) \log\left(\frac{P^\epsilon(X_j = i, Y = l, M_j = i)\, P^\epsilon(M_j = i)}{P^\epsilon(X_j = i, M_j = i)\, P^\epsilon(Y = l, M_j = i)}\right) + \\
&\quad + \sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k\neq i} P^\epsilon(X_j = i, Y = l, M_j = k) \log\left(\frac{P^\epsilon(X_j = i, Y = l, M_j = k)\, P^\epsilon(M_j = k)}{P^\epsilon(X_j = i, M_j = k)\, P^\epsilon(Y = l, M_j = k)}\right) \\
&= \sum_{i=1}^{n}\sum_{l=1}^{n} \frac{f(Y = l, M_j = i) + \epsilon}{n + n^3\epsilon} \log\left(\frac{\left(\dfrac{f(Y = l, M_j = i) + \epsilon}{n + n^3\,\epsilon}\right)\left(\dfrac{f(M_j = i) + \epsilon}{n + n\,\epsilon}\right)}{\left(\dfrac{f(M_j = i) + \epsilon}{n + n^2\epsilon}\right)\left(\dfrac{f(Y = l, M_j = i) + \epsilon}{n + n^2\epsilon}\right)}\right) + \\
&\quad + \sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k\neq i} \frac{\epsilon}{n + n^3\,\epsilon} \log\left(\frac{\left(\dfrac{\epsilon}{n + n^3\,\epsilon}\right)\left(\dfrac{f(M_j = k) + \epsilon}{n + n\,\epsilon}\right)}{\left(\dfrac{\epsilon}{n + n^2\,\epsilon}\right)\left(\dfrac{f(Y = l, M_j = k) + \epsilon}{n + n^2\,\epsilon}\right)}\right) \\
&= \sum_{i=1}^{n}\sum_{l=1}^{n} \frac{f(Y = l, M_j = i) + \epsilon}{n + n^3\epsilon} \log\left(\frac{\left(n + n^2\epsilon\right)\left(n + n^2\epsilon\right)}{\left(n + n^3\epsilon\right)\left(n + n\,\epsilon\right)}\right) + \\
&\quad + \sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k\neq i} \frac{\epsilon}{n + n^3\,\epsilon} \log\left(\frac{(n + n^2\,\epsilon)(n + n^2\,\epsilon)(f(M_j = k) + \epsilon)}{(n + n^3\,\epsilon)(n + n\,\epsilon)(f(Y = l, M_j = k) + \epsilon)}\right)
\end{aligned}$$

Taking the limit as $\epsilon \to 0$ we have that $I^\epsilon(X_j; Y|M_j)$ converges to,

$$\sum_{i=1}^{n}\sum_{l=1}^{n} \frac{f(Y = l, M_j = i)}{n} \log(1) + \sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k\neq i} 0 \log\left(\frac{f(M_j = k)}{f(Y = l, M_j = k)}\right) = 0\,. \tag{22}$$

Observe that the term $0\log(f(M_j = k)/f(Y = l, M_j = k))$ in the above equation equals 0 because it can be rewritten as $0\log f(M_j = k) - 0\log f(Y = l, M_j = k)$ and $0\log f(M_j = k) = 0\log(1/n) = 0$ and $0\log f(Y = l, M_j = k)$ equals $0\log 1 = 0$ if $l = k$ or $0\log 0 = 0$ if $l \neq k$. (Note that in this last case, we are adopting the convention that $0\log 0 = 0$, as customary in information theory.)

□

G.2 Proof of statement 1, when X_j and Y are categorical

Proof. We now consider the case where X_j and Y are categorical variables. Let b_x and b_y represent the number of levels of X_j and Y, and let $i = 1, \ldots, b_x$, $l = 1, \ldots, b_y$, and $k = 1, \ldots, n$ represent the indexes of the levels of variables X_j, Y, and M_j, respectively.

Now, the CMI of X_j and Y conditional of M_j is given by,

$$\begin{aligned} &I(X_j; Y|M_j) = \\ &= \sum_{i=1}^{b_x}\sum_{l=1}^{b_y}\sum_{k=1}^{n} P(X_j = i, Y = l, M_j = k) \log\left(\frac{P(X_j = i, Y = l \mid M_j = k)}{P(X_j = i \mid M_j = k)\, P(Y = l \mid M_j = k)}\right) \\ &= \sum_{i=1}^{b_x}\sum_{l=1}^{b_y}\sum_{k=1}^{n} P(X_j = i, Y = l, M_j = k) \log\left(\frac{P(X_j = i, Y = l, M_j = k)\, P(Z_j = k)}{P(X_j = i, M_j = k)\, P(Y = l, M_j = k)}\right) . \end{aligned} \tag{23}$$

To avoid division by 0, we again consider the smoothed version of the CMI with smoothed probabilities given by,

$$P^{\epsilon}(X_j = i, Y = l, M_j = k) = \frac{f(X_j = i, Y = l, M_j = k) + \epsilon}{n + b_x\, b_y\, n\, \epsilon} . \tag{24}$$

$$P^{\epsilon}(X_j = i, M_j = k) = \frac{f(X_j = i, M_j = k) + \epsilon}{n + b_x\, n\, \epsilon} , \tag{25}$$

$$P^{\epsilon}(Y = l, M_j = k) = \frac{f(Y = l, M_j = k) + \epsilon}{n + b_y\, n\, \epsilon} , \tag{26}$$

$$P^{\epsilon}(M_j = k) = \frac{f(M_j = k) + \epsilon}{n + n\, \epsilon} = \frac{1 + \epsilon}{n(1 + \epsilon)} = \frac{1}{n} . \tag{27}$$

By construction, when X_j is categorical, we have that the following relations hold between X_j and the discretized M_j,

$$f(X_j = i, M_j = k) = \begin{cases} f(M_j = k) , & \text{if } k \in I_i \\ 0 , & \text{if } k \notin I_i \end{cases} , \tag{28}$$

$$f(X_j = i, Y = l, M_j = k) = \begin{cases} f(Y = l, M_j = k) , & \text{if } k \in I_i \\ 0 , & \text{if } k \notin I_i \end{cases} , \tag{29}$$

where I_i represents the set of indexes of the discretized M_j variable for which $X_j = i$, and $\mathbb{1}\{k \in I_i\}$ represents the indicator function assuming value 1 when k belongs to I_i, and 0 otherwise. (Table 3 provides an illustrative example explaining the above relations.)

Using equations 28 and 29 we can re-express equations (25) and (24) as,

$$P^{\epsilon}(X_j = i, M_j = k) = \begin{cases} \dfrac{f(M_j = k) + \epsilon}{n + b_x\, n\, \epsilon} , & \text{if } k \in I_i \\ \dfrac{\epsilon}{n + b_x\, n\, \epsilon} , & \text{if } k \notin I_i \end{cases} , \tag{30}$$

and,

$$P^{\epsilon}(X_j = i, Y = l, M_j = k) = \begin{cases} \dfrac{f(Y = l, M_j = k) + \epsilon}{n + b_x\, b_y\, n\, \epsilon} , & \text{if } k \in I_i \\ \dfrac{\epsilon}{n + b_x\, b_y\, n\, \epsilon} , & \text{if } k \notin I_i \end{cases} . \tag{31}$$

Re-expressing the conditional mutual information in terms of the smoothed probabilities and separating the summation over the cases where $k \in I_i$ from the cases where $k \notin I_i$ we have,

$$I^\epsilon(X_j;Y|M_j) = \tag{32}$$

$$= \sum_{i=1}^{b_x}\sum_{l=1}^{b_y}\sum_{k\in I_i} P^\epsilon(X_j=i,Y=l,M_j=k)\log\left(\frac{P^\epsilon(X_j=i,Y=l,M_j=k)\,P^\epsilon(M_j=k)}{P^\epsilon(X_j=i,M_j=k)\,P^\epsilon(Y=l,M_j=k)}\right) +$$

$$+ \sum_{i=1}^{b_x}\sum_{l=1}^{b_y}\sum_{k\notin I_i} P^\epsilon(X_j=i,Y=l,M_j=k)\log\left(\frac{P^\epsilon(X_j=i,Y=l,M_j=k)\,P^\epsilon(M_j=k)}{P^\epsilon(X_j=i,M_j=k)\,P^\epsilon(Y=l,M_j=k)}\right)$$

$$= \sum_{i=1}^{b_x}\sum_{l=1}^{b_y}\sum_{k\in I_i} \frac{f(Y=l,M_j=k)+\epsilon}{n+b_x\,b_y\,n\,\epsilon}\log\left(\frac{\left(\frac{f(Y=l,M_j=k)+\epsilon}{n+b_x\,b_y\,n\,\epsilon}\right)\left(\frac{f(M_j=k)+\epsilon}{n+n\,\epsilon}\right)}{\left(\frac{f(M_j=k)+\epsilon}{n+b_x\,n\,\epsilon}\right)\left(\frac{f(Y=l,M_j=k)+\epsilon}{n+b_y\,n\,\epsilon}\right)}\right) +$$

$$+ \sum_{i=1}^{b_x}\sum_{l=1}^{b_y}\sum_{k\notin I_i} \frac{\epsilon}{n+b_x\,b_y\,n\,\epsilon}\log\left(\frac{\left(\frac{\epsilon}{n+b_x\,b_y\,n\,\epsilon}\right)\left(\frac{f(M_j=k)+\epsilon}{n+n\,\epsilon}\right)}{\left(\frac{\epsilon}{n+b_x\,n\,\epsilon}\right)\left(\frac{f(Y=l,M_j=k)+\epsilon}{n+b_y\,n\,\epsilon}\right)}\right)$$

$$= \sum_{i=1}^{b_x}\sum_{l=1}^{b_y}\sum_{k\in I_i} \frac{f(Y=l,M_j=k)+\epsilon}{n+b_x\,b_y\,n\,\epsilon}\log\left(\frac{(n+b_x\,n\,\epsilon)(n+b_y\,n\,\epsilon)}{(n+b_x\,b_y\,n\,\epsilon)(n+n\,\epsilon)}\right) +$$

$$+ \sum_{i=1}^{b_x}\sum_{l=1}^{b_y}\sum_{k\notin I_i} \frac{\epsilon}{n+b_x\,b_y\,n\,\epsilon}\log\left(\frac{(n+b_x\,n\,\epsilon)(n+b_y\,n\,\epsilon)(f(M_j=k)+\epsilon)}{(n+b_x\,b_y\,n\,\epsilon)(n+n\,\epsilon)(f(Y=l,M_j=k)+\epsilon)}\right)$$

Taking the limit as $\epsilon \to 0$ we have that $I^\epsilon(X_j;Y|M_j)$ converges to,

$$\sum_{i=1}^{b_x}\sum_{l=1}^{b_y}\sum_{k\in I_i} \frac{f(Y=l,M_j=i)}{n}\log(1) + \sum_{i=1}^{b_x}\sum_{l=1}^{b_y}\sum_{k\notin I_i} 0\log\left(\frac{f(M_j=k)}{f(Y=l,M_j=k)}\right) = 0\,. \tag{33}$$

□

G.3 Proof of statement 1, in the remaining cases

Proof. In the case where X_j is continuous and Y is categorical, a similar argument to the proof in the case that both variables are continuous shows that,

$$I^\epsilon(X_j;Y|M_j) = \tag{34}$$

$$= \sum_{i=1}^{n}\sum_{l=1}^{b_y} \frac{f(Y=l,M_j=i)+\epsilon}{n+b_y\,n^2\,\epsilon}\log\left(\frac{\left(n+n^2\,\epsilon\right)\left(n+b_y\,n\,\epsilon\right)}{\left(n+b_y\,n^2\,\epsilon\right)\left(n+n\,\epsilon\right)}\right) +$$

$$+ \sum_{i=1}^{n}\sum_{l=1}^{b_y}\sum_{k\neq i} \frac{\epsilon}{n+b_y\,n^2\,\epsilon}\log\left(\frac{(n+n^2\,\epsilon)(n+b_y\,n\,\epsilon)(f(M_j=k)+\epsilon)}{(n+b_y\,n^2\,\epsilon)(n+n\,\epsilon)(f(Y=l,M_j=k)+\epsilon)}\right)\,,$$

which again converges to 0 as $\epsilon \to 0$.

In the case that X_j is categorical and Y is continuous, a similar argument to the proof in the case that both variables are categorical shows that,

$$I^\epsilon(X_j;Y|M_j) = \tag{35}$$

$$= \sum_{i=1}^{b_x}\sum_{l=1}^{n}\sum_{k\in I_i} \frac{f(Y=l,M_j=k)+\epsilon}{n+b_x\,n^2\,\epsilon}\log\left(\frac{(n+b_x\,n\,\epsilon)(n+n^2\,\epsilon)}{(n+b_x\,n^2\,\epsilon)(n+n\,\epsilon)}\right) +$$

$$+ \sum_{i=1}^{b_x}\sum_{l=1}^{n}\sum_{k\notin I_i} \frac{\epsilon}{n+b_x\,n^2\,\epsilon}\log\left(\frac{(n+b_x\,n\,\epsilon)(n+n^2\,\epsilon)(f(M_j=k)+\epsilon)}{(n+b_x\,n^2\,\epsilon)(n+n\,\epsilon)(f(Y=l,M_j=k)+\epsilon)}\right)$$

which again converges to 0 as $\epsilon \to 0$.

□

G.4 Proof of statement 2

Proof. By definition, the conditional entropy of X_j given M_j is given by,

$$\begin{aligned} H(X_j \mid M_j) &= \sum_i \sum_k P(X_j = i, M_j = k) \log \left(\frac{P(X_j = i, M_j = k)}{P(M_j = k)} \right) \\ &= \sum_i \sum_k P(X_j = i \mid M_j = k) P(M_j = k) \log P(X_j = i \mid M_j = k) \end{aligned} \tag{36}$$

In the case where X_j is continuous (and discretized into n classes) we have that equation (36) reduces to,

$$\begin{aligned} H(X_j \mid M_j) &= \sum_{i=1}^{n} P(X_j = i \mid M_j = i) P(M_j = i) \log P(X_j = i \mid M_j = i) \\ &+ \sum_{i=1}^{n} \sum_{k \neq i} P(X_j = i \mid M_j = k) P(M_j = k) \log P(X_j = i \mid M_j = k) \, , \end{aligned} \tag{37}$$

after we split the summation over k into the cases where $k = i$ and $k \neq i$. Since, by construction, we have that,

$$P(X_j = i \mid M_j = k) = \begin{cases} 1, & \text{if } k = i \\ 0 & \text{if } k \neq i \end{cases} \, , \tag{38}$$

if follows that,

$$H(X_j \mid M_j) = \sum_{i=1}^{n} P(M_j = i) \log 1 + \sum_{i=1}^{n} \sum_{k \neq i} 0 P(M_j = k) \log 0 = 0 \tag{39}$$

where we take $0 \log 0$ to be defined as 0 (as its is customary in information theory).

Similarly, in the case where X_j is categorical (with b_x classes) equation (36) reduces to,

$$\begin{aligned} H(X_j \mid M_j) &= \sum_{i=1}^{b_x} \sum_{k \in I_i} P(X_j = i \mid M_j = k) P(M_j = k) \log P(X_j = i \mid M_j = k) \\ &+ \sum_{i=1}^{b_x} \sum_{k \notin I_i} P(X_j = i \mid M_j = k) P(M_j = k) \log P(X_j = i \mid M_j = k) \, , \end{aligned} \tag{40}$$

after we split the summation over k into the cases where $k \in I_i$ and $k \notin I_i$. Since, by construction, we also have that in the categorical case,

$$P(X_j = i \mid M_j = k) = \begin{cases} 1, & \text{if } k \in I_i \\ 0 & \text{if } k \notin I_i \end{cases} \, , \tag{41}$$

if follows again that,

$$H(X_j \mid M_j) = \sum_{i=1}^{b_x} \sum_{k \in I_i} P(M_j = i) \log 1 + \sum_{i=1}^{b_x} \sum_{k \notin I_i} 0 P(M_j = k) \log 0 = 0 \, . \tag{42}$$

□

Table 2: Illustrative toy example showing the generation of M_j using Algorithm 1 when X_j is continuous. The sample $\mathbf{x}_j = (0.92, -1.29, 0.34, -0.93, 0.71, 0.39, 0.65, 0.85, 0.84, 1.38, -0.32)^t$ (with $n = 11$) represents observations from X_j. Rows (i) and (ii) display the original values of X_j and their corresponding ranks. Row (iii) shows the generated variable M_j, which is uniformly distributed and rank-matched to X_j (i.e., the ordering of M_j is identical to that of X_j). Row (iv) confirms this by showing that the ranks of M_j match those of row (ii). Because of this perfect monotonic correspondence, the discretized versions of X_j and M_j are identical for any chosen number of bins. Rows (v) and (vi) illustrate this fact by presenting the discretizations of X_j and M_j based on n bins.

(I):	ORIGINAL X_j	0.92	-1.29	0.34	-0.93	0.71	0.39	0.65	0.85	0.84	1.38	-0.32
(II):	RANKS OF X_j	10	1	4	2	7	5	6	9	8	11	3
(III):	ORIGINAL M_j	0.69	0.12	0.41	0.20	0.61	0.44	0.50	0.67	0.62	0.74	0.39
(IV):	RANKS OF M_j	10	1	4	2	7	5	6	9	8	11	3
(V):	X_j (DISCRETIZED)	10	1	4	2	7	5	6	9	8	11	3
(VI):	M_j (DISCRETIZED)	10	1	4	2	7	5	6	9	8	11	3

Table 3: Toy illustrative example of the generation of M_j using Algorithms 1 and 10 in the case where X_j is categorical. Let $\mathbf{x}_j = (A, A, A, A, B, B, B, C, C, C, C)^t$ represent a sample of size $n = 11$ from the categorical variable X_j, which has 3 distinct levels, $\{A, B, C\}$. Let $i = 1, 2, 3$ represent the indexes of the levels of X_j. Row (i) shows the data in its original format, while row (ii) shows the same data in terms of the level indexes representation, where level A is indexed as 1, level B as 2, and level C as 3. Row (iii) shows the output, $\mathbf{r}$, of Algorithm 10 which generates a numeric rank encoding of the categorical variable X_j. (It adopts the arbitrary order $A < B < C$ to the categorical levels of X_j and starting with class A, it assigns ranks 1, 2, 3, and 4 (in random order) to the four tied elements A in $\mathbf{x}_j$. For class B, it assigns ranks 5, 6, and 7 (in random order) to the three tied elements B in $\mathbf{x}_j$. Finally, for class C it assigns ranks 8 to 11 (in random order) to the four tied elements C.) Row (iv) shows the output of Algorithm 1. It first sample and sort 11 values from a uniform distribution in the [0, 1] range (line 2) given, in this example, by (0.12, 0.20, 0.39, 0.41, 0.44, 0.50, 0.61, 0.62, 0.67, 0.69, 0.74) and then rank match this sorted random noise vector to the numeric rank encodings $\mathbf{r}$ show in line (iii). The result is a random noise vector with identical ranks as $\mathbf{r}$. Row (v) shows M_j after discretization into $n = 11$ levels (in terms of level indexes representation). Clearly, when we discretize M_j into n levels we recover the numerical rank encoding in row (iii). Now, let I_i represent the indexes of the M_j values for which $X_j = i$. In this example $I_1 = \{1, 2, 3, 4\}$, $I_2 = \{5, 6, 7\}$, and $I_3 = \{8, 9, 10, 11\}$. Note that, by construction, whenever M_j equals 1, 2, 3, or 4, the value of X_j will be 1. This implies that $f(X_j = 1, M_j = k) = f(M_j = k) = 1$ if $k \in \{1, 2, 3, 4\}$, and $f(X_j = 1, M_j = k) = 0$ if $k \notin \{1, 2, 3, 4\}$. Similarly, $f(X_j = 2, M_j = k) = f(M_j = k) = 1$ if $k \in \{5, 6, 7\}$, and $f(X_j = 2, M_j = k) = 0$ if $k \notin \{5, 6, 7\}$, and so on for other values of i. Clearly, this result holds in general, as stated in equation 28. A similar argument justifies equation 29.

(I):	ORIGINAL X_j	A	A	A	A	B	B	B	C	C	C	C
(II):	X_j LEVEL INDEXES	1	1	1	1	2	2	2	3	3	3	3
(III):	NUMERIC RANK ENCODING	3	1	4	2	7	5	6	9	8	11	10
(IV):	M_j (CONTINUOUS)	0.39	0.12	0.41	0.20	0.61	0.44	0.50	0.67	0.62	0.74	0.69
(V):	M_j (DISCRETIZED)	3	1	4	2	7	5	6	9	8	11	10
(VI):	I_i MAPPING	$I_1 = \{1, 2, 3, 4\}$,				$I_2 = \{5, 6, 7\}$,			$I_3 = \{8, 9, 10, 11\}$			

H Extended experiments section

H.1 Overview of data splitting into original, holdout, training, and test sets

Our experiments are performed as follows. Each dataset $\mathbf{D}$ is first split into two datasets $\mathbf{X}$ and $\mathbf{X}^h$, denoted as the "original" and the "holdout" datasets, respectively. In our evaluations, the original dataset, $\mathbf{X}$, plays the role of the "real" data, from which we generate a synthetic data copy. (We denote it as the "original" data because $\mathbf{D}$ might be a real-world dataset or a simulated dataset in our experiments.) The holdout dataset, $\mathbf{X}^h$, is used to "estimate" the performance of an ideal data generator capable of generating data from the same distribution as the original data. In our experiments, in addition to evaluating the synthetic datasets using the metrics described in Section H.4 below, we also compute the metric values for the holdout set in order to get a sense about the range of values we would expect to see for the metric in the ideal case of a generator truly able to draw independent samples from the same distribution as the original data.

For the generation of synthetic data, based on ICL using the TabPFN-based strategies described in the main test, namely, JF, FC, and MIAV, we further split the original data, $\mathbf{X}$, into two subsets, $\mathbf{X}_1$ and $\mathbf{X}_2$. As described in Algorithms 3, 5, and 7 in Section B, the synthetic data generation is performed as follows. First, the algorithms generate a synthetic copy of $\mathbf{X}_1$, using $\mathbf{X}_2$ as the training set and $\mathbf{X}_1$ as the test set. Second, the algorithms generate a synthetic copy of $\mathbf{X}_2$, by using $\mathbf{X}_1$ as the training set and $\mathbf{X}_2$ as the test set. Third, the algorithms concatenate the datasets generated in the previous steps to obtain the full synthetic dataset copy of $\mathbf{X}$.

For the generation of synthetic data using the SMOTE baseline, the original data $\mathbf{X}$ is directly fed into the synthesizer with no need for further data splits.

H.2 Simulated data experiments

For the simulated data experiments we evaluate the synthetic data generators over 5 distinct settings spanning different correlation structures among the variables. In all settings, we generate datasets with 5 variables from correlated beta distributions as described in Section C. The correlation structure (Toeplitz) is controlled by a single parameter ρ, and in our experiments we adopt correlation strengths, $|\rho|$, in the range $|\rho| = \{0, 0.25, 0.5, 0.75, 0.95\}$.

For each experimental setting (i.e., $|\rho|$ value) we simulate 10 distinct datasets, $\mathbf{D}$, of size 800, using Toeplitz correlation parameter randomly set to either ρ or $-\rho$ with equal probability, and adopting different shape parameters, a and b, for the beta distributions. For each variable $X_j \sim \text{Beta}(a, b)$, the a and b parameters are randomly sampled from uniform distributions as follows:

- $a \sim U(0.1, 0.9)$, $b \sim U(0.1, 0.9)$, for X_1.
- $a \sim U(0.1, 0.9)$, $b \sim U(1, 10)$, for X_2.
- $a \sim U(10, 50)$, $b \sim U(1, 10)$, for X_3.
- $a \sim U(5, 15)$, $b \sim U(5, 15)$, for X_4.
- $a \sim U(1, 10)$, $b \sim U(5, 15)$, for X_5.

Each dataset $\mathbf{D}$ is split into original ($\mathbf{X}$) and holdout ($\mathbf{X}^h$) datasets of size 400. Only the $\mathbf{X}$ datasets are used by the synthetic data generators.

H.3 Real-world data experiments

For the real-world data experiments we selected a subset of the datasets from the OpenML-CC18 benchmark suite (4), which were analyzed in (14). The OpenML-CC18 suite contains 72 datasets but, similarly to (14), we selected datasets with at most 2000 rows (examples), at most 100 columns (features), and containing categorical variables with at most 10 classes. However, because most of the evaluation metrics used in this paper to assess data fidelity and data privacy are tailored to numeric data, we applied the additional filter that the datasets should contain more numerical variables than categorical ones. After applying these filters to the 72 datasets in OpenML-CC18 we were left with the 21 datasets listed in Table 4.

Table 4: Datasets used in the real-world data evaluations. These include all 21 datasets in the OpenML-CC18 benchmark suite with at most 2000 samples, 100 variables, 10 classes per categorical variables, and a larger number of numeric variables than categorical ones. In the first column we assign simplified dataset names (D1 to D21) to the original dataset names. The number of categorical variables is abbreviated as #cat, and the number of classes of the categorical variable with most classes is abbreviated as #class.

NAME	ORIGINAL DATASET NAME	#SAMPLES	#COLUMNS	#CAT	#CLASS	OPENML ID
D1	BALANCE-SCALE	625	5	1	3	11
D2	MFEAT-FOURIER	2000	77	1	10	14
D3	BREAST-W	699	10	1	2	15
D4	MFEAT-KARHUNEN	2000	65	1	10	16
D5	MFEAT-MORPHOLOGICAL	2000	7	1	10	18
D6	MFEAT-ZERNIKE	2000	48	1	10	22
D7	DIABETES	768	9	1	2	37
D8	VEHICLE	846	19	1	4	53
D9	ANALCATDATA-AUTHORSHIP	841	71	1	4	3549
D10	PC4	1458	38	1	2	3902
D11	PC3	1563	38	1	2	3903
D12	KC2	522	22	1	2	3913
D13	PC1	1109	22	1	2	3918
D14	WDBC	569	31	1	2	9946
D15	QSAR-BIODEG	1055	42	1	2	9957
D16	ILPD	583	11	2	2	9971
D17	BANKNOTE-AUTHENTICATION	1372	5	1	2	10093
D18	BLOOD-TRANSFUSION-SERVICE-CENTER	748	5	1	2	10101
D19	MICEPROTEIN	1080	78	1	8	146800
D20	STEEL-PLATES-FAULT	1941	28	1	7	146817
D21	CLIMATE-MODEL-SIMULATION-CRASHES	540	19	1	2	146819

H.4 Evaluation metrics

We evaluated the quality of the synthetic data in terms of data fidelity and data privacy.

Data fidelity was evaluated with respect to the quality of the marginal distributions, quality of the pairwise statistical associations, and quality of the joint probability distribution, according to the following metrics:

- **Average KS-statistic (KS).** This metric is used to evaluate the quality of the synthetic data marginal distributions. It is based on the Kolmogorov-Smirnov two-sample statistical test (KS-test) for the equality of distributions. For each variable it computes the KS-test statistic between the synthetic and original data, and the metric corresponds to the average KS-statistic across all variables. Lower values of this metric indicate better agreement between the synthetic and original marginal distributions.
- **L2 distance between association matrices (L2D).** This metric is used to evaluate how well the synthetic data recovers the pairwise statistical associations observed in the original data. The L2 distance is computed as the average of the squared difference between the elements of the synthetic and original data association matrices. Lower values of this metric indicate better agreement between the synthetic and original data pairwise statistical associations. Since the datasets might include both numerical and categorical variables, we assess pairwise associations as follows: for numerical pairs we use Pearson correlation; for categorical pairs we use the Cramer's V statistic; for numeric/categorical pairs we regress the numeric variable on the categorical one and use the square root of the R^2 statistic as our association measure (this reduces to the absolute correlation coefficient when both variables are numerical).
- **Energy distance (ED).** This metric is used to evaluate how well the joint probability distribution of the synthetic data approximates the joint distribution of the original data. Lower values of this metric indicate better agreement between the synthetic and original joint probability distributions. (Energy distance (31) represents a special case of the maximum mean discrepancy statistic.) Since ED uses Euclidean norms to compare observations, it is is sensitive to scale (i.e., variables with larger numerical ranges dominate the distance). Hence, we first re-scale all variables before computing the ED.

Data privacy was evaluated according to the following metrics:

- **Distance to closest record (DCR).** This represents a record-level privacy metric that measures how similar a synthetic record is to the closest record in the original dataset. It is computed by measuring, for each synthetic record, the minimum distance to any record in the original dataset using Euclidean distance; values close to zero indicate higher disclosure risk because synthetic records are nearly indistinguishable from the original data ones, whereas larger values suggest safer privacy protection by ensuring greater separation between synthetic and real data (this might be achieved at the cost of lower data fidelity, though). Since Euclidean distance is sensitive to scale (i.e., variables with larger numerical ranges dominate the distance), we first re-scale all variables before computing the DCR.
- **Sorted distance-based record linkage (SDBRL).** This metric represents a variant of the Distance-Based Record Linkage (DBRL) metric. The DBRL metric (26; 10) is a widely used method for evaluating re-identification risk of perturbation methods within the Statistical Disclosure Control field (11). It operates by computing the Euclidean distance between each record in the perturbed dataset and all records in the original dataset, designating a perturbed record as 'linked' when its nearest neighbor corresponds to its true original record. The DBRL value is then given by the proportion of perturbed records successfully linked back to their original counterparts. Strictly speaking, the DBRL metric is only intended for evaluating data perturbation methods, since it assumes the existence of a direct mapping between the original and perturbed values (for example, when perturbed data are obtained by adding noise to the original data). In the case of synthetic data, where such a mapping is absent, an approximate correspondence can still be established following the approach of (9) and (5; 6). The idea is to sort the rows of both the original and synthetic datasets by the values of a chosen attribute (column) and then compute the metric on these sorted datasets. (A rationale for this procedure is given in section 3 of (9)) This adapted version of the DBRL metric is referred to as the "sorted DBRL" metric, or SDBRL for short.
- **Sorted standard deviation interval distance (SSDID).** This metric represents a variant of the Standard Deviation Interval Distance (SDID) metric. The SDID metric (20) is a commonly used method for evaluating attribute disclosure risk of perturbation methods in the Statistical Disclosure Control literature. It corresponds to the proportion of original records inside a standard deviation interval whose center is the corresponding perturbed record (where the interval width is computed in terms of a percentage p of the standard deviation of the variable). A record i in the original dataset is considered to be inside the standard deviation interval of the perturbed record i if, for all variables j, it is inside the respective standard deviation interval. Similarly to DBRL, the SDID metric also assumes the existence of a mapping between the original and perturbed values, and we adopted the sorted version of this metric (SSDID) proposed by (5; 6) in our synthetic data evaluations.

In our evaluations, we only compute the KS, ED, DCR, SDBRL, and SSDID metrics for numeric variables. (Note that our simulated datasets contain only numeric variables and, as shown in Table 4, the real-world datasets contained mostly numeric variables as well.)

H.5 Extended results

Due to space limitations, in the main text we only present experimental results pooled across all datasets. Here we provide more detailed results. Figure 11 presents the results from the simulated data experiments separated by simulation setting. Figure 12, presents separated results for the first 8 real-world datasets.

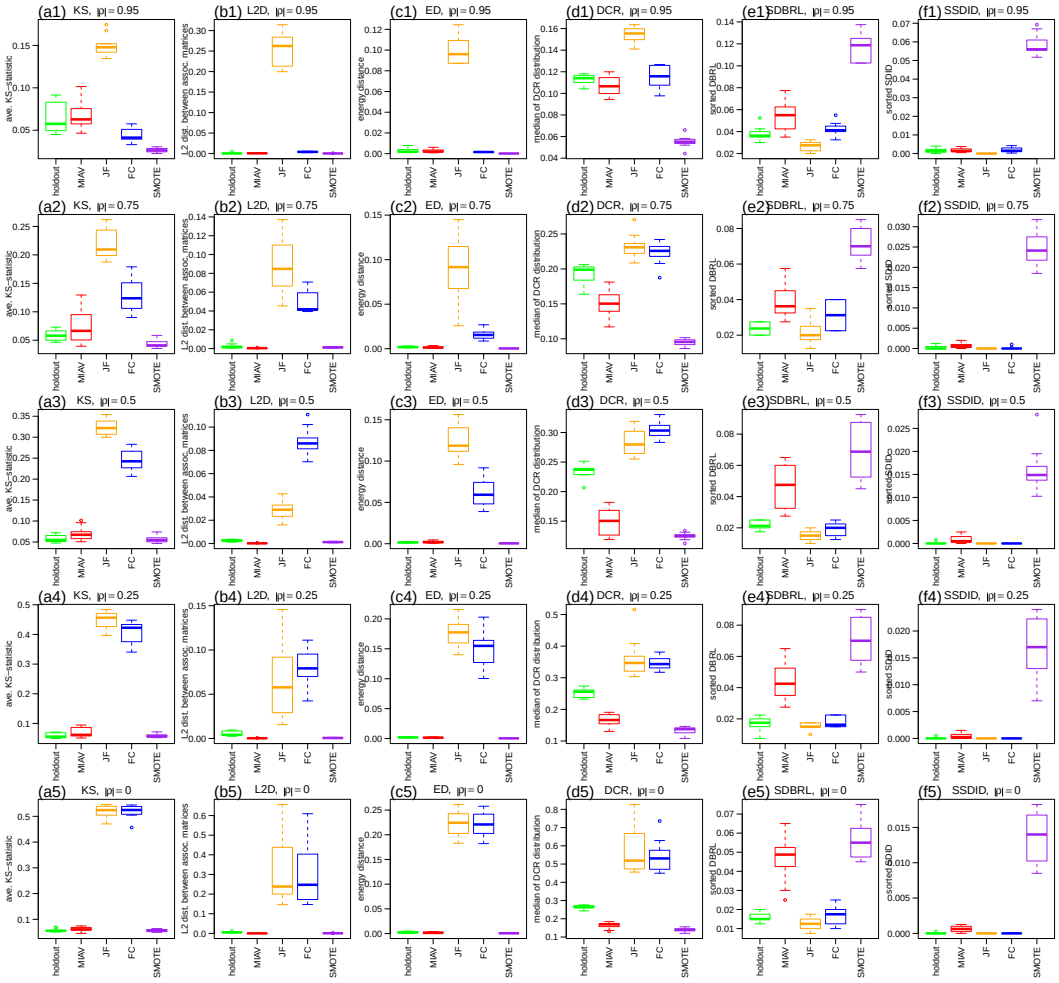

Figure 11: Simulated data experiments separated by simulation setting. Each boxplot displays the results from 10 separate replications based on different simulation parameters.

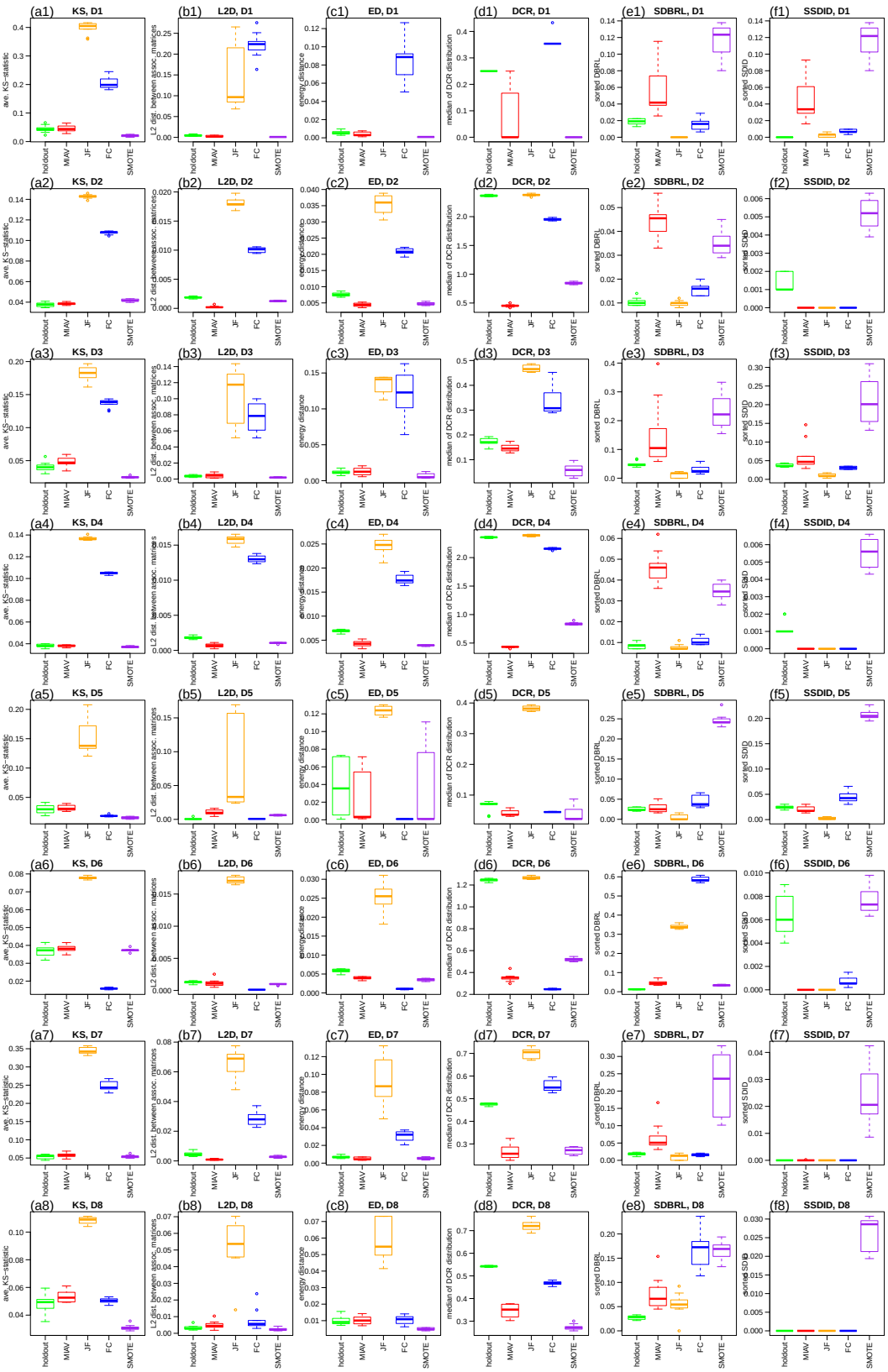

Figure 12: Experiment results for datasets D1 to D8 (see Table 4 for the original dataset names). Each boxplot displays the results from 10 distinct original/holdout data splits.